3
Integração da Modelagem Paramétrica no MG

A modelagem paramétrica permite descrever as propriedades de um determinado modelo por meio de descritores geométricos conhecidos como parâmetros de forma. Estes descritores poderiam ser utilizados para oferecer ao modelador geométrico uma maior abrangência na fase de modelagem.

Como foi dito anteriormente as abordagens de modelagem apresentadas por Oliveira [Oliveira08] e Mendonça [Mendonça04] criam uma particularização do processo de modelagem automática pois fazem uso de um ambiente externo ao MG, modelador geométrico utilizado neste trabalho.

Uma desvantagem da abordagem de Oliveira [Oliveira08] é o uso necessário e obrigatório de um ambiente externo de programação, para poder gerar os arquivos que representam o modelo geométrico. É necessário enfatizar que, além de conhecer os parâmetros de forma, precisa-se também conhecer a sintaxe ou formato padrão de arquivo geométrico que o MG reconhece, onde são armazenadas todas as entidades geométricas criadas. A visualização fica restrita ao programa MG.

É necessário mencionar que cada *script* da abordagem apresentada por Oliveira [Oliveira08] gera um arquivo geométrico do MG (casco, tanques internos, super-estrutura, etc). Dependendo da complexidade do modelo (figura 3.1) é necessário utilizar a opção *Append Files* que o MG oferece de forma a juntar num único arquivo geométrico o modelo completo a ser estudado.

A abordagem apresentada por Mendonça [Mendonça04] também faz uso de um ambiente externo para definir a forma do modelo de acordo aos parâmetros escolhidos, os quais serão otimizados. O ambiente de modelagem e o ambiente de otimização estão interconectados por meio de um *script*. Isto é, caso o modelo não satisfaça as restrições impostas pelo usuário, um novo modelo é gerado e o processo de otimização recomeça.

Pode-se perceber que os trabalhos anteriores que tratam da automatização do processo de modelagem não esgotam esta questão pois fazem uso de um ambiente externo e particular ao ambiente de modelagem para resolver determinado problema.

PUC-Rio - Certificação Digital Nº 0421097/CA

Desta forma apresenta-se uma generalização do problema de modelagem paramétrica de forma automática, utilizando a linguagem Lua [Ierusalimschy04] e o MG, onde é possível definir, reproduzir, sintetizar um *script* de modelagem a partir de um arquivo geométrico do MG. Este *script* é utilizado para modificar automaticamente a forma de um modelo e posteriormente realizar a sua correspondente análise estática e/ou dinâmica.

3.1 Modelagem por Scripts

Cada vez mais, grandes aplicações de software de diversas áreas estão adotando linguagens de extensão, como forma de aumentar a sua flexibilidade e o seu poder de expressão. Linguagens de extensão permitem que um usuário configure uma aplicação, definindo macros ou *scripts* que modifiquem a entrada de dados, descrição da interface, etc.

Existem muitas ferramentas de modelagem na área de engenharia que possuem um ambiente de programação via *scripts*. Dentro das mais conhecidas ferramentas temos: Ansys [Ansys09], AutoCad [AutoCad09], MSC/Patran [Patran09], entre outros.

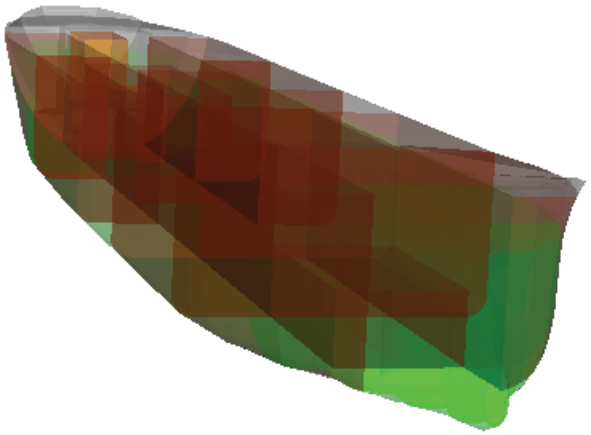


Figura 3.1: Modelo geométrico de um navio.

Devido à enorme complexidade em descrever e/ou modelar uma estrutura flutuante (figura 3.1) é conveniente dispor de um mecanismo de modelagem de forma automática, ou quase automática. Deve ser possível, através de uma linguagem de programação, modelar (geração de scripts), disponibilizar um histórico do processo de modelagem como um todo, definir um conjunto de variáveis que possam ser utilizados como parâmetros de entrada para as dimensões de um modelo, etc.

PUC-Rio - Certificação Digital Nº 0421097/CA

Para garantir estas funcionalidades será usada uma linguagem de extensão.

3.2 Linguagens de Extensão

Linguagens de extensão, ou *script*, geralmente são linguagens interpretadas, têm tipagem dinâmica e gerenciamento automático de memória, fornecem facilidades para a construção de estruturas de dados dinâmicos. Essas linguagens funcionam acopladas a programas hospedeiros, implementados em linguagens compiladas tradicionais, como C e C++.

As linguagens de extensão ou *scripts* podem ser classificados de acordo com sua complexidade [Ierusalimschy04]:

Linguagens de configuração servem para selecionar preferências e são uma lista de valores associados às variáveis. Um exemplo típico são os arquivos .ini do windows.

Linguagens de macros servem para automação de tarefas e são usualmente uma lista de ações de automação de conexões internet via modem.

Linguagens embutidas permitem acesso programável aos serviços da aplicação, com fluxo de controle completo e funções definidas pelo usuário a partir de primitivas exportadas pela aplicação. Essas linguagens são, muitas vezes, variantes simplificadas de linguagens tradicionais como Lisp ou C.

Independente da complexidade da linguagem, a adoção de uma linguagem de extensão é uma técnica poderosa de desenvolvimento de software, pois permite que muitos aspectos da aplicação sejam controlados externamente. Isso torna o desenvolvimento mais ágil, pois permite que partes importantes da aplicação sejam desenvolvidas por outros membros da equipe.

Estas linguagens são usualmente implementadas como bibliotecas e podem ser ligadas a qualquer aplicação que precise de uma linguagem de extensão. Tais linguagens têm como proposta fundamental serem pequenas e abrangentes o suficiente para servirem a todos os tipos de aplicação sem se prender a nenhum deles.

Em princípio, qualquer linguagem de extensão, tal como Phyton [Rossum03], Tcl [Ousterhout94] ou Lua [Ierusalimschy04] poderia ser utilizada para implementar uma linguagem de comandos e *scripts* que seja capaz de estender o poder de expressão do modelador geométrico aqui utilizado. A modelagem geométrica por *scripts* proposta neste trabalho será

PUC-Rio - Certificação Digital Nº 0421097/CA

implementada utilizado a linguagem Lua, devido à simplicidade de sua sintaxe e à facilidade de acoplar esta linguagem a um ambiente de modelagem já existente.

3.3 Linguagem Lua

Lua é uma linguagem de *script* extensível, projetada para oferecer meta-mecanismos que possibilitam a construção de objetos mais específicos. Com isso, é fácil adequar Lua às necessidades da aplicação, sem comprometer as suas características básicas, mantidas desde a sua criação, tais como portabilidade, pequeno tamanho e simplicidade. Lua foi projetada para estender aplicações como também para ser usada como uma linguagem de propósito geral [Lua09] e [Ierusalimschy04].

Esta linguagem é adotada para estender o programa MG (Mesh Generator) de forma a poder acrescentar um ambiente de modelagem por linha de comandos (definição de um conjunto básico de comandos de modelagem).

Lua fornece um ambiente de programação procedural com poderosas construções para descrição de dados. Esta descrição se baseia no uso de tabelas associativas e semântica extensível que o Lua fornece, e que permite poder criar *scripts*. Um conjunto de *scripts* pode realizar/automatizar diversas tarefas de modelagem. Todas estas características fazem de Lua uma linguagem ideal para configuração, automação (*scripting*) e prototipagem rápida.

Lua foi projetada e implementada no Tecgraf, o Grupo de Tecnologia em Computação Gráfica da PUC-Rio. A motivação para a criação de Lua no Tecgraf foi a necessidade crescente das suas aplicações de serem externamente configuráveis pelos usuários.

Desta forma, diversos aspectos essenciais das aplicações podem ser modificados sem recompilar a aplicação. Isto é, a linguagem Lua pode ser utilizada para modificar a interface do programa de forma a mostrar dados que o usuário deseja visualizar de acordo com a situação corrente.

No presente trabalho, Lua é utilizada com um fim distinto, pois pretende-se estender as funcionalidades do MG, de forma a gerar um histórico de comandos (*script*) da modelagem efetuada pelo usuário e incorporar um ambiente de programação que permite estender a modelagem geométrica do MG.

A sintaxe de Lua é semelhante à da linguagem C, mas com terminação explícita de comandos. Incorpora várias facilidades usualmente encontradas em linguagens procedurais como comandos de controle de fuxo, definição de sub-rotinas, atribuições, operadores aritméticos, etc.

PUC-Rio - Certificação Digital Nº 0421097/CA

Há 8 tipos de dados definidos em Lua, a saber: *nil*, *boolean*, *number*, *string*, *function*, *userdata*, *thread* e *table*. Outro mecanismo importante da linguagem é o de *fallbacks*. Semanticamente, o mecanismo de *fallback* pode ser comparado a um mecanismo de tratamento de exceção, e pode ser usado para a criação de recursos que o a linguagem Lua não oferece, como por exemplo a herança e criação de objetos. Mais detalhes podem ser encontrados em [Ierusalimschy04].

É importante mencionar que o custo, em termos de *bytes* que será acrescido ao programa executável final, devido a incorporação da linguagem Lua, é mínimo.

3.4 Ambiente de Programação

A seguir mostra-se o ambiente de programação e linha de comandos incorporado no MG (figura 3.2).

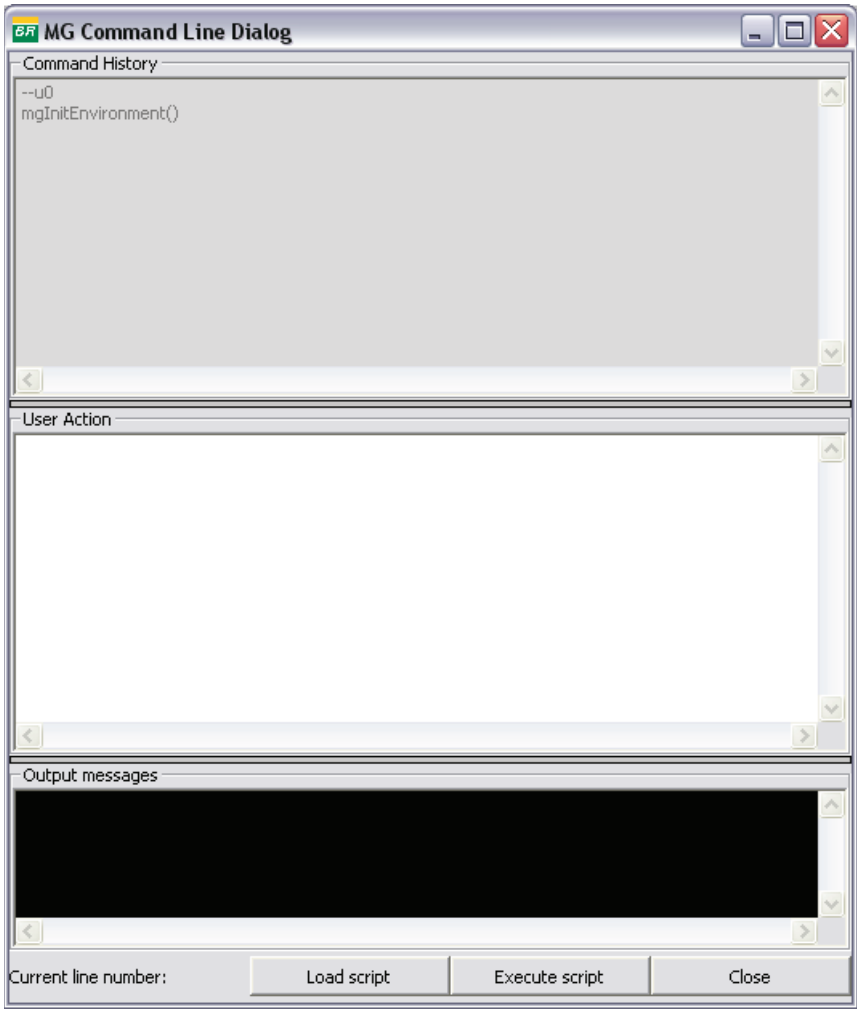


Figura 3.2: Interface MG com linha de comandos.

PUC-Rio - Certificação Digital Nº 0421097/CA

A interface apresentada na figura 3.2 é uma grande contribuição para a modelagem automática por *scripts* incorporada no MG.

Este ambiente é utilizado para criar variáveis, capturar as ações de modelagem do usuário em forma de comandos, executar um conjunto de comandos e sintetizar a geometria de um arquivo do MG em forma de um *script*.

O ambiente de linha de comandos está dividido em três partes: uma janela chamada de *Command History*, onde são armazenadas todas as ações executadas pelo modelador e que foram transformadas em um conjunto de comandos que reproduzem um modelo qualquer via *script*. Há ainda uma outra janela chamada de (*User Action*), onde o usuário insere um conjunto de comandos ou *scripts* a serem executados.

Finalmente, uma janela chamada de *Output Messages* onde são mostradas mensagens de saída definidas pelo usuário ou pela ação de determinado comando, como por exemplo: mensagens de erro, visualização do valor de determinada variável, etc. É possível ainda carregar um *script* na janela *User action* e depois executá-lo.

Qualquer comando ou sub-rotina em Lua, é definido e executado em um único **ambiente global** [Lua09]. Este ambiente é inicializado automaticamente quando o interpretador Lua é ativado e persiste enquanto estiver rodando. Em outras palavras, enquanto o MG estiver ativo, o ambiente de programação Lua também será ativado de forma transparente.

O ambiente global definido pela linguagem Lua, pode ser considerado o ambiente de programação implícita de modelagem por *scripts* do MG, onde o usuário pode vir a programar ações, da mesma forma que poderia ser feito utilizando uma outra linguagem qualquer, como por exemplo C.

Uma facilidade que a linguagem Lua oferece quando utilizado de forma embarcada para estender uma aplicação é a possibilidade de que todas as ações (variáveis, definição de funções, comandos, etc) do usuário sejam lembradas pelo MG independente do tipo de modelagem. As variáveis podem ser utilizadas ao longo do processo de modelagem para definir por exemplo a altura de uma plataforma, o raio de bojo de um navio, etc.

Na seguinte seção descreve-se o conjunto de comandos em Lua, definidos no MG, para geração de *scripts*. Estes comandos são divididos em dois tipos:

Comandos de Modelagem Baseados em Históricos Estes comandos são gerados de forma automática pelo MG enquanto o usuário está fazendo a modelagem de um determinado modelo.

PUC-Rio - Certificação Digital Nº 0421097/CA

Comandos de Modelagem Baseados em *Scripts* Estes comandos são definidos manualmente pelo usuário e geralmente são criados utilizando um editor de textos qualquer.

3.5 Comandos de Modelagem Baseados em Históricos

Devido à escolha da linguagem Lua para estender o MG (*Mesh Generator*), por meio de uma interface de linha de comandos, é necessário diferenciar os comandos de modelagem, que descrevem a criação de uma entidade geométrica do MG, dos comandos nativos próprios da linguagem Lua.

Os comandos que referem-se ao processo de modelagem têm o seguinte padrão de sintaxe: iniciam-se com as letras mg seguidos da ação que executarão (Ex: mgSetMeshType, mgSetSurfaceType,..,etc.). Desta forma, é possível gerar de forma automática e transparente para o usuário um histórico da modelagem de determinando modelo. Este histórico de modelagem permite que o usuário possa alterar determinados parâmetros (por exemplo, altura, profundidade, espessura e outros atributos) e controlar, assim, a resposta da modelagem.

É possível implementar herança em Lua e, portanto, podem-se criar objetos em Lua para as entidades básicas do MG (vértice, curva, superfície e volume). Estas entidades herdam funções previamente definidas conforme exemplo abaixo:

$$vertex : TranslateX(DeltaX)$$

O comando acima, executa a função TranslateX que aplica uma translação para a entidade vértice, com um valor de translação definido pela variável DeltaX.

Uma descrição detalhada dos comandos de modelagem implementados no modelador geométrico utilizado neste trabalho pode ser vista no Apêndice B.

3.6 Comandos de Modelagem Baseados em Scripts

Os comandos citados na seção anterior são gerados de forma automática e transparente, a fim de gerar um histórico da modelagem. Também foi necessário implementar um conjunto de objetos Lua que abstraia cada entidade do MG (vértice, curva, superfície e volume). Estes comandos têm uma sintaxe diferente dos comandos citados anteriormente e podem ser utilizados de forma independente ou em conjunto com os comandos baseados em histórico de modelagem.

PUC-Rio - Certificação Digital Nº 0421097/CA

As classes criadas são:

- mgVertex,
- mgCurve,
- mgSurface e,
- mgTank.

Estas novas classes possibilitam a criação de vértices, curvas, superfícies e volumes, prescindindo da interface do MG. Além disso, é possível realizar sobrecarga de operadores entre elas. Isto é, pode-se criar um novo vértice a partir da soma de outros vértices passados como parâmetros. Uma curva pode receber, como parâmetros, objetos do tipo vértice, além de dados numéricos referentes ao id de cada vértice e um volume pode receber, como parâmetros, objetos do tipo superfície ou o id correspondente de cada superfície.

É possível ainda modificar parâmetros internos de cada objeto fazendo uso dos metamétodos *__index* e *__newindex* que o Lua fornece. O metamétodo *__index* permite acessar os campos internos de um determinado objeto e se desejado imprimir estes valores na tela enquanto que o metamétodo *__newindex* permite acessar os campos internos do objeto e modificar os seus valores internos. Isto pode ser muito útil na programação por *scripts*, pois oferece ao usuário a possibilidade de mudar os valores internos dos objetos em tempo de execução.

Uma descrição detalhada da sintaxe de cada comando listado acima é apresentada no Apêndice C.

3.7 Exemplos de Uso dos Comandos de Modelagem

A seguir são apresentados alguns exemplos, onde é possível visualizar o uso dos comandos e modelagem em Lua incorporados no MG.

```
--u0
mgInitEnvironment()
mgCreateBox(10,10,10,0,0,0,0.0000000,0.0000000,0.0000000)
Box=mgCreateVolume("0 Volume","Hull_Volumes",0,5.0000000,5.0000000,-5.0000000,0.000000
print(Box)
mgEnableCopies(1)
for i=1,1,10 do
 Box:TranslateX(10)
end
```

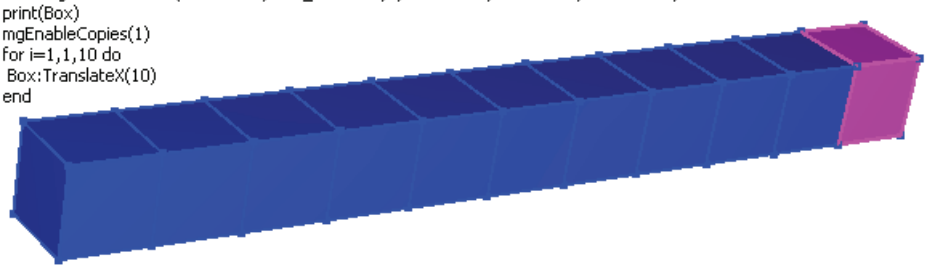


Figura 3.3: Resultado do script de modelagem 1.

PUC-Rio - Certificação Digital Nº 0421097/CA

A figura 3.3 mostra o *script* de modelagem de uma caixa, obtida através da geração de um volume (*mgCreateBox* e *mgCreateVolume*) que é posteriormente transladado e copiado 10 vezes no eixo X, utilizando controle de *loops* da linguagem Lua.

```
mgSelect("all")
messer=mgCreateVolume("0 Volume","Hull_Volumes",0,10.0000000,4.0000000,0.0000000,
4.7474970,1.7306522,1.2221438,64,128,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,
38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,
61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,
85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,
107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,
123,124,125,126,127)
messer:ScaleXYZ(0.5,0.5,0.5)
```

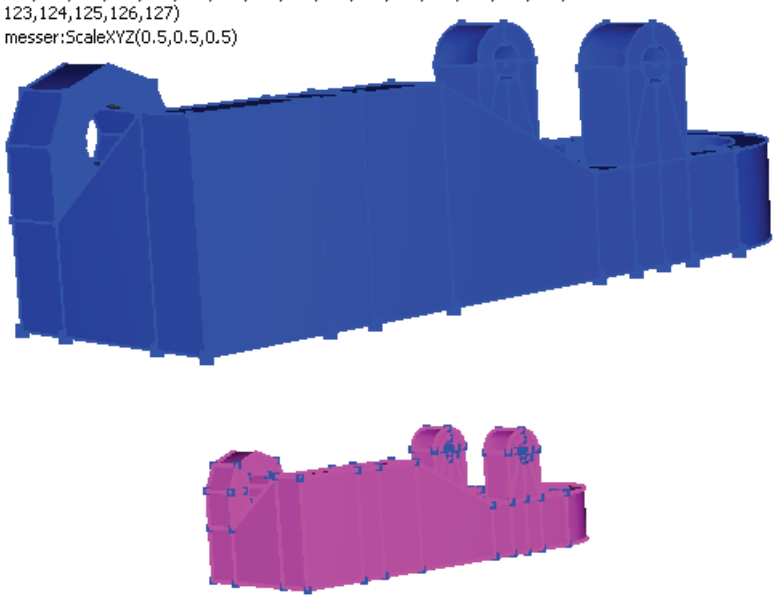


Figura 3.4: Modelo escalado em XYZ.

A figura 3.4 mostra o *script* de modelagem realizado em um modelo carregado no MG, composto apenas por superfícies. São selecionadas todas as entidades (*mgSelect("all")*) e depois é criado um volume chamado de *messer*. Os parâmetros do comando *mgCreateVolume* são obtidos diretamente de todas as entidades selecionadas. Este volume, depois de criado, foi escalado nos eixos X,Y,Z com um fator de 0.5 em cada eixo. Isto é, o volume será reduzido na metade do seu tamanho original.

É também possível utilizar as variáveis, definidas em Lua, através das interfaces existentes no MG, onde o usuário necessite informar parâmetros numéricos como comprimento, largura, altura etc de um determinado modelo.

A figura 3.5 mostra o uso destas variáveis definidas pelo usuário como parâmetros de entrada para a transformação de escala que poderia ser aplicada ao modelo da figura 3.4. É possível ainda compor estas variáveis com qualquer operação matemática como mostrado na figura acima.

PUC-Rio - Certificação Digital Nº 0421097/CA

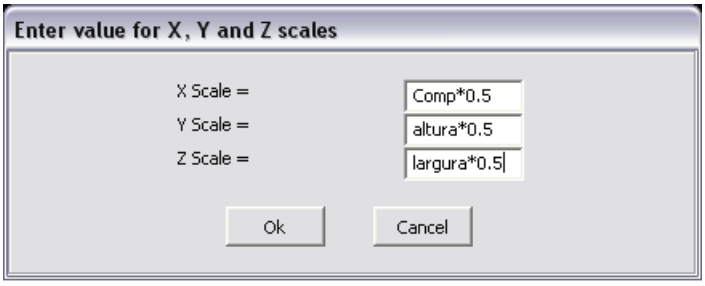


Figura 3.5: Interface para Escala utilizando variáveis.

A figura 3.6 mostra a criação de vértices e linhas utilizando os comandos *mgVertex* e *mgCurve*, onde pode-se perceber que é possível sobrecarregar os parâmetros de cada comando. Nesta figura são criados 4 vértices e uma curva. O primeiro vértice é criado na origem. Os vértices a e b são criados especificando-se dados numéricos, porém o vértice c é criado do resultado da multiplicação do vértice b por um escalar. Finalmente, a curva d, que é do tipo spline, é criada interpolando os pontos a, b, e c. Após a criação da curva d, aplica-se uma translação no eixo X.

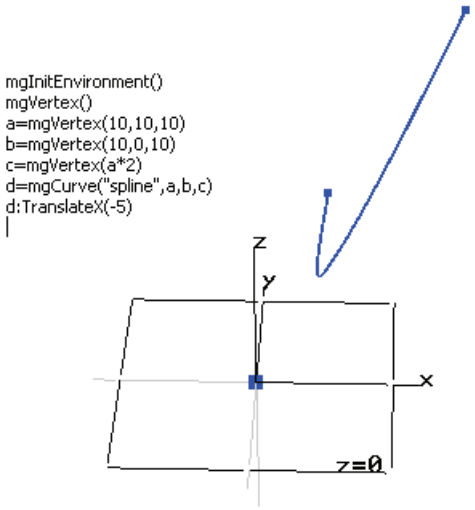


Figura 3.6: Sobrecarga de parâmetros das classes *mgVertex* e *mgCurve*.

A figura 3.7 mostra a criação de vértices e curvas com os comandos *mgVertex* e *mgCurve*. Após a criação das curvas, o vértice a, que é adjacente a duas curvas é transladado no eixo Z.

PUC-Rio - Certificação Digital Nº 0421097/CA

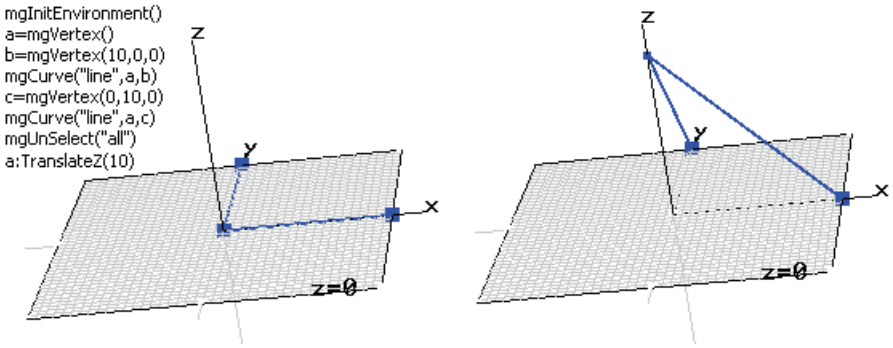


Figura 3.7: Translação de vértices.

Na figura 3.8 mostra-se a criação de uma curva spline cujo formato é uma helicoidal. Esta curva é criada utilizando-se funções matemáticas de senos, cosenos e da constante Pi. Além disso, o usuário pode definir as dimensões que a helicoidal terá em função das variáveis R, a, e H, que são: Raio, ângulo e altura, respectivamente. Após a criação da curva, faz-se uma translação da metade da altura de forma que a curva fique centralizada na origem.

É conveniente ressaltar que este tipo de curva seria difícil de ser criada sem o uso das funções matemáticas que a linguagem Lua oferece.

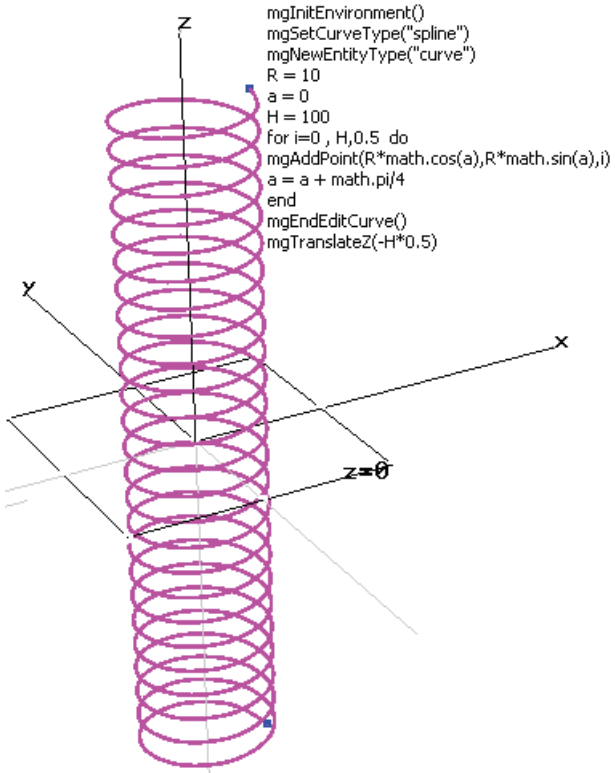
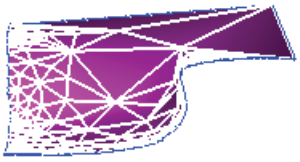


Figura 3.8: Spline com formato helicoidal.

PUC-Rio - Certificação Digital Nº 0421097/CA

Na figura 3.9 mostra-se a combinação dos comandos de histórico, gerados pela interface, com os comandos definidos pelas classes de cada entidade do MG (vértice, curva, superfície, e volume). Pode-se perceber que uma superfície pode ser criada utilizando um único comando, com a chamada *mgSurface*, passando como parâmetros objetos Lua do tipo curva.



```
a=mgNewEntityType("curve")
mgAddPoint(-5.0000000,0.0000000,0.0000000)
mgAddPoint(3.0000000,0.0000000,0.0000000)
mgEndEditCurve()
b=mgNewEntityType("curve")
mgAddPoint(3.0000000,0.0000000,0.0000000)
mgAddPoint(3.0000000,-10.0000000,0.0000000)
mgEndEditCurve()
mgSetCurveType("spline")
```

```
c=mgNewEntityType("curve")
mgAddPoint(3.0000000,-10.0000000,0.0000000)
mgAddPoint(0.0000000,-10.0000000,0.0000000)
mgAddPoint(0.0000000,-10.0000000,-1.0000000)
mgAddPoint(-1.0000000,-10.0000000,-2.0000000)
mgAddPoint(-5.0000000,-10.0000000,-2.0000000)
mgEndEditCurve()
--u3
d=mgNewEntityType("curve")
mgAddPoint(-5.0000000,0.0000000,0.0000000)
mgAddPoint(-5.0000000,-3.0000000,0.0000000)
mgAddPoint(-5.0000000,-5.0000000,-1.0000000)
mgAddPoint(-5.0000000,-6.0000000,-2.0000000)
mgAddPoint(-5.0000000,-10.0000000,-2.0000000)
mgEndEditCurve()
mgUnSelect("all")
mgSurface("bilinear","quadrilat",a,b,c,d)
```

Figura 3.9: Geração de superfície com a classe *mgSurface*.

Na próxima seção mostra-se o uso destes comandos de modelagem incorporados no MG, aplicados para um caso real de modelos de engenharia naval. Fica, então, exemplificado como a modelagem por *scripts* oferece uma grande facilidade para modelar e variar a forma do modelo final em tempo de execução.

3.7.1 Vantagens da Modelagem em Lua

Podem-se mencionar várias vantagens para o uso da linguagem Lua como mecanismo para estender a abrangência de um aplicativo qualquer:

- Lua fornece um ambiente embutido de modelagem no MG.
- Lua é uma linguagem *case-sensitive* e oferece a possibilidade de programar com objetos. Isto é possível devido aos mecanismos de meta-tabelas que Lua oferece.
- Todas as definições feitas pelo usuário em Lua (variáveis e funções) são lembradas pelo MG. Estas definições permanecem ativas até que o programa MG seja fechado.
- Possibilidade de acompanhar a geração do modelo graficamente enquanto o *script* estiver sendo executado.
- A construção e variação de parâmetros de forma de uma estrutura flutuante é feita de forma automática em Lua, de forma geral ou modularizada.

PUC-Rio - Certificação Digital Nº 0421097/CA

– Lua permite parametrizar geometrias em função de seus parâmetros chave.

PUC-Rio - Certificação Digital Nº 0421097/CA