

Utilizando FXRuby na construção de cenários 3D

Ana Luiza Dias, Geraldo Clézio Oliveira Junior, Eduardo Barrére¹

PUC MINAS, Campus Poços de Caldas

analuiza.dias@uol.com.br, geraldoclezio@yahoo.com.br, barrere@pucpcaldas.br

Resumo

Este artigo apresenta uma linguagem de script orientada a objeto, chama FXRuby, que está sendo utilizada na criação de uma ferramenta para a modelagem de objetos 3D. Esta linguagem apresenta uma série de facilidades na criação e manipulação de objetos 3D, baseadas na biblioteca OpenGL, proporcionando a elaboração mais fácil e rápida de protótipos.

Palavras-Chaves: Interfaces Gráficas, FXRuby.

1. Introdução

Existem vários pacotes baseados na Biblioteca OpenGL[1] para a construção de cenários 3D. Cada pacote apresenta suas vantagens e desvantagens (facilidades na linguagem, portabilidade, tempo de execução, etc). Dentre estas possibilidades existe a linguagem FXRuby[2].

A utilização de uma linguagem adequada ao propósito do projeto e ao tempo disponível é de extrema importância, principalmente aos trabalhos de iniciação científica (normalmente de curta duração). Este foi um dos quesitos avaliados na definição da linguagem a ser utilizada no projeto “Ferramenta multi-interface para a modelagem de objetos 3D” em execução na PUC MINAS campus Poços de Caldas pelos autores deste artigo. Este projeto visa estudar a instanciação de objetos 3D de diversas formas distintas (menus, manipulação direta, etc.).

Para dar início a este trabalho foi necessário um estudo detalhado da Linguagem FXRuby (extensão da linguagem Ruby), o que gerou a proposta de descrever o fruto deste estudo para que outros pesquisadores possam conhecer, avaliar e também fazer uso da linguagem na montagem de protótipos envolvendo modelagem geométrica e manipulação 3D de forma rápida e eficiente.

Este artigo apresenta as principais características desta

linguagem no que diz respeito à programação gráfica utilizando como base a biblioteca OpenGL.

2. Linguagem Ruby

Ruby[3] é uma linguagem para criação de scripts, para programação orientada a objeto, rápida e fácil. Ela foi criada por Yukihiro Matsumoto e apresenta muitas características para processar arquivos de texto e fazer administração de sistema, podendo ser utilizada para prototipação rápida, desenvolvimento na Web, inteligência artificial e o ensino de princípios orientados a objeto.

A linguagem Ruby tece as principais características das melhores linguagens de programação[4]:

- É poderosa, pois tem a expressividade e conveniência de uma linguagem de criação de scripts, seus programas são compactos, muito legíveis, e podem ser feitos em algumas linhas sem segredo;
- É simples, visto que a sintaxe e a semântica são intuitivas e claras. Os inteiros e as classes são objetos, uma vez que se aprendem os princípios é fácil supor algo novo corretamente.
- É transparente, pelo fato de poder trabalhar com os métodos; e
- É disponível, por ser uma fonte aberta e livremente disponível para o desenvolvimento e distribuição.

Seguindo a tendência de algumas linguagens, Ruby não se restringe a uma única plataforma, é uma linguagem portátil e pode ser utilizada sob: Linux (most *NIX), Macintosh (OS 9 and OS X), OS/2, DOS, Windows 95/98/NT/2K e Sistemas Especializados como BeOS.

Ruby tem sintaxe simples e é orientada a objeto puro, isto significa que todos os dados em Ruby são objetos, tendo a habilidade para acrescentar métodos a uma classe. Ela não precisa de nenhuma declaração de variável porque usa convenções de nomenclatura simples para denotar a extensão de variáveis e pode carregar bibliotecas de extensão dinamicamente.

Em termos de linguagem, pode-se citar como características positivas da Linguagem Ruby[3]:

¹ Orientador do Projeto “Ferramenta multi-interface para a modelagem de objetos 3D”, financiado pelo FIP - PUC MINAS e em execução na PUC MINAS campus Poços de Caldas.

- Sintaxe simples;
- Consistente;
- Fácil aprendizagem;
- Bibliotecas com recursos diversos;
- Fácil instalação;
- Desenvolvimento rápido com bom desempenho;
- Pode envolver e ser envolvido pelo código de C;
- Possui escalas;
- Metaclasses;
- Manipulação de exceção;
- Segurança; e
- Tudo num programa Ruby é um objeto, não há distinção entre Int / Integer, por exemplo.

Assim como qualquer linguagem, também apresenta algumas desvantagens[5]:

- Não é tão grande e automatizada;
- Documentos ainda escassos;
- Um único livro;
- Implementação com threads não-nativos; e
- Poucos codificadores experientes.

3. Linguagem FXRuby

A linguagem FXRuby é um módulo de extensão de Ruby, que provê uma interface do FOX com biblioteca GUI. Em outras palavras, a linguagem FXRuby incorpora uma série de facilidades de controle de interação e interface a linguagem Ruby. Combinando FXRuby com o OpenGL em Ruby pode-se desenvolver aplicações gráficas 3D muito poderosas.

Desde que sua primeira liberação pública em janeiro de 2001, FXRuby tem tornado a opção mais popular do GUI para a linguagem Ruby, pois FXRuby tem um código fonte pronto, que pode ser modificado visando as necessidades do programador.

Na próxima sessão é descrita a hierarquia de classes que envolvem as classes que manipulam os cenários 3D.

4. Hierarquia GL em FXRuby

Devido ao enfoque deste trabalho, dentro da hierarquia de classes da linguagem FXRuby, é apresentada a hierarquia das classes que envolvem diretamente a manipulação de cenários 3D, ou seja, a manipulação da biblioteca OpenGL.

4.1 Hierarquia das classes

A figura 1 apresenta a hierarquia de classes que envolvem o manipulação de cenários 3D em FXRuby, destacando-se as classes FXGL*** (em cinza) que utilizam os recursos do pacote "opengl" fornecido juntamente com a linguagem.

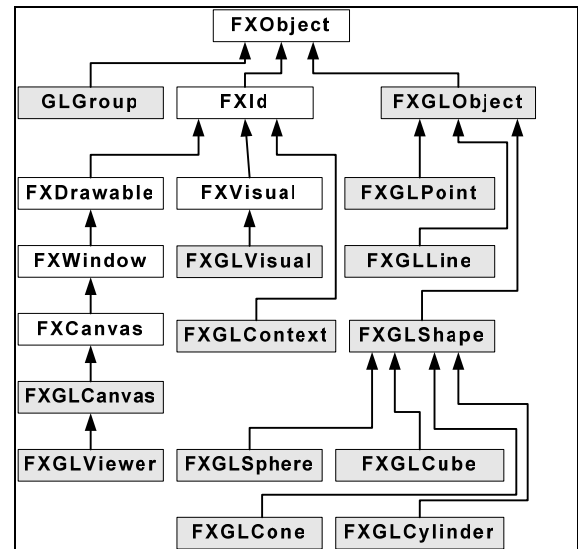


Figura 1-Hierarquia de Classes FXGL***

Estas classes podem ser divididas em 3 grupos principais:

- Grupo de visualização: engloba a classe FXId e as classes filhas. Este grupo é responsável pela apresentação dos elementos na tela e suas interações;
- Grupo de objetos: engloba a classe FXGLObject e suas classes filhas. São utilizadas para definirem os objetos que irão compor a cena; e
- Agrupamento de objetos: a classe GLGroup serve para agrupar os objetos a serem inseridos na área de projeção e interação.

Citando como exemplo de uma classe, apresenta-se abaixo a implementação da classe FXGLSphere, que possibilita a criação de objetos no formato de uma esfera. Neste exemplo vale ressaltar a herança da classe FXGLShape e o uso do pacote OpenGL (GL e GLU[1][6]).

```

class FXGLSphere < FXGLShape
  SLICES_NUMBER = 20
  STACKS_NUMBER = 20
  attr_accessor :radius
  attr_accessor :slices
  attr_accessor :stacks

  def initialize(*args)
    if args.length == 4
      super(args[0], args[1], args[2],
SHADING_SMOOTH|STYLE_SURFACE)
    else
      super(args[0], args[1], args[2],
SHADING_SMOOTH|STYLE_SURFACE, args[4],
args[4])
    end
    @radius = args[3] ? args[3] : 1.0
  end
end
  
```

```

@slices = SLICES_NUMBER
@stacks = STACKS_NUMBER
setRange(FXRangef.new(-@radius, @radius,
@radius, @radius, -@radius, @radius))
end

def drawshape(viewer)
quad = GLU::NewQuadric()
GLU::QuadricDrawStyle(quad, GLU::FILL)
GLU::Sphere(quad, @radius, @slices, @stacks)
GLU::DeleteQuadric(quad)
end
end

```

4.2 Descrição das classes

Nesta sessão são descritas as classes implementadas no FXRuby que fazem referência aos recursos de OpenGL.

As classes abaixo instanciam as telas onde são criados os objetos 3D. A classe FXGLViewer oferece vários recursos de interação com os objetos.

| |
|---|
| Nome da Classe: FXGLViewer. |
| Herança: FXGLCanvas. |
| Descrição da Classe: Área de visualização do objeto. |
| Atributos: worldPix, modelPix, viewport, Material, fieldOfView, Zoom, distance, Scale, Orientation, Center, eyeVector, eyePosition, helpText, tipText, transform, invTransform, scene, selection, projection, ambientColor, maxHits, turboMode, light. |
| Métodos: FXGLViewer.objectType, FXGLViewer.objectTypeName, Initialize, initialize, Lasso, setBounds, fitToBounds, eyeToScreen, ScreenToEye, screenToTarget, worldToEye, WorldToEyeZ, eyeToWorld, worldVector, ranslate, GetBoreVector, def locked?, readPixels, ReadFeedback, doesTurbo?, turboMode?, etBackgroundColor, getBackgroundColor. |

| |
|--|
| Nome da Classe: FXGLCanvas. |
| Herança: FXCanvas. |
| Descrição da Classe: Tela, área do objeto. |
| Atributos: ----- |
| Métodos: initialize, initialize, shared?, context, makeCurrent, makeNonCurrent, current?, swapBuffers |

A classe FXGLShape é utilizada para modelagem de objetos de modo rápido e eficaz.

| |
|---|
| Nome da Classe: FXGLShape. |
| Herança: FXGLObject. |
| Descrição da Classe: Modela o objeto, opção de forma de desenho. |
| Atributos: tipText, position |

| |
|---|
| Métodos: drawshape, initialize, initialize, getMaterial, setRange. |
|---|

A classe FXGLObject possui as definições de um objeto OpenGL básico.

| |
|---|
| Nome da Classe: FXGLObject. |
| Herança: FXObject. |
| Descrição da Classe: Define um objeto OpenGL básico. |
| Atributos: ----- |
| Métodos: Initialize, copy, bounds, draw, hit, canDrag, anDelete, drag, identify. |

A classe FXGLVisual faz o controle da visualização do objeto no sistema de vídeo do computador.

| |
|---|
| Nome da Classe: FXGLVisual. |
| Herança: FXVisual. |
| Descrição da Classe: Controle do sistema de vídeo, "Monitor", com a aplicação desenvolvida. |
| Atributos: redSize, greenSize, blueSize, alphaSize, depthSize, stencilSize, accumRedSize, accumGreenSize, accumBlueSize, accumAlphaSize, actualRedSize, actualGreenSize, actualBlueSize, actualAlphaSize, actualDepthSize, actualStencilSize, actualAccumRedSize, actualAccumGreenSize, actualAccumBlueSize, actualAccumAlphaSize. |
| Métodos: initialize, FXGLVisual.supported?, doubleBuffered?, stereo?, accelerated?, bufferSwapCopy?. |

A classe GLGroup permite agrupar todos os objetos 3D criados, em um único objeto facilitando assim a manipulação dos mesmos e a instanciação deles na tela de visualização, ou seja, na Viewer.

| |
|--|
| Nome da Classe: GLGroup. |
| Herança: FXGLObject. |
| Descrição da Classe: Grupo de objetos OpenGL. |
| Atributos: FLT_MAX, FLT_MIN, @list, |
| Métodos: Initialize, size, [](pos), []=(pos, obj), each_child, bounds, draw, hit, identify, canDrag, drag, insert, prepend, append, replace, remove, clear. |

A classe FXGLContext possui as informações sobre os objetos OpenGL.

| |
|---|
| Nome da Classe: FXGLContext. |
| Herança: FXId. |
| Descrição da Classe: Representa informações sobre um objeto OpenGL. |
| Atributos: Visual. |
| Métodos: initialize, initialize, shared?, begin, end, swapBuffers, swapSubBuffers. |

As classes abaixo FXGLPoint, FXGLCube, FXGLLine, FXGLCone, FXGLCylinder e FXGLSphere utilizam o módulo OpenGL para instanciação de objetos 3D, tais como, ponto, cubo, linha, cone, cilindro e esfera.

| |
|--|
| Nome da Classe: FXGLPoint. |
| Herança: FXGLObject. |
| Descrição da Classe: Desenha ponto na tela. |
| Atributos: pos, @pos. |
| Métodos: initialize, bounds, draw, hit. |

| |
|--|
| Nome da Classe: FXGLCube. |
| Herança: FXGLShape. |
| Descrição da Classe: Opção de desenhar um cubo. |
| Atributos: Width, height, depth. |
| Métodos: initialize, drawshape. |

| |
|--|
| Nome da Classe: FXGLLine . |
| Herança: FXGLObject. |
| Descrição da Classe: Opção para desenhar linha. |
| Atributos: Fm, to. |
| Métodos: initialize, bounds, draw, hit. |

| |
|---|
| Nome da Classe: FXGLCone. |
| Herança: FXGLShape. |
| Descrição da Classe: opção para instanciar um cone. |
| Atributos: SLICES_NUMBER, STACKS_NUMBER, LOOPS, height, radius, slices, stacks, loops. |
| Métodos: initialize, drawshape. |

| |
|---|
| Nome da Classe: FXGLCylinder . |
| Herança: FXGLShape. |
| Descrição da Classe: Opção para instanciar um cilindro. |
| Atributos: SLICES_NUMBER, STACKS_NUMBER, LOOPS, height, radius, slices, stacks, loops. |
| Métodos: initialize, drawshape. |

| |
|---|
| Nome da Classe: FXGLSphere . |
| Herança: FXGLShape. |
| Descrição da Classe: Opção para instanciar um esfera.. |
| Atributos: SLICES_NUMBER, STACKS_NUMBER, radius, slices, stacks. |
| Métodos: initialize, drawshape. |

5. Utilização das classes GL

Como exemplo de uso da linguagem FXRuby, optou-se por apresentar um cubo. Este cubo, conforme ilustra a figura 2, foi implementado de duas maneiras distintas: primeiro fazendo uso direto do pacote OpenGL e depois utilizando-se das classes FXGL***. Abaixo são apresentados os dois códigos e seus respectivos comentários.



Figura 2: Execução do exemplo

O código abaixo cria um cubo utilizando diretamente as bibliotecas OpenGL e GLU, onde todas as faces do cubo são definidas através de um vetor de faces, assim como a posição também é definida por um vetor de posições. O método “drawBox” faz toda a construção do cubo utilizando os comandos nativos do opengl. No método “myinit” é definido as configurações do ambiente, ou seja, tipo de projeção, material, perspectiva, translação, rotação, etc.

```
require "opengl"
require "glut"

$light_diffuse = [1.0, 0.0, 0.0, 1.0]
$light_position = [1.0, 1.0, 1.0, 0.0]
$n = [[-1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [1.0, 0.0, 0.0],
[0.0, -1.0, 0.0], [0.0, 0.0, 1.0], [0.0, 0.0, -1.0] ]
$faces = [[0, 1, 2, 3], [3, 2, 6, 7], [7, 6, 5, 4],
[4, 5, 1, 0], [5, 6, 2, 1], [7, 4, 0, 3] ]
$v = 0
def drawBox
  for i in (0..5)
    GL.Begin(GL::QUADS);
    GL.Normal(*($n[i]));
    GL.Vertex($v[$faces[i][0]]);
    GL.Vertex($v[$faces[i][1]]);
    GL.Vertex($v[$faces[i][2]]);
    GL.Vertex($v[$faces[i][3]]);
    GL.End()
  end
end
display = Proc.new {
  GL.Clear(GL::COLOR_BUFFER_BIT
GL::DEPTH_BUFFER_BIT);
```

```

drawBox
  GLUT.SwapBuffers
}
def myinit
  $v = [[-1, -1,1],[-1, -1,-1], [-1,1,-1], [-1,1,1], [1, -1,1],
        [1, -1,-1], [1, 1,-1], [1,1,1]]
  GL.Light(GL::LIGHT0, GL::DIFFUSE,
$light_diffuse);
  GL.Light(GL::LIGHT0, GL::POSITION,
$light_position);
  GL.Enable(GL::LIGHT0);
  GL.Enable(GL::LIGHTING);
  GL.Enable(GL::DEPTH_TEST);
  GL.MatrixMode(GL::PROJECTION);
  GLU.Perspective(40.0, 1.0, 1.0, 10.0);
  GL.MatrixMode(GL::MODELVIEW);
  GLU.LookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0)
  GL.Translate(0.0, 0.0, -1.0);
  GL.Rotate(60, 1.0, 0.0, 0.0);
  GL.Rotate(-20, 0.0, 0.0, 1.0);
end
GLUT.Init
GLUT.InitDisplayMode(GLUT::DOUBLE
GLUT::RGB | GLUT::DEPTH);
GLUT.CreateWindow("Cubo");
GLUT.DisplayFunc(display);
myinit
GLUT.MainLoop();

```

Todo o processo de construção do cubo está preso aos comandos puro do opengl, desta forma pode-se perceber as dificuldades de implementação de objetos 3D. Vale ressaltar que todas as partes de interação direta com o objeto ainda devem ser implementadas. Já se trabalharmos com as classes do FXRuby essas implementações já estão prontas, porém caso haja a necessidade de alguma alteração vale lembrar que os códigos dessas classes são abertos e o próprio programador pode alterá-los de acordo com suas necessidades a qualquer momento.

O código abaixo implementa o cubo através das classes FXRuby, obtendo-se o mesmo resultado visual do código anterior, com a diferença de que as classes FXRuby já trazem implementadas as manipulações direta do objeto, tais como: translação, rotação, etc; facilitando a sua implementação. Ao instanciar o objeto “glvisual” já é definido o tipo de visualização, assim como ao instanciar o objeto “grupo”, onde será adicionado a ele todos os objetos 3D através do método “grupo.append”, tais como, cubo, esfera, cone, etc. Ou seja nele são agrupados todos os objetos 3D, que são armazenados em um vetor facilitando portanto a manipulação desses objetos.

```

require 'fox12'
require 'fox12/responder'
require 'fox12/colors'

```

```

require 'fox12/glshapes'
require 'fox12/missingdep'
include Fox

class TesteManipulacao < FXMainWindow
  def initialize(app)
    super(app, "Cubo", nil, nil, DECOR_ALL, 0, 0, 250,
250)
    glvisual = FXGLVisual.new(getApp(),
VISUAL_DOUBLEBUFFER)
    grupo = FXGLGroup.new
    grupo.append(FXGLCube.new(-1.0,-1.0,0.2,0.2,0.2,
0.2))
    viewer = FXGLViewer.new(self, glvisual, nil,0,
LAYOUT_FILL_X|LAYOUT_FILL_Y|LAYOUT_TOP
|LAYOUT_LEFT)
    viewer.scene = grupo
    viewer.projection = 0
    viewer.zoom = 0.3
    viewer.ambientColor = [1.0, 0.0, 0.0, 1.0]
    teste = FXQuatf.new(0.62158,-0.115765,-
0.382987,0.673467)
    viewer.orientation = teste
  end

  def create
    super
    show(PLACEMENT_SCREEN)
  end
end

if __FILE__==$0
  application = FXApp.new("GLTest", "FoxTest")
  application.disableThreads
  TesteManipulacao.new(application)
  application.create
  application.run
end

```

O objeto “viewer” é a tela de visualização dos objetos 3D. A ele é atribuído o objeto “grupo”, carregando assim todos os objetos 3D instanciados de forma ágil e eficaz. O objeto “viewer” ainda traz boa parte da implementação para interação e manipulação dos objetos 3D.

As classes do FXRuby fornecem vários métodos e atributos, onde a utilização deles é fácil e direta. A utilização desses métodos e atributos, tornam bem mais fácil a modificação e a visualização dos resultados nesses objetos. Por exemplo, “viewer.zoom = 0.3”, onde pode-se definir o zoom com apenas um comando de atribuição.

6. Trabalhos futuros

Com o desenvolvimento das ferramentas para a área de computação gráfica através do FXRuby, tem-se como trabalhos futuros alguns desafios a serem resolvidos

como, por exemplo, fazer a interação entre os objetos, estruturar a cena, controlar os atributos dos objetos, resolver problemas de visibilidade, suportar diversos dispositivos gráficos e fazer programas independentes do sistemas operacionais. Para resolver esses e outros problemas é importante que os softwares para a área de computação gráfica tenham uma interface que permita ao usuário ter acesso às suas funcionalidades de forma mais simples e natural possível. Visando colaborar na solução desses desafios, o projeto em questão propõe obter dados que auxiliem na verificação de opções de interface mais adequada a cada funcionalidade de computação gráfica. A ferramenta a ser desenvolvida, possibilitará ao usuário configurar qual o tipo de interface irá utilizar na modelagem de objetos 3D.

7. Conclusão

As linguagens de script são conhecidas por serem de fácil utilização, mas nem sempre com recursos importantes de programação. Já a linguagem Ruby, apresenta todas as facilidades de uma linguagem de script, com o poder da orientação a objetos, o que proporciona maior flexibilidade ao programador. Além disso, o fato de se ter o código fonte das classes a disposição, permite melhor entendimento e flexibilidade para a linguagem, podendo esse código ser modificado se necessário.

Para o desenvolvimento de cenários 3D, seja em protótipos de uma aplicação específica ou apenas de forma didática, o pacote FXRuby possibilita utilizar a biblioteca OpenGL (e suas derivadas) da forma convencional ou através das classes FXGL****, trazendo ao programador a possibilidade de escolha entre os recursos disponíveis pelas bibliotecas em várias linguagens ou na simplicidade de uso proporcionada pela FXRuby, onde entre outros pontos, destaca-se o controle de interação (teclado e mouse) ou algumas funcionalidades (projeção, forma de exibição do objeto, etc.) já herdadas de outras classes.

No projeto em desenvolvimento na PUC MINAS, o uso da linguagem FXRuby vem se mostrando muito eficaz para a realização de testes intermediários e resolução rápida de pequenas soluções.

8. Referências

[1] The Industry's Foundation for High Performance Graphics from games to virtual reality, mobile phones to supercomputers <http://www.opengl.org/>, visited 06/2005.

[2] Lyle Johnson, FXRuby, <http://www.fxruby.org/>, visited in 06/2005.

[3] Ruby Central... the Source for Ruby, <http://www.rubycentral.com/>, visited in 06/2005.

[4] The Object-Oriented Programming Language Ruby, <http://www.ruby-lang.org/en/>, visited in 06/2005.

[5] Dave Thomas and Andy Hunt, Programming un Ruby, <http://www.ddj.com/documents/s=871/ddj0101b/>, visited 06/2005.

[6] Paul Rademacher, GLUI User Interface Library <http://www.cs.unc.edu/%7Erademach/glui/>, visited 06/2005.