FUNDAÇÃO GETULIO VARGAS
ESCOLA DE MATEMÁTICA APLICADA

CRISTHIAN GRUNDMANN

GEODESIC TRACING: VISUALIZAÇÃO DE CURVAS E SUPERFÍCIES ATRAVÉS DE GEODÉSICAS

Rio de Janeiro
2022

CRISTHIAN GRUNDMANN

GEODESIC TRACING: VISUALIZAÇÃO DE CURVAS E SUPERFÍCIES ATRAVÉS DE GEODÉSICAS

Trabalho de conclusão de curso apresentada para a Escola de Matemática Aplicada (FGV/EMAp) como requisito para o grau de bacharel em Matemática Aplicada.

Área de estudo: curvas e superfícies.

Orientador: Asla Medeiros e Sá

Rio de Janeiro

2022

Lista de códigos

1 Programas

O usuário se comunica com a interface através de um texto, chamado de programa, que contém os objetos de interesse. Por exemplo:

Código 1.1 – Exemplo de objetos

```
1  #circle and tangents
2  param r : [/2, 1];
3  param o : [0, 2pi];
4  curve c(t) = r(cost, sint, 0), t : [0, 2pi];
5  grid k : [0, 2pi, 8];
6  define k2 = k + o;
7  point p = ck2;
8  vector v = c'k2 @ p;
9
10 #function and surface
11 #function f(x, y) = x^2+y^2;
12 #surface s(u,v) = (u,v,f(u,v)), u : [-1, 1], v : [-1, 1];
```

O programa começa com uma linha de comentário, no estilo da linguagem Python. Os objetos `r` e `o` são parâmetros que podem ser alterados na interface. Seus valores devem estar nos intervalos indicados.

O objeto `c` é uma curva parametrizada por `t`. O domínio da parametrização é o intervalo indicado. A curva depende do parâmetro `r`, que foi definido anteriormente.

O objeto `k` é uma grade de 8 pontos igualmente espaçados no intervalo indicado. Uma grade é tratada como uma constante, assim como um parâmetro. Se um objeto desenhável depende de uma grade, uma instância é desenhada para cada valor da grade. Um objeto pode depender de mais de uma grade.

O objeto `k2` é uma constante, e não pode ser alterada na interface. Esse tipo de objeto pode ser usado para deixar o programa mais legível.

O objeto `p` é o ponto da curva `c` de parâmetro `t = k2`. Esse objeto depende indiretamente de `k`, então é instanciado 8 vezes.

O objeto `v` é o vetor tangente da curva `c` no ponto `p` e desenhado a partir do mesmo ponto. O vetor também depende indiretamente de `k`, então é desenhado 8 vezes.

Os objetos `f` e `s` estão comentados, então não são considerados. Estão presentes apenas para o exemplo ter todos os tipos de objeto.

Os objetos desenháveis são pontos, vetores, curvas e superfícies. Pontos e vetores podem ser usados como valores em outros objetos. Pontos representam suas posições e

vetores seus deslocamentos. Curvas e superfícies podem ser usadas como funções, sem a restrição no domínio.

Há duas constantes pré-definidas: `pi` e `e`; e diversas funções pré-definidas: `sin`, `cos`, `tan`, `exp`, `log`, `sqrt` e `id`. A função `id` é a identidade é útil apenas no funcionamento interno do sistema.

As operações metemáticas e suas precedências estão descritas no código 1.2.

Código 1.2 – Ordem das operações

```
1     0) (), [] e {}
2     1) , (construção de tupla)
3     2) + e - (associativas à esquerda)
4     3) multiplicação por justaposição (associativa à esquerda)
5     4) * e / (associativas à esquerda)
6     5) +, -, * e / (unárias)
7     6) aplicação de função
8     7) ^ (associativa à direita)
9     8) _ (elemento de tupla) (associativa à esquerda)
10    9) ' e _ (derivada total e parcial) (associativas à esquerda)
```

Parâmetros e grades podem ser multidimensionais: `param T : [0, 1], [0, 1];`. Assim, o objeto `T` é uma tupla, e seus elementos podem ser obtidos com `T_1` e `T_2`.

Há 4 operadores unários: `+` e `-` são os usuais. A operação `*x` representa `xx`, e `/x` é igual a `1/x`.

Para números reais, multiplicação com `*` e por justaposição são equivalentes. Porém, para tuplas, `a*b` representa o produto vetorial e `ab` representa o produto escalar. Assim, `*x` calcula o quadrado do módulo do vetor `x`.

1.1 Gramática formal

O programa deve seguir uma gramática formal, que especifica a sintaxe das declarações dos objetos e das expressões matemáticas. As expressões matemáticas podem seguir uma notação mais natural que as de várias linguagens de programação. Por exemplo, há multiplicação por justaposição: `3x = 3*x`; e a aplicação de funções não exige parênteses: `sin-x = sin(-x)`.

A gramática livre de contexto é definida pelo código 1.3 (AHO, 1986). A sintaxe das expressões matemáticas foi baseada na gramática da linguagem C (UNIVERSITY, s.d.).

Código 1.3 – Gramática livre de contexto

```
1  PROG    = DECL PROG | ;
2
3  DECL    = "param"     id ":" INTS ";" ;
4  DECL    = "grid"      id ":" GRIDS ";" ;
5  DECL    = "define"    id "=" EXPR ";" ;
6  DECL    = "curve"     FDECL "," TINTS ";" ;
7  DECL    = "surface"   FDECL "," TINTS ";" ;
8  DECL    = "function"  FDECL ";" ;
9  DECL    = "point"     id "=" EXPR ";" ;
10 DECL    = "vector"    id "=" EXPR "@" EXPR ";" ;
11
12 FDECL   = id "(" IDS ")" "=" EXPR ;
13 IDS     = IDS "," id | id ;
14 INT     = "[" EXPR "," EXPR "]" ;
15 GRID    = "[" EXPR "," EXPR "," EXPR "]" ;
16 TINT    = id ":" INT ;
17 INTS    = INTS "," INT | INT ;
18 TINTS   = TINTS "," TINT | TINT ;
19 GRIDS   = GRIDS "," GRID | GRID ;
20
21 EXPR    = ADD ;
22 ADD     = ADD "+" JUX | ADD "-" JUX | JUX ;
23 JUX     = JUX MULT2 | MULT ;
24 MULT    = MULT "*" UNARY | MULT "/" UNARY | UNARY ;
25 MULT2   = MULT2 "*" UNARY | MULT2 "/" UNARY | APP ;
26 UNARY   = "+" UNARY ;
27 UNARY   = "-" UNARY ;
28 UNARY   = "*" UNARY ;
29 UNARY   = "/" UNARY ;
30 UNARY   = APP ;
31 APP     = FUNC UNARY | POW ;
32 FUNC    = FUNC2 "^" UNARY | FUNC2 ;
33 FUNC2   = FUNC2 "_" var | FUNC2 "'" | func ;
34
35 POW     = COMP "^" UNARY | COMP ;
36 COMP    = COMP "_" num | FACT ;
37 FACT    = const | num | var | "(" TUPLE ")" | "[" TUPLE "]" | "{" TUPLE
       "}" ;
38 TUPLE   = ADD "," TUPLE | ADD ;
```

Os termos em maiúsculo(não-terminais) representam variáveis gramaticais. O lado direito de uma igualdade especifica as possíveis formas sentencias que um não-terminal pode assumir, separadas por uma barra vertical ou em diferentes equações. Por exemplo, `MULT` possui 3 formas: `MULT * UNARY`, `MULT / UNARY` e `UNARY`. Cada forma tem um significado diferente. Uma forma pode ser vazia, como ocorre para `PROG`.

Os símbolos entre aspas representam textos literais, e os termos em minúsculo(terminais) representam uma classe de "palavras": Por exemplo, `num` representa um número e `var` um nome de uma variável.

O termo `PROG` representa um programa completo, que é uma sequência de declarações(`DECL`). O termo `EXPR` representa uma expressão matemática. Os símbolos abaixo de `EXPR` definem a sintaxe das operações, suas ordens de precedência e associatividades.

Um programa coeso é formado a partir de `PROG`. Enquando houver não-terminais, deve-se substituí-los por uma de suas formas.

Para extraír o significado de um programa, o processo contrário deve ser feito. É necessário encontrar uma maneira de se obter o programa a partir de `PROG`. Para um programa coeso, sempre há uma maneira e essa é única. Algumas transformações nessa gramática a torna LL1, uma propriedade que garante que o *parsing* pode ser feito de forma fácil e rápida. Além disso, LL1 garante que a gramática não é ambígua. (CALGARY, s.d.)

Referências

AHO, Alfred V. **Compilers: Principles, Techniques, & Tools**. [S.l.]: Pearson, 1986. Syntax Analysis.

CALGARY, University of. **The Context Free Grammar Checker**. [S.l.: s.n.]. http://smlweb.cpsc.ucalgary.ca/. Acessado em 2022-09-28.

UNIVERSITY, Western Michigan. **The syntax of C in Backus-Naur Form**. [S.l.: s.n.]. https://cs.wmich.edu/~gupta/teaching/cs4850/sumII06/The%20syntax%20of%20C%20in%20Backus-Naur%20form.htm. Acessado em 2022-09-28.