Grupo PTERO-VIS

ASA-Ensaio3D:
Especificação da Linguagem de Script

Autores: João Luiz Luguesi (editor)
José Martim Nicoladelli

Colaboradores: -

RESUMO

O texto especifica a linguagem de script disponibilizada no ASA-Ensaio3D. O ambiente do interpretador de script objetiva iniciar o estudante no uso do OpenGL, para em curto prazo de tempo exercitar diversos conceitos relacionados com a computação gráfica, tais como, transformações geométricas, modelagem de objetos e cenários, tratamento de cores e iluminação.

Em relação à linguagem, descreve os aspectos léxicos, sintáticos e semânticos dos elementos da linguagem. Em relação aos aspectos semânticos, descreve os conceitos de unidades de programas, comandos, expressões, funções e operadores. Descreve o sistema de tipos de dados, variáveis e células de memórias. Ao final mostra alguns exemplos, utilizando alguns recursos disponíveis na linguagem.

Palavras-chave: OpenGL, Computação gráfica, ASA.

SUMÁRIO

LISTA DE TABELAS E ILUSTRAÇÕES

REFERÊNCIAS

1 OBJETIVO

O *software* educativo ASA-Ensaio3D se presta ao ensino de computação gráfica, utilizando como base as bibliotecas do OpenGL. A princípio implementa o formato de parte dos comandos e opções do OpenGL, integrados numa linguagem de programação e um ambiente visual. O objetivo final é encurtar o caminho entre os conceitos e sua aplicação prática, sem entrar nas idiossincrasias do ambiente C++, da relação Windows x OpenGL e das bibliotecas envolvidas.

A especificação da linguagem de script para o *software* educativo ASA-Ensaio3D é fortemente inspirado no paradigma de GCL (*Guarded Command Language*) de Dijkstra. O texto está dividido em quatro partes. A primeira parte descreve os aspectos léxicos, do reconhecimento dos símbolos, das palavra reservadas, das palavras-chave, das constantes numéricas e lógicas, e dos nomes de variáveis e sub-rotinas. A segunda parte descreve os aspectos sintáticos, do reconhecimento de unidades, de comandos e de expressões. A terceira parte descreve os aspectos semânticos, do reconhecimento das restrições sobre as frases que podem ser construídas sintaticamente corretas, mas sem significado prático. A quarta e última parte mostra alguns exemplos de algoritmos aplicados à computação gráfica com o uso o OpenGL.

2 ANALISADOR LÉXICO

A gramática léxica especifica o modo como o analisador léxico interpreta o texto fonte e o decompõe em lexemas. Os lexemas são fichas (*tokens*) com informações de tipo, valor, características físicas (linha, coluna e tamanho), e possível código de erro, de cada elemento léxico reconhecido. As fichas podem ser de oito tipos:

a) ficha de constante inteira;
b) ficha de constante real;
c) ficha de constante hexadecimal;
d) ficha de palavra reservada;
e) ficha de símbolo;
f) ficha de nome;
g) ficha de final de fonte (*eof*); e
h) ficha desconhecida com erro.

Os tipos de fichas a) a g) são fichas válidas para o analisador sintático e estão destacadas com coloração de fundo ao longo da gramática léxica. As fichas desconhecidas ocorrem sempre que a gramática léxica não reconhecer determinada sequência de caracteres. O ocorrência de ficha desconhecida causa falha no analisador sintático, que reporta o erro de acordo com o código de erro associado à ficha.

2.1 DECOMPOSIÇÃO DO TEXTO FONTE

Do ponto de vista do analisador léxico, o texto fonte é uma sequência de caracteres gráficos, com alguns elementos especiais intercalados. Os elementos especiais podem ser: a) *eof*, que assinala o final do texto fonte; b) *nl*, que determina o final das linhas, mesmo as linhas vazias; c) □, que resume sequências não vazias formadas pelos caracteres *espaço* e *tab*; e d) comentário..., que compreende uma sequência de caracteres que se estende até o final da linha, excluindo o caractere *nl*.

2.2 NOTAÇÃO EBNF UTILIZADA

Para a especificação da gramática utiliza-se a notação EBNF, em que a primeira linha – cabeçalho - declara o nome de um sintagma (elemento não-terminal) seguido por : (dois pontos) e opcionalmente por **um-de**. As linhas seguintes definem o sintagma utilizando-se de sintagmas e lexemas (elementos terminais). Alguns lexemas podem ser construídos por sequências de caracteres, e outros podem ser os próprios caracteres. A interpretação das linhas depende do cabeçalho: a) com a presença de **um-de**, cada elemento das linhas seguintes é uma definição do sintagma; b) sem a presença de **um-de**, cada linha seguinte é uma definição completa do sintagma, em que a ordem é relevante. As definições podem conter elementos opcionais, assinalados pelo termo *opt* aposto, p. ex., o sinal na especificação do expoente é opcional. As especificações podem ser recorrentes, p. ex., a especificação das linhas.

O texto da especificação da gramática léxica utiliza cores para marcar categorias de elementos: a) a cor magenta marca os lexemas formados por mais de um caractere; b) a cor verde marca os lexemas formados pelo próprio caractere do texto fonte; c) a cor azul marca todos os outros elementos. O termo *opt* vem subscrito e em cor laranja.

2.3 ESPECIFICAÇÃO DA GRAMÁTICA LÉXICA

A especificação 2.1 mostra a gramática léxica em EBNF. Pressupõe-se que todos os elementos estejam separados pelo símbolo ■, isto é, espaço vazio. Os casos em que isto não se aplica estão assinalados explicitamente.

Especificação 2.1 - Gramática léxica (EBNF)

```
fonte:
    linhas_opt eof

linhas:
    □_opt linha
    linhas nl □_opt linha

linha:
    elementos □_opt
    elementos □ – – comentário...
    – – comentário...

elementos: um-de
    verbetes símbolos

verbetes:
    verbete
    verbetes □ verbete
    símbolos □_opt verbete

símbolos:
    símbolo
    símbolos □_opt símbolo
    verbetes □_opt símbolo
```

verbete: um-de
palavra-reservada palavra-chave nome constante-inteira constante-real constante-hexa

palavra-reservada: um-de
nome-de-comando-opengl nome-de-opção-opengl nome-de-pseudo-comando-opengl
nome-especial-reservado nome-de-função nome-de-operador nome-de-comando-de-script
defalta delta fim final inicial PRAGMA

nome-de-comando-opengl: um-de
glBegin glColor3f glColor3fv glColor4f glColor4fv glCullFace glDisable glEnable glEnd
glFrontFace glLineStipple glLineWidth glNormal3f glNormal3fv glPointSize glPopAttrib
glPopMatrix glPushAttrib glPushMatrix glRotatef glScalef glShadeModel glTranslatef
glVertex2f glVertex2fv glVertex3f glVertex3fv glVertex4f glVertex4fv

nome-de-opção-opengl: um-de
GL_BACK GL_CCW GL_CULL_FACE GL_CURRENT_BIT GL_CW GL_FLAT GL_FRONT
GL_FRONT_AND_BACK GL_LINE_BIT GL_LINE_LOOP GL_LINE_STRIPPLE GL_LINE_STRIP
GL_LINES GL_NORMALIZE GL_POINT_BIT GL_POINTS GL_POLYGON GL_POLYGON_BIT
GL_QUAD_STRIP GL_QUADS GL_SMOOTH GL_TRIANGLE_STRIP GL_TRIANGLE_FAN
GL_TRIANGLES

nome-de-pseudo-comando-opengl: um-de
glRotatefv glScalefv glTranslatefv

nome-especial-reservado: um-de
B1 B2 B3 B4 F GRAUS K1 K2 K3 K4 PI Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 QMK SEQ TEMPO V

nome-de-função: um-de
abs acos ângulo asen atan cos cosel cosh def expn hsv int ln max min módulo
normal random real rgb sen senel senh sinal sqrt tan tanh tipo trocaw unitário

nome-de-operador: um-de
div e equ mod ou xou

nome-de-comando-de-script: um-de
aborta executa mostra randomiza repete seleciona termina

palavra-chave: um-de
x y z w r g b a h s v 1 2 3 4 T DELTA MATERIAL MOSTRA ROTAÇÃO

nome: um-de
nome-de-variável nome-de-subrotina

nome-de-variável:
@$_{opt}$ nome-genérico

nome-de-subrotina:
nome-genérico

nome-genérico:
caractere-alfa qualquer-caractere$_{opt}$

qualquer-caractere:
qualquer-caractere$_{opt}$ _$_{opt}$ caractere-alfa
qualquer-caractere$_{opt}$ _$_{opt}$ dígito

caractere-alfa: um-de
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
á à â ã é ê í ó ô õ ú ü ç Á À Â Ã É Ê Í Ó Ô Õ Ú Ü Ç

constante-real:
constante-inteira . fração expoente$_{opt}$
constante-inteira expoente

fração:
dígito fração$_{opt}$

expoente:
E sinal$_{opt}$ constante-inteira

sinal: um-de
+ –

constante-inteira:
constante-inteira$_{opt}$ dígito

constante-hexa:
0 X dígitos-hexa

dígitos-hexa:
dígitos-hexa$_{opt}$ dígito-hexa

dígito-hexa: um-de
dígito A B C D E F a b c d e f

dígito: um-de
0 1 2 3 4 5 6 7 8 9

símbolo: um-de
operador ( ) ; , ? | := ==>

operador: um-de
~ = < > + – * / [ ] { } ' " ! ^ # <> <= >= ><

2.4 CONSIDERAÇÕES SOBRE O ANALISADOR LÉXICO

Algumas situações são válidas para o analisador léxico, o que não significa que o analisador sintático esteja de acordo. Seguem algumas observações:

a) o texto pode estar vazio;
b) as linhas podem estar rodeadas por espaços em branco;
c) os comentários devem estar separados dos elementos antecedentes por um espaço em branco;
d) os comentários de linha inteira podem iniciar no primeiro caractere da linha;
e) as palavras reservadas, palavras chave, nomes genéricos, constantes inteiras, constantes reais e constantes hexadecimais devem estar separadas entre si por um espaço em em branco;
f) os nomes genéricos podem conter, mas não iniciar e terminar com o caractere _ (sublinhado);

g) o analisador léxico é sensível à caixa alta e caixa baixa;
h) o analisador léxico retorna o símbolo *eof* ao final do texto, todas as vezes que for requisitado;
i) palavras reservadas não podem ser utilizadas como nomes de variáveis e ou sub-rotinas;
j) palavras chave possuem significado específico em determinados contextos, fora destes podem ser utilizadas como nomes ou números;
k) o analisador léxico não considera os limites do processador para reais, inteiros e hexadecimais;
l) os valores inteiros que excedem a capacidade do processador são majorados para valores reais; e
m) os valores hexadecimais que excedem a capacidade do processador causam erro de execução.

Algumas destas observações afetam a analisador sintático ou a máquina virtual, e constam somente para fins explicativos.

É interessante observar que as fichas geradas pelo analisador léxico, segundo a gramática léxica, são repassadas ao analisador sintático na mesma sequência em que são reconhecidas. As fichas do tipo palavra-reservada e símbolo em geral não são identificadas pelo tipo e sim pelo valor. O analisador léxico aceita a devolução de uma ou mais fichas (cuidados com a ordem!) para facilitar a implementação da compilação de linguagens LL(*k*), quando se torna necessário *olhar* algumas fichas em avanço.

3 ANALISADOR SINTÁTICO

A gramática sintática especifica os elementos e construtos da linguagem de script baseado na classe de linguagens LL(*k*) que permite a construir a árvore de sintaxe baseada em objetos, em que cada objeto compila a parte pertinente. Por exemplo, o comando de atribuição requer uma expressão, portanto, cria um objeto da classe de expressões e aciona seu método de compilação, que *consome* a parte das fichas as quais reconhece como expressão.

O analisador sintático busca as fichas reconhecidas pelo analisador léxico, monta a árvore de sintaxe que implementa a máquina virtual que executa os comandos da linguagem, e monta as estruturas das expressões executadas por calculadoras RPN.

3.1 NOTAÇÃO EBNF UTILIZADA

A notação utilizada é semelhante a EBNF descrita em 2.2. O texto da especificação da gramática sintática utiliza cores para marcar categorias de elementos: a) a cor magenta marca os lexemas identificados diretamente pelo valor informado nas fichas; b) a cor verde marca os lexemas identificados pelo tipo informado nas fichas; e c) a cor azul marca todos os outros elementos.

3.2 ESPECIFICAÇÃO DA GRAMÁTICA SINTÁTICA

A especificação 3.1 mostra a gramática sintática em EBNF. Não há considerações a respeito da justaposição de elementos, ou seja, o reconhecimento das fichas é realizado completamente pelo analisador léxico.

Especificação 3.1 - Gramática sintática (EBNF)

programa:
pragmas unidade-principal subunidades$_{opt}$ eof

pragmas:
pragmas$_{opt}$ pragma

pragma:
PRAGMA assertivas

assertivas:
assertiva
assertivas , assertiva

assertiva:
nome-especial-de-variável-lógica = constante-lógica
nome-especial-de-variável-escalar = constante-escalar
nome-especial-de-variável-vetor4 = constante-vetor4
DELTA = constante-lógica
MATERIAL = opção-de-material
MOSTRA = constante-lógica
ROTAÇÃO = (constante-lógica , constante-lógica , constante-inteira , constante-lógica)
TEMPO = constante-inteira
glCullFace (opção-cull-face-pragma)
glDisable (opção-enable-pragma)
glEnable (opção-enable-pragma)
glFrontFace (opção-front-face)
glShadeModel (opção-shade-model-pragma)

opção-enable-pragma: um-de
GL_CULL_FACE GL_NORMALIZE

opção-cull-face-pragma: um-de
GL_BACK GL_FRONT GL_FRONT_AND_BACK

opção-shade-model-pragma: um-de
GL_FLAT GL_SMOOTH

constante-vetor4:
[ ]
[constante-escalar]
[constante-escalar , constante-escalar]
[constante-escalar , constante-escalar , constante-escalar]
[constante-escalar , constante-escalar , constante-escalar , constante-escalar]

constante-escalar:
sinal$_{opt}$ constante-inteira
sinal$_{opt}$ contante-real

opção-de-material: um-de
1 2 3 4 5 6 7 8

sinal: um-de
+ –

unidade-principal:
comandos$_{opt}$

subunidades:
subunidades$_{opt}$ subunidade

subunidade:
==> nome-de-subrotina parâmetros comandos$_{opt}$

parâmetros:
(nomes-de-variáveis$_{opt}$) ;$_{opt}$

nomes-de-variáveis:
nome-de-variável
nomes-de-variáveis , nome-de-variável

comandos:
comandos$_{opt}$ comando-guardado

comando-guardado:
comando pós-guarda$_{opt}$;$_{opt}$

pós-guarda:
? expressão-condicional

comando: um-de
comando-opengl pseudo-comando-opengl comando-de-script

comando-opengl:
glBegin (opção-begin)
glColor3f (expressão-escalar , expressão-escalar , expressão-escalar)
glColor3fv (expressão-vetor4)
glColor4f (expressão-escalar , expressão-escalar , expressão-escalar , expressão-escalar)
glColor4fv (expressão-vetor4)
glDisable (opção-enable)
glEnable (opção-enable)
glEnd ( )
glFrontFace (opção-front-face)
glLineStipple (expressão-inteira , expressão-inteira)
glLineWidth (expressão-escalar)
glNormal3f (expressão-escalar , expressão-escalar , expressão-escalar)
glNormal3fv (expressão-vetor4)
glPointSize (expressão-escalar)
glPopAttrib ( )
glPopMatrix ( )
glPushAttrib (opções-attrib)
glPushMatrix ( )
glRotatef (expressão-escalar , expressão-escalar , expressão-escalar , expressão-escalar)
glScalef (expressão-escalar , expressão-escalar , expressão-escalar)
glTranslatef (expressão-escalar , expressão-escalar , expressão-escalar)
glVertex2f (expressão-escalar , expressão-escalar)
glVertex2fv (expressão-vetor4)
glVertex3f (expressão-escalar , expressão-escalar , expressão-escalar)
glVertex3fv (expressão-vetor4)

glVertex4f (expressão-escalar , expressão-escalar , expressão-escalar , expressão-escalar)
glVertex4fv (expressão-vetor4)

opção-begin: um-de
GL_LINE_LOOP GL_LINE_STRIP GL_LINES GL_POINTS GL_POLYGON
GL_QUAD_STRIP GL_QUADS GL_TRIANGLE_STRIP GL_TRIANGLE_FAN GL_TRIANGLES

opção-enable: um-de
GL_LINE_STIPPLE GL_NORMALIZE GL_SMOOTH

opção-front-face: um-de
GL_CCW GL_CW

opções-attrib:
opção-attrib
opções-attrib | opção-attrib

opção-attrib: um-de
GL_CURRENT_BIT GL_LINE_BIT GL_POLYGON_BIT

pseudo-comando-opengl:
glRotatefv (expressão-escalar , expressão-vetor)
glScalefv (expressão-vetor4)
glTranslatefv (expressão-vetor4)

comando-de-script:
atribuição
nome-de-subrotina (expressões$_{opt}$)
aborta (expressão-inteira)
executa comandos$_{opt}$ fim
mostra (expressões$_{opt}$)
randomiza (expressão-inteira$_{opt}$)
repete retardo$_{opt}$ guardas$_{opt}$ fim
seleciona guardas$_{opt}$ fim
termina (expressão-inteira$_{opt}$)

atribuição:
nome-de-variável := expressão
nome-de-variável " índice-de-linha := expressão-vetor4
nome-de-variável ' índice-de-parte := expressão-escalar
nome-de-variável " índice-de-linha ' índice-de-parte := expressão-escalar

expressões:
expressão
expressões , expressão

nome-de-subrotina:
nome

retardo:
delta expressão-inteira

guardas:
guardas$_{opt}$ guarda

guarda:
defalta ? comandos$_{opt}$ fim ;$_{opt}$
final ? comandos$_{opt}$ fim ;$_{opt}$
inicial ? comandos$_{opt}$ fim ;$_{opt}$
expressão-condicional ? comandos$_{opt}$ fim ;$_{opt}$

expressão: um-de
expressão-condicional expressão-numérica expressão-indefinida

expressão-condicional:
expressão-lógica

expressão-numérica: um-de
expressão-escalar expressão-vetor4 expressão-matriz4

expressão-indefinida: um-de
expressão-lógica expressão-aditiva

expressão-escalar:
expressão-aditiva

expressão-vetor4:
expressão-aditiva

expressão-matriz4:
expressão-aditiva

expressão-inteira:
expressão-aditiva

expressão-lógica:
expressão-relacional
expressão-lógica operador-lógico expressão-relacional

operador-lógico: um-de
e equ ou xou

expressão-relacional:
expressão-aditiva
expressão-aditiva operador-de-igualdade expressão-aditiva
expressão-aditiva operador-relacional expressão-aditiva

operador-de-igualdade: um-de
= <>

operador-relacional: um-de
< <= > >=

expressão-aditiva:
expressão-multiplicativa
expressão-aditiva operador-aditivo expressão-multiplicativa

operador-aditivo: um-de
+ –

expressão-multiplicativa:
expressão-potenciação

expressão-multiplicativa operador-multiplicativo expressão-potência
expressão-multiplicativa operador-de-produto expressão-potência

operador-multiplicativo: um-de
* / div mod

operador-de-produto: um-de
><

expressão-potenciação:
expressão-unária
expressão-unária ^ expressão-potenciação

expressão-unária:
expressão-pós-fixada
– expressão-unária
+ expressão-unária
~ expressão-unária

expressão-pós-fixada:
expressão-primária
expressão-pós-fixada !
expressão-pós-fixada " T
expressão-pós-fixada " índice-de-linha
expressão-pós-fixada ' índice-de-parte

índice-de-linha: um-de
1 2 3 4

índice-de-parte: um-de
x y z w r g b a h s v 1 2 3 4

expressão-primária:
constante-inteira
constante-hexa
constante-real
construtor-de-vetor4
construtor-de-matriz4
nome-especial-de-variável
nome-especial-de-constante
nome-de-variável
função
(expressão-lógica)
(expressão-aditiva)

construtor-de-vetor4:
[ ]
[expressão-escalar]
[expressão-escalar , expressão-escalar]
[expressão-escalar , expressão-escalar , expressão-escalar]
[expressão-escalar , expressão-escalar , expressão-escalar , expressão-escalar]

construtor-de-matriz4x4:
{ }

{ expressão-vetor4 }
{ expressão-vetor4 , expressão-vetor4 }
{ expressão-vetor4 , expressão-vetor4 , expressão-vetor4 }
{ expressão-vetor4 , expressão-vetor4 , expressão-vetor4 , expressão-vetor4 }

nome-especial-de-variável: um-de
nome-especial-de-variável-lógica nome-especial-de-variável-escalar
nome-especial-de-variável-vetor4 GRAUS SEQ TEMPO

nome-especial-de-variável-lógica: um-de
B1 B2 B3 B4

nome-especial-de-variável-escalar: um-de
K1 K2 K3 K4

nome-especial-de-variável-vetor4: um-de
Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 QMK

nome-especial-de-constante: um-de
constante-lógica PI

constante-lógica: um-de
F V

nome-de-variável:
nome

função:
abs (expressão-escalar)
acos (expressão-escalar)
ângulo (expressão-escalar)
asen (expressão-escalar)
atan (expressão-escalar)
cos (expressão-escalar)
cosel (expressão-escalar , expressão-escalar)
cosel (expressão-escalar , expressão-escalar , expressão-escalar)
cosh (expressão-escalar)
def (expressão)
expn (expressão-escalar)
hsv (expressão-vetor4)
hsv (expressão-escalar , expressão-escalar , expressão-escalar , expressão-escalar)
int (expressão-escalar)
ln (expressão-escalar)
max (expressões-escalares)
min (expressões-escalares)
módulo (expressão-vetor4)
normal (expressão-vetor4 , expressão-vetor4 , expressão-vetor4)
random ( )
random (expressão-inteira , expressão-inteira)
real (expressão-escalar)
rgb (expressão-vetor4)
rgb (expressão-escalar , expressão-escalar , expressão-escalar , expressão-escalar)
sen (expressão-escalar)

```
    senel ( expressão-escalar , expressão-escalar )
    senel ( expressão-escalar , expressão-escalar , expressão-escalar )
    senh ( expressão-escalar )
    sinal ( expressão-escalar )
    sqrt ( expressão-escalar )
    tan ( expressão-escalar )
    tanh ( expressão-escalar )
    tipo ( expressão )
    trocaw ( expressão-vetor4 , expressão-escalar )
    unitário ( expressão-vetor4 )

expressões-escalares:
    expressão-escalar
    expressões-escalares , expressão-escalar
```

4 DESCRIÇÃO DA SEMÂNTICA

A descrição da semântica cobre diversos aspectos do ciclo de vida de um programa, da construção do programa, da interpretação dos elementos sintáticos, das funcionalidades de comandos, operadores e funções, e dos tipos de dados com os mecanismos de armazenamento dos dados.

4.1 INTERPRETADOR DE SCRIPT

O interpretador de script é composto por um editor de programas, um compilador, e uma máquina virtual associada a uma calculadora de expressões, que executa os programas. O objetivo principal dos programas é desenhar cenários sobre o portal do ASA-Ensaio3D, utilizando comandos OpenGL.

Fundamentalmente o interpretador possui dois momentos principais: a) a edição, que permite editar e manter a biblioteca de programas, imprimir e enviar por e-mail programas; e b) a execução de programas. Cada ciclo de execuções caracteriza uma **sessão**, que inicia com a compilação e criação da máquina virtual do programa, seguida por sequência de execuções ativadas a intervalos regulares.

4.1.1 Ciclo de vida do interpretador de script

A evolução do interpretador de script segue o ciclo de vida descrito pelo diagrama de estados da figura 4.1. Inicialmente o interpretador é colocado no estado `edição`. A partir deste estado, eventos internos e eventos gerados pelo usuário provocam as transições previstas pelo diagrama de estados.

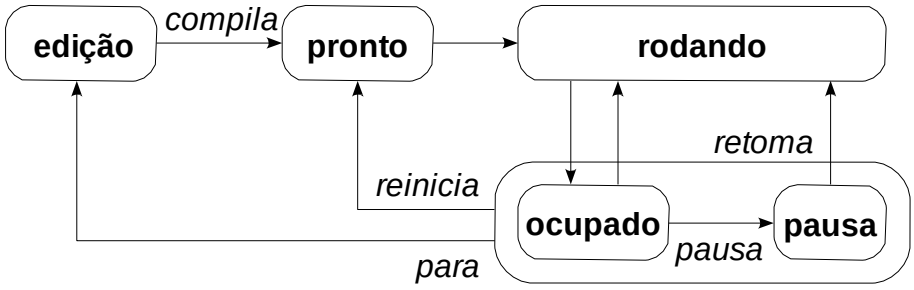


Figura 4.1 - Ciclo de vida do interpretador de script

As seções a seguir descrevem os estados um a um, e o efeitos eventos e as transições possíveis em cada estado.

O estado `ocupado` é importante para detectar o *estouro* dos intervalos de execução do programa, e indicar esta situação no painel do editor de programas.

4.1.1.1 Estado edição

Quando o interpretador está no estado `edição`, todas as funcionalidades do editor de programas estão disponíveis. Ao acionar a opção *compila*, o interpretador compila o programa selecionado e ativa a máquina virtual, entrando no estado `pronto`. Em caso de erro de compilação, o interpretador permanece no estado `edição`.

Existem dois modos de ativação da máquina virtual: a) modo de execução contínua, e b) modo de execução passo à passo. Os modos afetam diretamente as transições no estado `ocupado`.

4.1.1.2 Estado pronto

O estado `pronto` é um estado transitório, durante o qual contadores, estados internos e células de memória são fixados em uma situação inicial. Em seguida o interpretador entra no estado `rodando` automaticamente.

4.1.1.3 Estado rodando

O estado `rodando` é um estado transitório, durante o qual o cenário do OpenGL é iniciado, as opções configuradas através da interface do cenário são instaladas, e o contador de execuções SEQ é incrementado. Em seguida o interpretador entra no estado `ocupado` automaticamente.

4.1.1.4 Estado ocupado

O programa de script executa efetivamente durante o estado `ocupado`. O programa pode terminar a) normalmente, b) por erro de execução, ou c) pelo comando de aborto. Nos casos b) e c) a sessão para definitivamente, e o interpretador entra no estado `edição`.

Após o término normal do programa, cinco eventos/situações podem ser identificadas: a) a solicitação de *pausa*, que leva o interpretador ao estado `pausa`; b) a solicitação de *para*, que leva o interpretador ao estado `edição`; c) a solicitação de *reinicia* (no modo contínuo ou passo à pas-

so), que leva o interpretador ao estado `pronto`; d) a situação do programa estar operando no modo passo a passo, que leva o interpretador ao estado `pausa`; e e) a situação do programa estar operando no modo contínuo, que leva o interpretador ao estado `rodando`.

Os comandos `mostra` e `repete` abrem uma janela para tratamento da interface, permitindo que um ciclo de execução seja interrompido pelo usuário.

4.1.1.5 Estado pausa

Durante o estado `pausa` o ciclo de execuções do programa fica suspenso. Três eventos podem ser identificados: a) a solicitação de *retoma*, que leva o interpretador ao estado `rodando` no mesmo modo em que se encontra; b) a solicitação *para* que leva o interpretador ao estado `edição;` e c) a solicitação *reinicia* (no modo contínuo ou passo à passo), que leva o interpretador ao estado `pronto`.

4.1.2 Máquina virtual

A máquina virtual, ou executor, executa os comandos do programa, que por sua vez, resolvem as expressões associadas, utilizando-se da calculadora RPN. Durante determinada sessão, a máquina virtual em modo contínuo é ativada a intervalos regulares de 60ms. O sistema é de *tempo real fraco*, e eventualmente o tempo de execução do programa excede os 60ms, *pulando* um ou mais tiques de relógio. Os erros fatais encontrados durante a execução dos comandos causam o aborto do programa, terminando a sessão.

4.1.3 Calculadora RPN

A calculadora RPN (*Reverse Polish Notation*) executa os cálculos dos valores das expressões, utilizando uma máquina de pilha. Os operadores verificam os tipos dos operandos dinamicamente, e realizam operações especificas (sobrecarga, polimorfismo), dependendo das combinações dos tipos. Os erros fatais encontrados durante a execução dos cálculos causam o aborto do programa, terminando a sessão.

4.2 O PROGRAMA

Os programas possuem quatro aspectos relevantes: a) a divisão dos algoritmos em unidades, módulos ou sub-rotinas; b) os tipos de valores e a tipagem de variáveis e parâmetros; c) os comandos de ação, de controle do fluxo, e do interação com o OpenGL; e d) as expressões, os operadores e as funções.

4.2.1 Unidades do programa

O programa pode ser dividido em diversas unidades. A primeira unidade é a unidade principal, ativada naturalmente pela máquina virtual. As demais unidades são sub-rotinas não recorrentes (inclusive recorrência transitiva), que podem ser chamadas por qualquer uma das unidades.

A opção de não oferecer recorrência de chamadas de sub-rotinas decorre do fato de que cenários típicos em computação gráfica possuem estrutura hierárquica. Tipicamente, objetos são

compostos e articulados por outros objetos em forma de árvore, não obstante, suas formas possam ser semelhantes. Por exemplo, se uma esfera estiver associada a outras três esferas, deve existir um objeto maior que coordena esta associação e articulação, descaracterizando a visão inicial da recorrência de esferas.

4.2.2 Comunicação entre unidades

A comunicação de dados entres as unidades pode ser realizada através de variáveis globais, ou pela passagem de parâmetros. O cabeçalho da sub-rotina declara a lista de parâmetros, e a chamada da sub-rotina repassa valores em células de memória. A passagem de parâmetros opera por valor conjugado com passagem por referência. Esta forma de comunicação é bidirecional, isto é, se o chamador passa como parâmetro atual uma variável (como expressão primária!), a sub-rotina pode declarar o parâmetro formal como global, e o valor atribuído a este parâmetro pode ser *visto* pelo chamador na variável.

As variáveis globais e locais do programa possuem comportamento semelhante ao de variáveis estáticas, e propagam seus valores de uma execução do programa para outra. Além disto, as variáveis locais das sub-rotinas propagam seus valores de uma chamada para outra.

4.2.3 Tipos de valores

A máquina virtual opera com cinco tipos de valores básicos que podem ser gerados pela execução de expressões e armazenados em células de memória. A natureza destes valores é adequada ao tipo de processamento necessário para objetos de computação gráfica. Os valores podem ser:

a) valores inteiros de 32 bites com sinal;
b) valores reais de dupla precisão – *double*;
c) valores vetor4, arranjo de 4 valores reais;
d) valores matriz4, arranjo de 4x4 valores reais; e
e) valores lógicos.

Os valores são vinculados dinamicamente a células de memória, de modo que, em determinado momento estas poderão conter um valor de qualquer um dos cinco tipos, ou ainda o valor especial *indefinido*, representado pelo caractere ? (interrogação). O quadro 4.2 mostra os grupos de tipos de valores e suas designações.

		tipos de valores básicos					**valor**
		lógico	**inteiro**	**real**	**vetor4**	**matriz4**	**indefinido**
grupos	**condicional**	X					
	escalar		X	X			
	numérico	X	X	X	X	X	
	qualquer	X	X	X	X	X	X

Quadro 4.2 - Relação de tipos de valores

Nas mensagens do compilador e da máquina virtual, os tipos vetor4 e matriz4 também são citados como `vetor-4 […]` e `matriz-4x4 {…}` respectivamente. Os termos vetor4 e matriz4 foram talhados para evitar confusão em torno dos termos *vetor* e *matriz* como entidades matemáticas.

4.2.3.1 Tipo de valores lógicos

Valores lógicos representam os estados `verdadeiro` e `falso`, e são adequados para formular condições. As condições são cláusulas associadas ao mecanismo de guardas para controlar os comandos de seleção e repetição.

4.2.3.2 Tipo de valores inteiros

Valores inteiros podem ser de dois tipos, os inteiros de 32 bites com sinal, e os inteiros maior que são convertidos (ou majorados) automaticamente em valores reais. Os valores inteiros declarados como hexadecimais não podem ser majorados. A função `real()` implementa a majoração explícita.

4.2.3.3 Tipo de valores reais

Valores reais são tratados como valores em ponto flutuante de dupla precisão. Qualquer valor além da capacidade do processador, `infinito` e `NaN`, causam erro de execução.

4.2.3.4 Tipo de valores vetor4

Valores vetor4 são formados por quatro valores reais, e são adequados para representar: a) pontos, vértices e vetores no sistema de coordenadas cartesianas homogêneas 3D; b) cores no sistema de cores RGB e HSV; e c) valores adimensionais como por exemplo escalas. A linguagem oferece dois mecanismos básicos para manipular vetores4: a) o construtor `[x,y,z,w]` que permite criar vetores4, p. ex., `[12,3.5,0,1]` cria um vetor4 com `x=12`, `y=3.5`, `z=0` e `w=1`; e b) o extrator que permite extrair ou modificar uma das partes ou valores associadas aos vetores4, p. ex., dado o vetor4 `v`, as expressões `v'x`, `v'r`, `v'h` e `v'1` extraem (ou modificam) a primeira parte ou valor do vetor4.

No sistema de coordenadas homogêneas 3D, os quatro valores representam as quantidades `(x,y,z,w)` nos eixos `X`, `Y` e `Z`, e da magnitude `W`. Os valores efetivos de `(x,y,z)` são obtidos pela divisão destes pelo valor `w`, que deve ser diferente de zero. Pontos em que `w=0`, são considerados fora do espaço dimensional. Um vetor4 pode representar um ponto ou vértice `(x,y,z)` ou um vetor `(x,y,z)` no espaço de coordenadas 3D. As operações aritméticas sobre vetores4 sempre levam em consideração o valor `w` das coordenadas homogêneas.

No sistema de cores RGB os quatro valores representam as quantidades `(r,g,b,a)` referentes às cores vermelho, verde e azul, e fator alfa. O fator alfa determina a transparência de objetos, em que 0.0 significa totalmente transparente e 1.0 totalmente opaco. Cada um dos valores pode variar de 0.0 a 1.0, e quando extrapolarem estes limites serão recortados adequadamente. No sistema de HSV os quatro valores representam as quantidades `(h, s, v, a)` referentes à tonalidade ou matiz, saturação, valor ou luminosidade, e fator alfa.

Além destas duas interpretações, vetores4 podem armazenar quaisquer outras quantidades, sendo inadequado operar aritmeticamente sobre estes dados.

A interpretação dos valores tipo vetor4 depende do contexto em que são utilizados, p. ex., no comando `glVertex3fv(v)`, `v` é interpretado como vértice ou ponto `(x/w,y/w,z/w)`, no comando `glColor4fv(cor)`, `cor` é interpretado como `(r,g,b,a)`, e no comando `glScalefv(s)`, `s` fornece as escalas `Sx`, `Sy` e `Sz`. As descrições de comandos, operadores e funções detalham a interpretação dos vetores4.

4.2.3.5 Tipo de valores matriz4

Valores matriz4 são formados por quatro linhas de vetores4, e são adequados para representar as matrizes das transformações geométricas, como rotação, translação e escalonamento em coordenadas homogêneas. A linguagem oferece dois mecanismos básicos para manipular matrizes4: a) o construtor `{v1,v2,v3,v4}` que permite criar matrizes4 em que `v1`, `v2`, `v3` e `v4` são vetores4, p. ex., `{[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7]}` cria uma matriz4, utilizando o construtor de vetores4; e b) o extrator que permite extrair ou modificar linhas da matriz4, p. ex., dada a matriz4 `m`, a expressão `m"1` extrai (ou modifica) a primeira linha da matriz4. As descrições de comandos, operadores e funções detalham a interpretação das matrizes4.

4.2.3.6 Tipo de valor indefinido

O valor indefinido indica que uma célula de memória ainda não recebeu um dos cinco valores básicos. Na iniciação da memória, todas as células serão fixadas como indefinidas. Valores indefinidos não podem participar da resolução de expressões, exceto nos operadores de igualdade (igual e diferente). As funções `def(v)` e `tipo(v)` retornam informações a respeito do tipo atual da expressão `v`.

4.2.3.7 Tipo de valores escalares

Valores escalares reúnem os valores inteiros e valores reais. Na resolução de expressões escalares, os valores inteiros são majorados para reais. Valores reais em ponto flutuante de dupla precisão preservam os valores inteiros sem perdas.

4.2.3.8 Tipo de valores numéricos

Valores numéricos reúnem os valores inteiros, escalares, vetor4 e matriz4. Alguns operadores aceitam diversas combinações de valores numéricos, oferecendo tratamentos específicos para cada combinação. Por exemplo, dois vetores4 podem ser somados um ao outro, normalizando as magnitudes `w`. Os comandos de atribuição distinguem os grupos de valores numéricos e lógicos.

4.2.3.9 Tipo de valores quaisquer

Valores quaisquer reúnem os valores de todos os tipos básicos e o valor indefinido. As funções `def()` e `tipo()`, e o comando `mostra()` aceitam valores de qualquer tipo.

4.2.4 Células de memória

O armazenamento de valores durante a execução de um programa é realizada pelas células de memória. As células de memória compõem um arranjo linear, dividido em quatro áreas: a) as células globais, que guardam valores acessíveis por todas as unidades; b) as células locais, que guardam valores acessíveis por determinada unidade; c) as células de parâmetros, alocadas dinamicamente e utilizadas para intercâmbio de valores entre unidades chamadoras e chamadas; e d) as células temporárias, alocadas dinamicamente e associadas à pilha de execução da calculadora RPN.

As células globais estão associadas aos nomes de variáveis globais. Os nomes de variáveis globais iniciam com o caractere `@`, e são únicos ao longo de todo o programa. As células locais estão associadas aos nomes de variáveis locais. As células de parâmetros estão associadas aos nomes dos parâmetros. Cada unidade possui seu próprio domínio de nomes, que compreende os nomes de variáveis locais e de parâmetros.

As células de memória são polimorfas, isto é, adequam-se ao tipo do valor que armazenam em determinado momento durante a execução de um programa. Inicialmente todas a células são fixadas com o valor indefinido.

4.2.5 Comandos guardados

O mecanismo de pós-guarda permite que um comando pós-guardado seja executado efetivamente somente quando sua guarda estiver aberta, ou seja, a condição associada ao comando for verdadeira. Todos os comandos podem ser pós-guardados.

Exemplos:

```
1  mostra(v) ?B1 --executa o comando mostra se B1 verdadeiro
2  m:={} ?(tipo(m)<>5) --modifica o valor de m se não for matriz4
3  repete fim ?B4 --executa a repetição se B4 verdadeiro
4  desenhaArco(raio,ângulo) ?B1 e B2 ou ~B3 --pós-guarda complexa
```

Os comandos devem ser pós-guardados em situações de opcionalidade; no caso da execução de sequências de comandos alternativas, deve-se utilizar o comando `seleciona,` especialmente no caso de grupos de comandos, e também para efeito de documentação.

4.2.6 Comandos do OpenGL

Os comandos do OpenGL incluídos na linguagem preservam a sintaxe original das bibliotecas `gl\gl.h` e `gl\glu.h`. Os comandos nativos que recebem arranjos (*arrays*) como parâmetros, p. ex., `glVertex3fv(v)`, não operam com ponteiros, uma vez que a linguagem não implementa estes tipos de valores. Os comandos do OpenGL não estão descritos nesta especificação e devem ser consultados diretamente nos respectivos manuais do OpenGL.

4.2.7 Pseudo comandos OpenGL

Os comandos denominados pseudo comandos do OpenGL não são oferecidos pela pelas bibliotecas `gl\gl.h` e `gl\glu.h`. No entanto, possuem sintaxe e funcionalidade semelhantes à comandos do OpenGL. Por exemplo, o pseudo comando `glTranslatefv(v)` recebe um parâ-

metros vetor4, e possui comportamento idêntico ao comando `glTranslatef(x,y,z)`, para `v=[x,y,z,1]`.

4.2.8 Comandos de script

A linguagem comporta oito comandos de script, perfazendo atividades de controle e fluxo e de ação. Os comandos de controle de fluxo tratam da sequenciação da execução, implementando mecanismos de sequência, seleção, repetição e términos. Os comandos de ação, juntamente com os comandos específicos para o OpenGL, realizam o trabalho de computação e memória de valores, depuração, e de construção de cenários gráficos.

Os comandos são:

a) `aborta`;
b) atribuição;
c) chamada de sub-rotina;
d) `mostra`;
e) `randomiza`;
f) `repete` … `fim`;
g) `seleciona` … `fim`;
h) `termina`.

4.2.8.1 Comando aborta

O comando `aborta` permite parar a sessão de execução, de forma semelhante a causas por erro. O motivo do aborto deve ser indicado por um valor positivo maior que zero. O comando mostra uma mensagem na janela de eventos na qual consta o motivo do aborto.

Exemplos:

```
1  aborta(4)  --para o programa com o motivo 4
2  aborta(-2) --motivo inválido causa erro
```

4.2.8.2 Comando de atribuição

O comando de atribuição possui quatro modalidades: a) atribuição pura, b) atribuição associada ao extrator de linhas de matrizes4, c) atribuição associada ao extrator de valores de vetores4, e d) atribuição combinada dos extratores de linhas e valores de matrizes4. A modalidade a), atribuição pura, resolve uma expressão qualquer e atribui o resultado a uma variável, que a partir deste momento assume o tipo do valor resultante da expressão.

A modalidade b) requer ao lado esquerdo uma variável matriz4 associada ao extrator de linhas, e ao lado direito uma expressão vetor4. A atribuição resulta na substituição da linha referenciada no extrator pelos valor da expressão vetor4. Por exemplo, seja `m` uma variável matriz4, e `v` uma expressão vetor4, então `m"2 := v` resulta em um novo valor da variável `m` em que a segunda linha foi substituída pelo valor do vetor4 `v`.

A modalidade c) requer ao lado esquerdo uma variável vetor4 associada ao extrator de partes, e ao lado direito uma expressão escalar. A atribuição resulta na substituição da parte ou valor referenciada no extrator pelo valor da expressão escalar. Por exemplo, seja `v` uma variável ve-

tor4, e p uma expressão escalar, então v'z, v'b, v'v ou v'1 := p resultam em um novo valor da variável v em que a terceira parte ou valor foi substituída pelo valor do escalar p.

A modalidade d) combina as modalidades b) e c) em cascata, ou seja, requer ao lado esquerdo uma variável matriz4 associada ao um extrator de linhas seguido por um extrator de partes ou valores, e ao lado direito uma expressão escalar. A atribuição resulta na substituição da parte ou valor referenciada no extrator de partes ou valores na linha referenciada pelo extrator de linhas pelo valor da expressão escalar. Por exemplo, seja m uma variável matriz4, e p uma expressão escalar, então m"3'y := p resulta em um novo valor da variável m em que a segunda parte ou valor da terceira linha foi substituída pelo valor do escalar p.

Exemplos:

```
1  b := 3>5          --atribuição de um valor lógico
2  i := 12345        --atribuição de valor inteiro
3  r := 10.4E-5      --atribuição de um valor real
4  v := [1,1,0,1]    --atribuição de um vetor4
5  m := {}           --atribuição de uma matriz4 (identidade)
6  m"2 := [2,3,4,5] --substituição da linha 2 da matriz4
7  v'y := 2.67       --substituição de y do vetor4
8  m"1'w := 1        --substituição de w da linha 1 da matriz4
```

Nos exemplos, as linhas 1 a 5 são atribuições puras, e as linhas 6, 7 e 8 são atribuições das modalidades com extratores.

4.2.8.3 Comando de chamada de unidades

As chamadas de sub-rotinas permitem modularizar o programa, e reaproveitar o código. A passagem de parâmetros entre a unidade chamadora e a unidade chamada ocorre por passagem de valores nas células de parâmetros. Os parâmetros podem ser de dois tipos: a) parâmetros unidirecionais, que transportam valores no sentido unidade chamadora unidade chamada, e b) parâmetros bidirecionais, que transportam valores nos dois sentidos.

Os parâmetros atuais na unidade chamadora são expressões. Os valores dos parâmetros atuais são capturados pelos parâmetros formais, que se comportam como variáveis locais da unidade chamada. Os parâmetros bidirecionais são identificados pelo caractere @ preposto (mas não faz parte!) ao nome do parâmetro. O retorno pode ser capturado pela unidade chamadora somente se o parâmetro atual correspondente for uma expressão primária formada por uma variável. Variáveis globais podem ser acessadas tanto na unidade chamadora como na unidade chamada.

Exemplo:

```
1  hipotenusa(3,4,v3)
2  hipotenusa(3,4,0)
3  mostra(v3)
4  ==>
5  hipotenusa(p1,p2,@p3)
6  p3:=sqrt(p1^2+p2^2)
```

A sub-rotina calcula a hipotenusa de um triângulo retângulo com lados p1 e p2, e retorna o valor através do parâmetro bidirecional p3. Os parâmetros p1, p2 e p3 transportam valores da unidade chamadora para a unidade chamada, e o parâmetro p3 transporta valores da unidade chamada para a unidade chamadora. A chamada da linha 1 recebe o retorno na variável v3,

porque a célula da variável coincide com a célula do resultado da expressão. A chamada da linha 2 também recebe o retorno, mas na célula temporária do resultado da expressão, à qual o programa não tem acesso após a chamada.

O esquema de chamadas de sub-rotinas não permite chamadas recorrentes, uma vez que cenários típicos de computação gráfica são composições hierárquicas. Sempre que um objeto estiver dentro de outro com o mesmo formato (p. ex., uma esfera dentro de uma esfera), deve existir uma rotina que organiza esta composição. Em geral a recorrência pode ser rescrita com o uso de repetições.

4.2.8.4 Comando executa

O objetivo do comando `executa` é agrupar uma sequência de comandos em um bloco. Quando o comando é pós-guardado, todo o bloco é executado condicionalmente.

Exemplo:

```
1  executa
2    b := 5*cos(alfa)
3    mostra(b) ?b>0.5
4  fim ?B1
```

No exemplo acima, o bloco será executado somente se `B1` for verdadeiro. Além disso, o comando da linha 3 também é pós-guardado, sendo executado ou não, dependendo da condição associada.

4.2.8.5 Comando mostra

O objetivo principal do comando `mostra` é apoiar a depuração de programas. O comando mostra na janela de eventos os valores de uma série de expressões, ou simplesmente marca o ponto em que ocorreu durante a execução do programa. A linguagem oferece duas funções importantes para depuração: `def()` e `tipo()`, que permitem verificar se determinado valor está definido, ou qual o tipo do valor. A combinação inadequada de operandos e operadores gera valores inválidos, e para indicar este fato, o comando mostra um `X`.

Exemplos:

```
1  mostra()         --Mostra! (só para marcar presença)
2  mostra(v)        --Mostra: ? (valor indefinido)
3  mostra(3+v)      --Mostra: X (valor inválido)
4  mostra(tipo(PI)) --Mostra: 3 (tipo real)
5  mostra([],{})    --Mostra: [...], {...} (vetor4 e matriz4)
```

O custo computacional do comando `mostra` é muito elevado, violando o ritmo de execuções do programa, portanto, deve ser utilizado apenas para fins de depuração, e de preferência no modo de execução passo à passo. A execução do comando `mostra` pode ser ativada/desativada a partir da interface de cenários.

4.2.8.6 Comando randomiza

A linguagem implementa um gerador de números pseudo aleatórios, que atende a função `random()` na geração de valores aleatórios. O comando `randomiza` fixa a semente do gerador

de duas formas: a) sem parâmetros, caso em que a semente é um valor arbitrário; e b) com um parâmetro inteiro que específica a nova semente.

O gerador inicia com semente arbitrária na primeira vez em que o programa é executado após a compilação. A função `random()` gera a mesma série de valores a partir do momento em que determinada semente for fixada.

4.2.8.7 Comando repete

O comando `repete` implementa a capacidade de repetir determinadas sequências de comandos. O comando não possui controle explícito do número de iterações. O corpo do comando é formado por uma série de guardas, em que cada guarda é formada por uma expressão condicional e uma lista de comandos associados. No início de cada iteração, as condições de todas as guardas serão resolvidas, podendo resultar em abertas (valor `verdadeiro`) ou fechadas (valor `falso`). Em seguida os comandos de cada guarda aberta serão executados, na ordem em que estão declaradas.

A iteração em que nenhuma guarda estiver aberta, determina o final do comando. Para evitar um número de iterações interminável, existe um limite de iterações possíveis por execução do programa, considerando a soma de todas as iterações dos comandos de repetição até o momento.

Existem duas guardas especiais com as condições: a) `inicial`, que estará aberta com exclusividade na primeira iteração, e b) `final`, que estará aberta com exclusividade na última iteração, determinada pela ausência de outras guardas abertas. Com o uso destas guardas é possível iniciar e terminar todo o contexto da repetição no corpo do comando.

Exemplo:

```
1  repete
2    inicial? i:=1 fim
3    i<=6? mostra(i) i:=i+1 fim
4    i=6? i:=8 mostra(i) fim
5    i=7? mostra() fim --nunca ocorre!
6    final? mostra(i) fim
7  fim
```

Na primeira iteração a guarda `inicial?` (linha 2) estará aberta e o valor variável `i` será fixado em `1`. Nas próximas iterações a guarda `i<=6?` (linha 3) estará aberta para os valores de `i` iguais a 1, 2, 3, 4, 5 e 6, e a guarda `i=6?` (linha 4) estará aberta para o valor de `i` igual a 6. Na última iteração da linha 3, o valor de `i` será fixado em `7`, mas a guarda da linha 4, resolvida no entrada da iteração, fixará o valor de `i` em `8`. Assim, `i=7` não será estável (invariante de iterações) , e a guarda `i=7?` (linha 5) nunca será executada. A guarda da `final?` (linha 6) ocorre quando nenhuma outra guarda estiver aberta, e será a última iteração.

O comando `repete` possui uma cláusula de retardo que estabelece um prazo, em milissegundos, para o progresso das iterações. O comando passa para a próxima iteração (não se aplica à iteração da guarda inicial) somente após a decorrência do prazo. Enquanto isso, o programa será interrompido para visualizar o desenho produzido até o momento, e será iniciada nova execução que progredirá até a iteração interrompida. A interface do cenário permite ativar e desativar o efeito da cláusula de retardo, além de oferecer a variável `TEMPO` para dimensionar o prazo dinamicamente.

Exemplo:

```
1  repete delta 500
2    inicial? i:=1 fim
3    i<=6? reta([i-1],[i]) i:=i+1 fim
4  fim
```

No exemplo, após a primeira iteração (execução da linha 3), o programa será interrompido. Em seguida iniciará nova execução, e enquanto não decorrer o prazo de 500ms, executará a primeira iteração (a linha 3). Decorrido o prazo executará a segunda iteração (duas vezes a linha 3), e assim sucessivamente até terminar o comando de repetição. O efeito visual sobre o desenho é o de mostrar sua construção passo à passo.

4.2.8.8 Comando seleciona

O comando `seleciona` implementa a capacidade de executar determinadas sequências alternativas de comandos. O comando não possui estrutura condicional nos moldes tradicionais. O corpo do comando é formado por uma série de guardas, em que cada guarda é formada por uma expressão condicional e uma lista de comandos associados. Inicialmente as condições de todas as guardas serão resolvidas, podendo resultar em abertas (valor `verdadeiro`) ou fechadas (valor `falso`). Em seguida os comandos de cada guarda aberta serão executados, na ordem em que estão declaradas.

O comando requer que pelo menos uma guarda esteja aberta, do contrário causará erro. Existe uma guarda especial com a condição `defalta`, que estará aberta com exclusividade sempre que nenhuma outra guarda estiver aberta. Com o uso desta guarda é possível contornar o requisito da necessidade de guardas abertas.

Exemplo:

```
1  seleciona
2    B1? reta(v1,v2) fim
3    B2? reta(v2,v3) fim
4    B3? reta(v3,v1) fim
5    defalta? mostra() fim
6  fim
```

Considere-se `B1`, `B2` e `B3` variáveis com valores lógicos (oferecidos pela interface do cenário). Quando o comando `seleciona` for executado, as condições das guardas serão resolvidas, e as guardas abertas serão executadas. A três guardas apresentam sete combinações possíveis de aberto/fechado, e a oitava possibilidade recai na guarda `defalta`.

4.2.8.9 Comando termina

O comando `termina` com parâmetros quebra a execução de comandos `repete`. O comando pode ocorrer em qualquer uma das guardas. Considerando uma construção hierárquica com vários níveis envolventes de comandos `repete`, o parâmetro do comando `termina` determina qual nível envolvente deve ser afetado. Os níveis são enumerados de dentro para fora: `0` refere-se ao nível envolvente mais próximo, `1` ao nível anterior ao mais próximo, `2` ao nível anterior ao anterior, e assim por diante.

Exemplo:

```
1  termina(0) --termina o laço envolvente mais próximo
2  termina(1) --termina o segundo laço envolvente mais próximo
```

A contagem invertida dos níveis de afetação do comando `termina` torna o sistema de referências relativa à posição do comando na hierarquia de repetições.

Sem parâmetros, o comando `termina` quebra a execução da sub-rotina corrente. É importante observar a consistências dos pares de comandos glBegin/glEnd, glPushMatrix/glPopMatrix e glPushAttrib/glPopAttrib.

Exemplo:

```
3  termina()  --termina a execução da sub-rotina.
```

4.2.9 Descrição das expressões

As expressões são formadas por operandos e operadores. Os operadores são organizadas em oito níveis de precedência, ordenados do menor para o maior nível. O quadro 4.3 mostra os níveis de precedências, os grupos de operadores e operadores em cada nível, e a associatividade dos operadores nos grupos.

precedência	grupos de operadores	operadores	associatividade
8	primários	`[...] {...} (...)`	-
7	unários pós-fixados	`! "T "i 'i`	esquerda
6	unários pré-fixados	`+ - ~`	direita
5	potenciação	`^`	direita
4	multiplicativos, produto	`* / div mod >< #`	esquerda
3	aditivos	`+ -`	esquerda
2	igualdade e relação	`= <> < <= > >=`	-
1	lógicos	`e equ ou xou`	esquerda

Quadro 4.3 - Relação de precedência de operadores

A associatividade dos operadores determina sua ordem de execução, que pode ser da esquerda para direita, p. ex., `a+b-c` é resolvido da forma `((a+b)-c)`, e da direita para a esquerda, p. ex., `a^b^c` é resolvido da forma `(a^(b^c))`. A precedência e a associação dos operadores pode ser modificada pelo uso do operador de subexpressão `(...)` – expressão entre parênteses – p. ex., `(a+b)*c` modifica a precedência da soma.

As variáveis globais, locais e externas, as constantes reais, inteiras e hexadecimais, e as funções são consideradas expressões primárias, e ocupam o nível mais alto de precedência.

As seções que seguem descrevem todos os operadores, considerando o modo de operação, as variantes, os tipos de valores e a comutabilidade dos operandos envolvidos. As combinações de tipos de valores não apresentados são proibidas, e causam erros de execução.

4.2.9.1 Operadores lógicos

Os operadores lógicos, `e`, `equ`, `ou` e `xou`, operam sobre valores lógicos e retorna valores lógicos. O quadro 4.4 mostra os resultados das operações.

operandos		**operadores lógicos**			
op1	**op2**	**e**	**equ**	**ou**	**xou**
V	V	V	V	V	F
V	F	F	F	V	V
F	V	F	F	V	V
F	F	F	V	F	F

Quadro 4.4 - Operadores lógicos

4.2.9.2 Operadores de igualdade

Os operadores de igualdade, `igual` e `diferente`, comparam os valores de operandos numéricos. O quadro 4.5 mostra as combinações de tipos de valores e os resultados das operações. (Valores lógicos podem ser *comparados*, utilizando-se os operadores `equ` e `xou`.)

operandos			**descrição dos operadores igual (=) e diferente (<>)**
op1		**op2**	
vetor4	**=** **<>**	vetor4	O operador `igual` retorna `verdadeiro`, e o operador `diferente` retorna `falso`, se os valores das partes `x-y-z-w` correspondentes de op1 e op2 forem iguais. A operação não considera as coordenadas homogêneas.
escalar		escalar	O operador `igual` retorna `verdadeiro`, e o operador `diferente` retorna `falso`, se os valores de op1 e op2 forem iguais. Aceita todas as combinações de valores reais e inteiros.
indefinido		indefinido	O operador `igual` retorna `verdadeiro`, e o operador `diferente` retorna `falso`, se os valores de op1 e op2 forem indefinidos.

Quadro 4.5 - Operadores de igualdade

4.2.9.3 Operadores relacionais

Os operadores relacionais, `menor`, `menor ou igual`, `maior` e `maior ou igual`, comparam os valores de operandos escalares. O quadro 4.6 mostra as combinações de tipos de valores e os

resultados das operações. (Valores lógicos, vetor4 e matriz4 não guardam relações de precedência.)

operandos			descrição dos operadores menor (<), menor ou igual (<=), maior (>) e maior ou igual (>=)
op1		**op2**	

escalar	`<`
	`<=`
	`>`
	`>=`
		escalar	Os operadores comparam os valores dos operandos numericamente, e retornam `verdadeiro` se a relação entre os valores for verificada, senão retorna `falso`. Aceita todas as combinações de valores reais e inteiros.

Quadro 4.6 - Operadores relacionais

4.2.9.4 Operadores aditivos

Os operadores aditivos, `soma` e `subtração`, operam sobre dois operandos numéricos. O quadro 4.7 mostra as combinações de tipos e os resultados das operações.

operandos			descrição dos operadores soma (+) e subtração (–)
op1		**op2**	
vetor4		vetor4	O operador `soma` adiciona as partes `x-y-z` dos valores de op1 e op2 e retorna o vetor4 resultante. O operador `subtração` subtrai as partes `x-y-z` do valor de op2 do valor de op2 e retorna o vetor4 resultante. Os dois operadores são normalizados para `w=1`, e o resultado tem `w=1`. Se algum dos operandos tiver `w=0`, ocorre um erro de execução.
real	`+` `-`	escalar	O operador `soma` adiciona os valores de op1 e op2. O operador `subtração` subtrai o valor de op2 do valor de op1. O resultado das operações é um valor real.
escalar		real	
inteiro		inteiro	O operador `soma` adiciona os valores de op1 e op2. O operador `subtração` subtrai o valor de op2 do valor de op1. O resultado das operações é um valor inteiro. Se o resultado exceder a capacidade dos inteiros, ocorre um erro de execução.

Quadro 4.7 - Operadores aditivos

No caso da soma ou subtração de dois valores inteiros cujo resultado exceda a capacidade dos inteiros, um dos valores pode ser majorado explicitamente, utilizando-se a função `real()`.

4.2.9.5 Operadores multiplicativos

Os operadores multiplicativos, `multiplicação`, `divisão real`, `divisão inteira` e `resto de divisão inteira`, operam sobre dois operandos numéricos de modo escalar. O quadro 4.8 mostra as combinações de tipos e os resultados das operações.

operandos op1		operandos op2	descrição dos operadores multiplicação (*), divisão real (/), divisão inteira (div) e resto de divisão inteira (mod)
vetor4	*	escalar	O operador de `multiplicação` multiplica as partes `x-y-z` do vetor4 pelo valor escalar e retorna um vetor4.
escala		vetor4	
vetor4	/	escalar	O operador de `divisão real` divide as partes `x-y-z` do vetor4 pelo valor escalar e retorna um vetor4. Se o divisor for igual a 0, ocorre um erro de execução.
real	*	escalar	O operador de `multiplicação` multiplica op1 por op2. O resultado da operação é um valor real.
escalar		real	
escalar	/	escalar	O operador de `divisão real` divide op1 por op2. O resultado da operação é um valor real. Se o divisor for igual a 0, ocorre um erro de execução.
inteiro	* div mod	inteiro	O operador `multiplicação` multiplica op1 por op2. O operador `divisão inteira` divide op1 por op2 e retorna o quociente, e o operador `resto` retorna o resto da `divisão inteira`. O resultado das operações é um valor inteiro. Se o divisor for igual a 0, ou o resultado exceder a capacidade dos inteiros, ocorre um erro de execução.
inteiro		inteiro	
inteiro		inteiro	

Quadro 4.8 - Operadores multiplicativos

4.2.9.6 Operadores de produtos

Os operadores de produtos, `produto interno` e `produto vetorial`, operam sobre matrizes e vetores. O quadro 4.9 mostram as combinações de tipos e os resultados das operações.

operandos op1		operandos op2	descrição dos operadores produto vetorial (#) e produto interno (><)
vetor4	#	vetor4	O `produto vetorial` multiplica dois vetores4 e retorna um vetor4. Os vetores4 op1 e op2, e o vetor4 resultante são normalizados para `w=1`. Se algum dos operandos tiver `w=0`, ocorre um erro de execução.
matriz4	><	matriz4	O `produto interno` de duas matrizes4 multiplica as duas matrizes e retorna uma matriz4.
matriz4	><	vetor4	O `produto interno` de matriz4 por vetor4 multiplica a matriz4 pelo vetor4 e retorna um vetor4. Considera o op1 vetor linha e o op2 vetor coluna.
vetor4		matriz4	
vetor4	><	vetor4	O `produto interno` multiplica dois vetores4 e retorna um real. Considera o op1 vetor linha e o op2 vetor coluna.

Quadro 4.9 - Operadores produto interno

4.2.9.7 Operador de potenciação

O operador potenciação opera com escalares. A potenciação é único operador binário associativo à direita, p. ex., a expressão `a^b^c` equivale à expressão totalmente *parentetizada* `(a^(b^(c)))`. O quadro 4.10 mostra a combinação de tipos e o resultado da operação.

operandos op1		operandos op2	descrição do operador potenciação (^)
escalar	^	escalar	O operador `potenciação` eleva o operando op1 à potência de op2 e retorna um valor real.

Quadro 4.10 - Operador de potenciação

4.2.9.8 Operadores unários pré-fixados

Os operadores unários pré-fixados, `negativo`, `positivo` e `negação lógica`, operam sobre vetores4, escalares e valor lógicos. O quadro 4.11 mostra as combinações de tipos e os resultados das operações.

operando		op1	descrição dos operadores unários negativo (–), positivo (+) e negação lógica (~)
	– +	vetor4	O operador `negativo` retorna um vetor4 com o sinal das partes `x-y-z` do valor op1 trocados, o operador `positivo` retorna o próprio valor op1. O `w` do vetor4 permanece inalterado.
	– +	real	O operador `negativo` retorna o valor de op1 com sinal trocado, e o operador `positivo` retorna o próprio valor op1. O resultado tem o mesmo tipo do valor op1.
		inteiro	
	~	lógico	O operador `negação lógica` retorna o valor lógico de op1 invertido, de `verdadeiro` para `falso`, ou `falso` para `verdadeiro`.

Quadro 4.11 - Operadores unários pré-fixados

4.2.9.9 Operadores unários pós-fixados

Os operadores pós-fixados, `fatorial`, `transposta`, `extrator de linha` e `extrator de parte`, operam sobre matrizes4, vetores4 e inteiros. Alguns operadores possuem modificadores literais, que não são operandos. O quadro 4.12 mostra as combinações de tipos e os resultados das operações.

operando e modificador			descrição dos operadores unários pós-fixados fatorial (!), transposta ("T), extrator de linha ("i) e extrator de parte ('i)
op1		modif.	
inteiro	!		O operador fatorial retorna o fatorial do valor op1. O resultado é um valor real. Valores acima de 17 perdem a exatidão do fatorial.
matriz4	"	T	O operador transposta retorna uma matriz4 que é a matriz transposta do valor op1.
		1-2-3-4	O operador extrator de linha retorna o vetor4 do índice da linha do valor op1. Os índices das linhas podem ser 1, 2, 3 e 4.
vetor4	'	x-y-z-w	O operador extrator de parte retorna o valor real da parte indexada do valor de op1, interpretado como elemento de coordenadas homogêneas. O resultado da operação é normalizado para w=1. Os índices das partes podem ser x, y, z e w. (Veja exceção em 4.2.12)
		r-g-b-a h-s-v-a	O operador extrator de parte retorna o valor real da parte indexada do valor de op1, interpretado como elemento do sistema de cores RGB e HSV. Os índices das partes podem ser r, g, b e a ou h, s, v e a.
		1-2-3-4	O operador extrator de parte retorna o valor real da parte indexada do valor de op1. Os índices das partes podem ser 1, 2, 3 e 4.

Quadro 4.12 - Operadores unários pós-fixados

4.2.9.10 Construtor de vetores4

O construtor de vetores4 é um operador especial que retorna um valor vetor4 a partir de expressões escalares. Cada parte é gerada pela expressão escalar correspondente. O construtor apresenta cinco variantes, que permitem suprimir partes progressivamente, e que assumem valores padrão. A base dos valores padrão é a coordenada homogênea (0,0,0,1) ou a cor preta opaca (0,0,0,1).

Exemplos:

```
1  [x,y,z,w] --gera o vetor4 completamente definido
2  [1,0,1]   --gera o vetor4 [1,0,1,1], p. ex., a cor magenta
3  [r*cos(alfa),r*sen(alfa)] --com alfa=90, gera o vetor4 [r,0,0,1]
4  [v'x]     --supondo v=[2,1,4,1], gera o vetor4 [2,0,0,1]
5  []        --gera o vetor4 [0,0,0,1], com w=1 ou a=1
6  [2,módulo([x,y,z]),2,1] --uso recorrente do construtor
```

A linha 1 gera um vetor4 completamente definido, no caso com intenção de representar um elemento de coordenadas homogêneas. A linha 2 gera um vetor4 com a intenção de representar uma cor no sistema RGB, em que a última parte assume o valor a=1 para superfícies opacas (fator alfa). A linha 5 gera um valor padrão zerado, preparado para w=1 ou a=1.

Os vetores4 podem ser modificados através dos comandos de atribuição com extratores de partes, substituindo partes por novos valores escalares.

4.2.9.11 Construtor de matrizes4

O `construtor de matrizes4` é um operador especial que retorna um valor matriz4 a partir de expressões vetor4. Cada linha da matriz é gerada pela expressão vetor4 correspondente. O construtor apresenta cinco variantes, que permitem suprimir linhas progressivamente, e que assumem valores padrão. A base dos valores padrão é a matriz identidade.

Exemplos:

```
1  {v1,v2,v3,v4} --gera a matriz a partir dos vetores v1, v2, v3 e v4
2  {[cos(a),0,-sen(a)],[],[sen(a),0,cos(a)]} --gera matriz de rotação
3  {[Sx,0],[0,Sy]} --gera a matriz de escala em 2D
4  {[1,2,3,1]} --gera a matriz identidade com o vetor4 na linha 1
5  {} --gera a matriz identidade
```

A linha 1 gera uma matriz4 completa, utilizando os quadro vetores4 para as linhas da matriz. As linhas 2, 3 e 4 geram outros exemplos de matrizes úteis em computação gráfica. A linha 5 gera a matriz identidade `{[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]}`.

As matrizes4 podem ser modificadas através dos comandos de atribuição com extratores de linha substituindo linhas inteiras por novos vetores4, ou em combinação com extratores de partes, substituindo partes em linhas por novos valores escalares.

4.2.9.12 Operador de sub expressões

O `operador de sub-expressões` modifica a ordem de precedência e associação dos operadores, retornando valores de sub expressões para dentro de outras expressões. O uso do operador é recorrente.

Exemplos:

```
1  x^(2+5) --modifica a precedência da soma
2  (x^2)^3 --modifica a associatividade à direita
3  a/(b*c) --modifica a associatividade dos multiplicativos
4  (m"2+[2,1,1])'x --modifica a precedência da soma
```

A interpretação sem parênteses da expressão da linha 1 seria `((x^2)+5)`, da linha 2 seria `(x^(2^3))`, da linha 3 seria `((a/b)*c)`, da linha 4 seria `(m"2+([2,1,1]'x))`, que causaria um erro de execução na soma de valores incompatíveis.

4.2.10 Constantes e variáveis

Constantes são valores sem sinal declarados literalmente no texto do programa, e são armazenados em células da memória. Podem ser números inteiros escritos em decimal ou em hexadecimal, e números reais com frações e expoentes.

As variáveis podem ser internas ou externas. As variáveis internas são células de memória referenciadas por nomes simbólicos, e são dinamicamente vinculadas a tipos de valores. As variá-

veis externas são valores modificáveis através da interface do cenário, ou são constantes relevantes, p. ex., `PI`.

4.2.10.1 Constantes inteiras

As constantes inteiras são formadas pelos números naturais de 32 bites. As constantes inteiras geram valores inteiros. Valores que excedem a capacidade dos inteiros são automaticamente majorados para valores reais.

4.2.10.2 Constantes hexadecimais

As constantes hexadecimais são formados por no máximo 31 bites significativos (o bite de sinal deve ser igual a zero). As constantes hexadecimais geram valores inteiros. Valores que excedem a capacidade dos inteiros não são majorados, e causam erro de execução.

Exemplos:

```
1  0X00000001 --equivale ao valor 1
2  0X7FFFFFFF --é o maior valor possível
3  0X55a --as letras podem ser minúsculas
4  0X0 --X deve ser maiúsculo; valor igual a zero
```

As constantes hexadecimais são úteis para definir máscaras para linhas tracejadas, e outros objetos que dependem da interpretação binária dos valores inteiros. Por exemplo, o comando OpenGL `glLineStipple(1,0X5555)` define o padrão de tracejado `1010101010101010` – ponto espaço ponto espaço...

4.2.10.3 Constantes reais

As constantes reais são formados pelos valores positivos em ponto flutuante, armazenados em dupla precisão. A notação permite declarar partes inteiras, partes fracionárias e expoentes base 10. Os valores que excedem a capacidade do processador causam erro de execução.

Exemplos:

```
1  1.234   --valor decimal
2  0.23E-3 --equivale a 0.00023
3  12E+30  --é um valor grande
4  0E0     --representa o valor real zero (não é inteiro)
5  0.0     --representa o valor real zero (não é inteiro)
```

Os valores inteiros são majorados automaticamente para valores reais, sempre que o contexto em que ocorrem requer valores escalares. Podem ser majorados explicitamente, utilizando-se a função `real()`.

4.2.10.4 Variáveis internas

As variáveis de armazenamento são identificadas por nomes de variáveis, e são divididas em três grupos: a) variáveis globais, que iniciam com o símbolo @, e são acessíveis por todas as unidades do programa; b) variáveis locais, que são acessíveis somente pela unidade em que ocorrem; e c) parâmetros que figuram como variáveis locais. As variáveis são de tipagem dinâmica,

e assumem o tipo no momento em que recebem um valor. A seção 4.2.4 descreve os aspectos relevantes das variáveis em relação às células de memória.

4.2.10.5 Variáveis externas

As variáveis externas são divididas em três grupos: a) constantes com valores fixos; b) variáveis captados pela interface do cenário; e c) variáveis controladas pelo relógio. O quadro 4.13 mostra a relação das variáveis externas.

	tipo de valor	descrição da variável externa
`Bi`	variáveis lógicas	Valores lógicos captados pela interface do cenário.
`Ki`	variáveis reais	Valores reais captados pela interface do cenário.
`Qi`, `QMK`	variáveis vetor4	Valores vetor4 captados pela interface do cenário.
`F`, `V`	constantes lógicas	Valores constantes `falso` e `verdadeiro`.
`GRAUS`	variável inteira	Valor progressivo em graus controlado pelo relógio e pelas opções de passo, sentido e modo a partir da interface do cenário. No modo absoluto repassa o ângulo, e no modo incremento repassa a diferença entre o ângulo anterior e o corrente.
`PI`	constante real	Valor constante de `PI` em precisão dupla.
`SEQ`	variável inteira	Contador sequencial que inicia em zero com o acionamento da partida contínua ou simples, e é incrementado a cada execução do programa.
`TEMPO`	variável inteira	Valor inteiro captado pela interface do cenário.

Quadro 4.13 - Relação das variáveis externas

Os valores das variáveis captadas pela interface do cenário são apresentados ao programa a cada ciclo de execução. As variáveis podem ser utilizadas para dar movimento aos desenhos. As variáveis externas não podem ser modificadas pelo programa.

4.2.11 Funções

As funções são sub-rotinas embutidas na linguagem de script, que podem ser chamadas em expressões. As funções estão divididas em sete grupos:

a) funções escalares;
b) funções de tipagem;
c) funções trigonométricas;
d) funções elípticas;
e) funções logarítmicas;
f) funções vetoriais;
g) funções sobre cores; e
h) funções de aleatórios.

As seções a seguir abordam cada função, descrevendo seus argumentos e suas funcionalidades.

4.2.11.1 Funções escalares

função	argumento		descrição do argumento
abs (valor)	`valor`	escalar	Valor relativo.
	retorno	escalar	Valor absoluto.
	Calcula o valor absoluto de `valor`. O retorno é do mesmo tipo do `valor`.		
ângulo (valor)	`valor`	escalar	Ângulo não normalizado.
	retorno	escalar	Ângulo normalizado.
	Normaliza o ângulo. Se `valor>=0`, normaliza o ângulo para o intervalo 0 a 360 graus. Se `valor<0`, normaliza o ângulo para o intervalo -360 a 0 graus.		
int (valor)	`valor`	escalar	Inteiro ou real.
	retorno	inteiro	Valor `(int)valor`.
	Trunca `valor` para o maior inteiro menor que `valor`. Se o inteiro resultante exceder a capacidade, ocorre erro de execução.		
max (valor1,...,valor*n*)	`valori`	escalar	Lista de um ou mais valores.
	retorno	escalar	Maior valor da lista de valores.
	Calcula o valor máximo da lista de valores. Se todos os valores `valori` forem inteiros, retorna um valor inteiro, senão retorna um valor real.		
min (valor1,...,valor*n*)	`valori`	escalar	Lista de um ou mais valores.
	retorno	escalar	Menor valor da lista de valores.
	Calcula o valor mínimo da lista de valores. Se todos os valores `valori` forem inteiros, retorna um valor inteiro, senão retorna um valor real.		
real (valor)	`valor`	escalar	Valor inteiro ou real.
	retorno	real	Valor `(double)valor`.
	Converte o `valor` inteiro para real.		
sinal (valor)	`valor`	escalar	Valor.
	retorno	inteiro	Valor unitário com sinal.
	Se `valor<0` retorna -1, se `valor>0` retorna +1, senão retorna 0.		
sqrt (valor)	`valor`	escalar	`valor>=0`.
	retorno	real	Raiz quadrada do `valor`.
	Calcula a raiz quadrada do `valor`.		

4.2.11.2 Funções de tipagem

função	argumento		descrição do argumento
def (valor)	`valor`	qualquer	Valor de qualquer tipo, inclusive indefinido.
	retorno	lógico	Verdadeiro se o tipo for definido.
	Verifica o tipo do `valor` e retorna `verdadeiro` se o tipo for definido, senão retorna `falso`.		
tipo (valor)	`valor`	qualquer	Valor de qualquer tipo, inclusive indefinido.
	retorno	inteiro	Índice do tipo.
	Determina o tipo do `valor` e retorna o índice do tipo: 0 para indefinido, 1 para valor lógico, 2 para valor inteiro, 3 para valor real, 4 para vetor4, e 5 para matriz4.		

4.2.11.3 Funções trigonométricas

função	argumento		descrição do argumento
acos (valor)	`valor`	escalar	Valor do cosseno.
	retorno	real	Ângulo em graus.
	Calcula o arco do cosseno em graus.		
asen (valor)	`valor`	escalar	Valor do seno.
	retorno	real	Ângulo em graus.
	Calcula o arco do seno em graus.		
atan (valor)	`valor`	escalar	Valor da tangente.
	retorno	real	Ângulo em graus.
	Calcula o arco da tangente em graus.		
cos (ângulo)	`ângulo`	escalar	Ângulo em graus.
	retorno	real	Valor do cosseno.
	Calcula o cosseno do `ângulo.` O `ângulo` é dado em graus, e é reduzido ao intervalo de 0 a 360.		
cosh (ângulo)	`ângulo`	escalar	Ângulo em graus.
	retorno	real	Valor da tangente.
	Calcula o cosseno hiperbólico do `ângulo.` O `ângulo` é dado em graus, e é reduzido ao intervalo de 0 a 360.		

função	argumento		descrição do argumento
sen (ângulo)	`ângulo`	escalar	Ângulo em graus.
	retorno	real	Valor do seno.
	Calcula o seno do `ângulo`. O `ângulo` é dado em graus, e é reduzido ao intervalo de 0 a 360.		
senh (ângulo)	`ângulo`	escalar	Ângulo em graus.
	retorno	real	Valor da tangente.
	Calcula o seno hiperbólico do `ângulo`. O `ângulo` é dado em graus, e é reduzido ao intervalo de 0 a 360.		
tan (ângulo)	`ângulo`	escalar	Ângulo em graus.
	retorno	real	Valor da tangente.
	Calcula a tangente do `ângulo`. O `ângulo` é dado em graus, e é reduzido ao intervalo de 0 a 360.		
tanh (ângulo)	`ângulo`	escalar	Ângulo em graus.
	retorno	real	Valor da tangente.
	Calcula a tangente hiperbólica do `ângulo`. O `ângulo` é dado em graus, e é reduzido ao intervalo de 0 a 360.		

4.2.11.4 Funções elípticas

A figura 4.14 mostra a elipse com o semieixo maior `a`, o semieixo menor `b`, e o ângulo `ϕ` formado pelo semieixo maior e o vetor `r`.

A equação polar (4.1) relaciona as componentes `r`, `a`, `b` e `ϕ`, considerando o centro da elipse. A partir desta equação pode-se definir funções que facilitam a geração de elipses em um plano.

$$r = \frac{a \cdot b}{\sqrt{b^2 \cdot \cos^2\langle\rangle + a^2 \cdot \sin^2\langle\rangle}} \tag{4.1}$$

As equações (4.2) definem o seno e cosseno elípticos, `senel()` e `cosel()`, que permitem calcular a posição de um ponto `P(x,y)` sobre a elipse em função da relação entre os semieixos `ρ`, e do semieixo maior `a`. A equação (4.3) permite calcular o módulo do vetor `<x,y>`.

$$\begin{aligned} \rho &= a/b \\ x &= a \cdot cosel\langle, \rho\rangle \\ y &= a \cdot senel\langle, \rho\rangle \end{aligned} \tag{4.2}$$

$$r = \sqrt{x^2 + y^2} \tag{4.3}$$

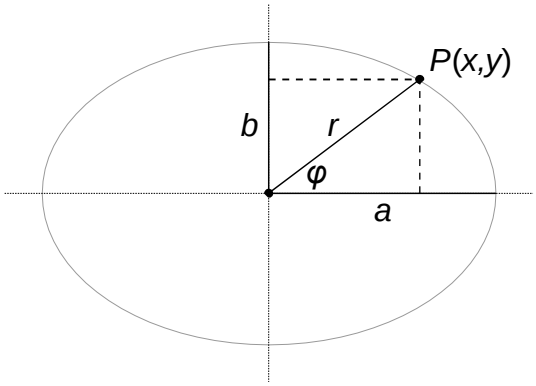


Figura 4.14 - Elipse e suas componentes

função	argumento		descrição do argumento
cosel (ângulo, rho)	`ângulo`	escalar	Ângulo em relação ao semieixo maior.
	`rho`	escalar	Relação entre o semieixo maior e o semieixo menor, `rho>0`.
	retorno	real	Cosseno elíptico.
	Calcula o cosseno elíptico do `ângulo` em função de `rho`. O `ângulo` é dado em graus, e é reduzido ao intervalo de 0 a 360.		
cosel (ângulo, a, b)	`ângulo`	escalar	Ângulo em relação ao semieixo maior.
	`a`	escalar	Comprimento do semieixo maior, `a>0`.
	`b`	escalar	Comprimento do semieixo menor, `b>0`.
	retorno	real	Cosseno elíptico.
	Calcula o cosseno elíptico do `ângulo` em função de `a/b`. O `ângulo` é dado em graus, e é reduzido ao intervalo de 0 a 360 graus.		
senel (ângulo, rho)	`ângulo`	escalar	Ângulo em relação ao semieixo maior.
	`rho`	escalar	Relação entre o semieixo maior e o semieixo menor, `rho>0`.
	retorno	real	Seno elíptico.
	Calcula o seno elíptico do `ângulo` em função de `rho.` O `ângulo` é dado em graus, e é reduzido ao intervalo de 0 a 360.		
senel (ângulo, a, b)	`ângulo`	escalar	Ângulo em relação ao semieixo maior.
	`a`	escalar	Comprimento do semieixo maior, `a>0`.
	`b`	escalar	Comprimento do semieixo menor, `b>0`.
	retorno	real	Seno elíptico.
	Calcula o seno elíptico do `ângulo` em função de `a/b`. O `ângulo` é dado em graus, e é reduzido ao intervalo de 0 a 360.		

4.2.11.5 Funções logarítmicas

função	argumento		descrição do argumento
expn (valor)	`valor`	escalar	Expoente.
	retorno	real	Valor `de` e^{valor}.
	Calcula e^{valor}.		
ln (valor)	`valor`	escalar	Valor.
	retorno	real	Log base `e` do `valor.`
	Calcula o `ln(valor)`.		

4.2.11.6 Funções vetoriais

A figura 4.15 (a) mostra o vetor **v**=<x,y,z>, e a figura 4.15 (b) mostra o esquema para a determinação do vetor normal **n** em relação aos vetores **u** e **v**. A equação 4.4 permite calcular o módulo do vetor <x,y,z>.

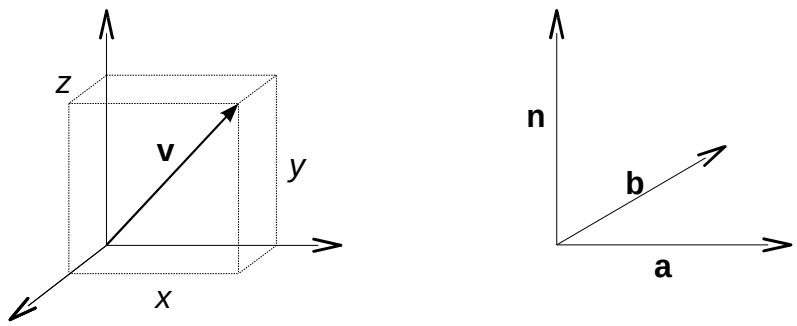
$$\|\boldsymbol{v}\| = \sqrt{\boldsymbol{v}.x^2 + \boldsymbol{v}.y^2 + \boldsymbol{v}.z^2} \tag{4.4}$$

A equação 4.5 permite calcular o vetor normal **n** em função dos vetor **a** e **b**, utilizando a regra da mão direita. O operador **x** realiza o produto vetorial.

$$\boldsymbol{n} = \boldsymbol{a} \times \boldsymbol{b} \tag{4.5}$$

A equação 4.6 permite calcular o vetor unitário **u**, dividindo os valores **v.**x, **v.**y e **v.**z pelo módulo do vetor **v**.

$$\boldsymbol{u} = \left\langle \frac{\boldsymbol{v}.x}{\|\boldsymbol{v}\|} \; \frac{\boldsymbol{v}.y}{\|\boldsymbol{v}\|} \; \frac{\boldsymbol{v}.z}{\|\boldsymbol{v}\|} \right\rangle \tag{4.6}$$



(a) (b)

Figura 4.15 - (a) Módulo, (b) Vetor normal

função	argumentos		descrição dos argumentos
módulo (vetor)	`vetor`	vetor4	Valor do vetor em coordenadas homogêneas.
	retorno	real	Valor do módulo.
	Calcula o módulo do `vetor` dado em coordenadas homogêneas.		
normal (v1, v2, v3)	`v1`	vetor4	Origem dos vetores `u` e `v` em coordenadas homogêneas.
	`v2`	vetor4	Alvo do vetor `u` em coordenadas homogêneas.
	`v3`	vetor4	Alvo do vetor `v` em coordenadas homogêneas.
	retorno	vetor4	Vetor normal com magnitude `w=1`.
	Calcula o vetor normal para os vetores `u` e `v`, utilizando o produto vetorial `u v`. O resultado da função é um vetor unitário. O cálculo utiliza a regra da mão direita.		
trocaw (valor1, valor2)	`valor1`	vetor4	Valor original em coordenadas homogêneas.
	`valor2`	escalar	Novo valor de magnitude `w`.
	retorno	vetor4	Valor modificado em coordenadas homogêneas.
	Troca o valor `w` do vetor4 `valor1` para `valor2`, e ajusta as partes `x`, `y` e `z` para o novo valor `w`.		
unitário (vetor)	`vetor`	vetor4	Valor do vetor em coordenadas homogêneas.
	retorno	vetor4	Vetor unitário com `w=1`.
	Reduz o `vetor` para vetor unitário, e ajusta as pates `x`, `y` e `z` para a magnitude `w=1`.		

4.2.11.7 Funções sobre cores

As cores podem ser representadas no sistema RGB(A) e no sistema HSV(A). No sistema tradicional RGB(A) as cores são decompostas em quatro componentes: vermelho, verde, azul e "*alfa blending*". Os valores situam-se entre 0 e 1, sendo 0 a ausência do componente e 1 a presença total do componente.

No sistema HSV(A) as cores também são representadas por quatro componentes: matiz, saturação, valor ou luminosidade e "*alfa blending*". A matiz representa a tonalidade da cor, e situa-se entre 0 e 360, os demais valores situam-se entre 0 e 1.

Valores vetor-4 podem ser interpretados como cores, tanto no sistema RGB(A) como no sistema HSV(A). No sistema RGB(A) o valor é interpretado por `[r,g,b,a]`, e no sistema HSV(A) é por `[h,s,v,a]`. Os extratores de valores utilizados devem ser r, g, b e a, ou 1, 2, 3 e 4. Assim, `[h,s,v,a]'3` extrai o componente `v`. É bom lembrar que o construtor de valores vetor-=4 assume como defalta o valor [0,0,0,1].

função	argumento		descrição do argumento
hsv (cor)	`cor`	vetor-4	Cor no sistema RGB(A).
	retorno	vetor-4	Cor no sistema HSV(A).
	Converte a cor do sistema RGB(A) para o sistema HSV(A).		
hsv (r, g, b, a)	`r`	escalar	Componente vermelho, `0<=r<=1`.
	`g`	escalar	Componente verde, `0<=g<=1`.
	`b`	escalar	Componente azul, `0<=b<=1`.
	`a`	escalar	Componente alfa, `0<=a<=1`.
	retorno	vetor-4	Cor no sistema HSV(A).
	Converte a cor do sistema RGB(A) para o sistema HSV(A).		
rgb (cor)	`cor`	vetor-4	Cor no sistema HSV(A).
	retorno	vetor-4	Cor no sistema RGB(A).
	Converte a cor do sistema HSV(A) para o sistema RGB(A).		
rgb (h, s, v, a)	`h`	escalar	Componente matiz, `0<=h<=360`.
	`s`	escalar	Componente saturação, `0<=s<=1`.
	`v`	escalar	Componente valor (luminosidade), `0<=l<=1`.
	`a`	escalar	Componente alfa, `0<=a<=1`.
	retorno	vetor-4	Cor no sistema RGB(A).
	Converte a cor do sistema HSV(A) para o sistema RGB(A).		

4.2.11.8 Funções de aleatórios

função	argumento		descrição do argumento
random ( )	*retorno*	real	Valor aleatório.
	Gera um valor aleatório entre 0.0 e 1.0... inclusive. O comando `normaliza` prepara a semente para a geração de séries de valores aleatórios.		
random (valor1,valor2)	`valor1`	inteiro	Limite inferior, `valor1>=0`.
	`valor2`	inteiro	Limite superior, `valor2>=0`.
	retorno	inteiro	Número aleatório.
	Gera um número aleatório no intervalo do limite inferior e superior, inclusive. Se `valor1>valor2`, os valores serão trocados. O comando `normaliza` prepara a semente para a geração de séries de números aleatórios.		

4.2.12 Considerações sobre a variável QMK

A variável vetor-4 `QMK` contém quatro valores coletados nos captores de teclas e do *mouse*. Os dois primeiros valores, `QMK'x` e `QMK'y`, contêm a posição do *mouse* sobre o captor, normalizada em relação à dimensão do captor. As formas `QMK'x` e `QMK'y` não estão sujeitas às regras de coordenadas homogêneas, e são tratadas com `QMK'1` e `QMK'2`, respectivamente.

O terceiro valor, `int(QMK'3)` em binário `0xCCMMKKKK`, é composto por três partes: a) `CC` contém o número do captor; b) o `MM` indica o botão do *mouse* acionado, 0 para nenhum, 1 para o botão esquerdo e 2 para o botão direito; e c) `KKKK` contém o código da tecla. O código da tecla pode ser interpretado em binário da forma `00000acskkkkkkkk`, em que `a=1` indica que a tecla `Alt` está acionada, `c=1` indica que a tecla `Ctrl` está acionada, `s=1` indica que a tecla `Shift` está acionada, e `k...kkk` é o código nativo da tecla.

O quarto valor, `QMK'4`, contém o incremento do rolador do *mouse.* Em vetores-4 de coordenadas homogêneas, este valor tipicamente representa o fator de normalização de `x`, `y` e `z`, que não se aplica a esta variável em particular.

4.3 OS PRAGMAS

Os `pragmas` compõem um conjunto de assertivas que estabelecem os valores originais para os dados captados pela interface do cenário. Devem ser declarados no início do programa, antes de qualquer comando. As assertivas podem ser distribuídas em uma ou mais sentenças com uma ou mais assertivas.

Exemplo:

```
1  PRAGMA B1=V, ROTAÇÃO=(V,V,100,F)
2  PRAGMA K1=12, K2=-100 --reta de controle
3  PRAGMA Q1=[-6.12,+7,0], glEnable(GL_NORMALIZE)
```

Na primeira execução após a criação ou da abertura de um programa, a interface é iniciada com os valores originais. Nas execuções subsequentes, a interface permanece no estado corrente, preservando as alterações realizadas pelo usuário. Os valores originais podem ser restaurados por opção do usuário.

4.3.1 Assertivas para as variáveis externas

As assertivas `Bi`, `Ki`, `Qi` e `TEMPO` fixam os valores iniciais das respectivas variáveis externas. O índice `i` deve ser substituído pela identificação da variável externa:

a) `Bi=V|F`, fixa os valores das variáveis externas lógicas `B1` a `B4`;
b) `Ki=valor`, fixa os valores das variáveis externas escalares `K1` a `K4`, com valores reais no intervalo -360 a 360;
c) `Qi=[x,y,z,w]`, fixa os valores das variáveis externas vetor-4 `Q1` a `Q8`, com valores construídos pelos valores reais `x`, `y`, `w` e `z` no intervalo -360 a 360; aceita os formatos `[]`, `[x]`, `[x,y]`, `[x,y,z]` e `[x,y,z,w]` baseados no valor inicial `[0,0,0,1]`;
d) `TEMPO=ms`, fixa o valor inteiro em milissegundos no intervalo 0 a 1000.

4.3.2 Assertivas para configuração de opções OpenGL

As assertivas para a configuração de opções OpenGL afetam diretamente as marcações de configuração acessíveis pela interface do cenário. São expressas utilizando-se a sintaxe dos comandos OpenGL subjacentes:

a) `glCullFace(opção)`, com as opções `GL_BACK`, `GL_FRONT` e `GL_FRONT_AND_BACK`;
b) `glDisable(opção)`, com as opções `GL_CULL_FACE` e `GL_NORMALIZE`;
c) `glEnable(opção)`, com as opções `GL_CULL_FACE` e `GL_NORMALIZE`;
d) `glFrontFace(opção)`, com as opções `GL_CCW` e `GL_CW`; e
e) `glShadeModel(opção)`, com as opções `GL_FLAT` e `GL_SMOOTH`.

Os comandos do OpenGL não estão descritos nesta especificação e devem ser consultados diretamente nos respectivos manuais do OpenGL.

4.3.3 Assertivas para opções de execução

As assertivas para opções de execução afetam diretamente as marcações e valores que ativam e desativam funcionalidade, uso de materiais e o passo do relógio (graus/s), acessíveis pela interface do cenário:

a) `DELTA=V|F`, fixa a opção de ativação da cláusula 'delta' nos comandos 'repete';
b) `MATERIAL=1|2|...|8`, fixa a seleção do material a ser usado no desenho;
c) `MOSTRA=V|F`, fixa a opção de ativação dos comandos 'mostra'; e
d) `ROTAÇÃO=(V|F,V|F,passo,V|F)`, fixa os parâmetros para o controle da rotação, na ordem: opção de incremento absoluto, opção rotação acionada, o passo da rotação, e a opção de rotação reversa.

A notação V|F indica verdadeiro (marcado) ou falso (não marcado); os materiais são numerados de 1 a 8 literalmente; e o passo é um número inteiro no intervalo 1 a 1000.

5 EXEMPLOS DE ALGORITMOS

As seções seguintes mostram alguns algoritmos, utilizando os recursos disponibilizados pela linguagem de script do ASA-Ensaio3D.

5.1 CÍRCULO VERSÃO 1

O algoritmo 5.1 desenha um círculo em 2D sobre o plano xy. O círculo possui `raio=K1` e é composto por `K2` segmentos. A linha 11 constrói o vértice inicial `v1` para `i=0`, e a linha 16 constrói o vértice final `v2` para um ângulo calculado em função do segmento corrente.

Algoritmo 5.1 - Desenho de um círculo (versão 1)

```
1    --K1: raio do círculo
2    --K2: segmentos de reta
3
4    repete
5       inicial?
6          raio := K1
7          seg := max(int(K2),1)
```

```
 8          i := 0
 9       fim
10       i=0?
11          v1 := [raio,0]
12          i := 1
13       fim
14       i>0 e i<=seg?
15          theta := i*360/seg
16          v2 := [raio*cos(theta),raio*sen(theta)]
17          glColor3f(1,0,0)
18          glBegin(GL_LINES)
19          glVertex3fv(v1)
20          glVertex3fv(v2)
21          glEnd()
22          v1 := v2
23          i := i+1
24       fim
25    fim
```

5.2 CÍRCULO VERSÃO 2

O algoritmo 5.2 desenha o mesmo círculo do algoritmo 5.1, utilizando outro método de geração de linhas. A opção `GL_LINE_STRIP` constrói uma linha conexa envolvendo todos os vértices. A cláusula `delta 300` na linha 06 faz com que o desenha seja visualizado segmento a segmento. Os pragmas fixam valores iniciais razoáveis para o programa.

Algoritmo 5.2 - Desenho de um círculo (versão 2)

```
01    --K1: raio do círculo
02    --K2: segmentos de reta
03
04    PRAGMA K1=6.0, K2=20, DELTA=V
05
06    glColor3f(1,0,0)
07    glBegin(GL_LINE_STRIP)
08    repete --delta 300
09       inicial?
10          raio := K1
11          seg := max(int(K2),1)
12          i := 0
13       fim
14       i<=seg?
15          theta := i*360/seg
16          v := [raio*cos(theta),raio*sen(theta)]
17          glVertex3fv(v)
18          i := i+1
19       fim
20    fim
21    glEnd()
```

5.3 ROTAÇÃO E TRANSLAÇÃO DE UM PONTO

O algoritmo 5.3 rotaciona e depois translada um ponto 2D sobre o plano xy. A linha 07 inicia o ponto baseado no raio do círculo. A linha 08 cria a matriz de rotação `m`. A linha 09 multiplica (produto interno) a matriz de rotação pelo vetor do ponto, obtendo o vetor do ponto rotacio-

nado v. A linha 10 translada o ponto, somando o vetor do ponto a um vetor de deslocamento formado por `[K1,K2]`.

Algoritmo 5.3 - Rotação e translação de um ponto

```
01    --K1: deslocamento em x
02    --K2: deslocamento em y
03    --Q1: [raio, phi]
04
05    raio := Q1'x
06    phi := Q1'y
07    v := [raio,0] ?~def(v)
08    m := {[cos(phi),-sen(phi),0,0],[sen(phi),cos(phi),0,0]}
09    v := m >< v
10    v1 := v + [K1,K2]
11    mostra(v)
12
13    glPointSize(5)
14    glColor3f(1,0,0)
15    glBegin(GL_POINTS)
16    glVertex2fv(v1)
17    glEnd()
```