

# Visualização de curvas e superfícies a partir de geodésicas

Cristhian Grundmann

31 de Março de 2022

## Conteúdo

## Listagens

## Lista de Figuras

## Lista de Tabelas

## 1 Gramática de descrição de objetos

A gramática permite a declaração de 8 tipos de objeto: parâmetro, curva, superfície, definição, função, grade, ponto e vetor.

Os objetos podem fazer referência apenas a objetos definidos anteriormente, porém apenas pontos e vetores podem se referir a grades, diretamente ou indiretamente.

A gramática descreve duas estruturas principais: as declarações dos objetos(DECL) e as expressões matemáticas(ADD). As declarações têm estrutura simples, já as expressões são mais complexas.

A gramática das expressões matemáticas é mais especial quando comparada às linguagens de programação gerais, pois o domínio desse projeto é muito mais limitado. As partes não usuais são:

- Multiplicação justaposta:  $2 \ x \ y \ z$
- Função sem parênteses: `sin -x`
- Recíproco unário:  $/x = 1/x$
- Multiplicação unária(sem efeito):  $*x = x$
- Potenciação e derivação em funções:  $f^2 \ x$ ,  $\sin' \ x$

1	PROG	-> DECL PROG   DECL
2		
3	DECL	-> param id : INT ;
4	DECL	-> curve id = ADD, t : INT ;
5	DECL	-> surface id = ADD, u : INT, v : INT ;
6	DECL	-> define id = ADD ;
7	DECL	-> function id ( ARGS ) = ADD ;
8	DECL	-> grid id : INT , ADD ;
9	DECL	-> point id = ADD ;
10	DECL	-> vector id = ADD @ ADD ;
11		
12	ARGS	-> FTAG id , ARGS   id
13		
14	FTAG	-> *   eps
15		
16	INT	-> [ ADD , ADD ]
17		
18	ADD	-> ADD + MULT
19	ADD	-> ADD - MULT
20	ADD	-> MULT
21		
22	MULT	-> MULT * UNARY
23	MULT	-> MULT APP
24	MULT	-> MULT / UNARY
25	MULT	-> UNARY
26		
27	UNARY	-> + UNARY
28	UNARY	-> - UNARY
29	UNARY	-> * UNARY
30	UNARY	-> / UNARY
31	UNARY	-> APP
32		
33	APP	-> FUNC UNARY   EXP
34		
35	FUNC	-> FUNC ^ UNARY
36	FUNC	-> FUNC _ var_id
37	FUNC	-> FUNC ' ,
38	FUNC	-> func_id
39		
40	EXP	-> COMP ^ UNARY   COMP
41		
42	COMP	-> FACT . number   FACT
43		
44	FACT	-> const
45	FACT	-> number
46	FACT	-> var_id
47	FACT	-> TUPLE
48		
49	TUPLE	-> ( LIST )
50	LIST	-> ADD , LIST   ADD

Listagem 1: Gramática completa

## 2 Processamento da descrição

Há um pequeno detalhe a ser notado antes de se fazer o *parsing* da descrição: não é possível determinar se um lexema como **k** forma um token do tipo **var\_id**, **func\_id** ou **id**. Isso é um problema pois a expressão **k 2** pode representar uma multiplicação ou uma aplicação de função, dependendo do tipo de **k**. A declaração do objeto determina seu tipo, então a análise léxica (que lê o texto e produz tokens adequados) e a análise sintática (que lê a sequência de tokens e produz uma estrutura de dados) deverão trabalhar em sincronia. O lexer e o parser devem compartilhar uma *tabela de símbolos*, indicando quais símbolos (identificadores) estão definidos e quais são seus tipos. Quando o parser termina de processar a declaração de **k**, ele o põe na tabela de símbolos com o tipo adequado. Durante a leitura da declaração, o lexer lia **k** como um token do tipo **id**, que serve para indicar um símbolo novo. Depois da declaração, o lexer passará a ler **k** como **var\_id** ou **func\_id**, dependendo da declaração.

Para processar uma declaração de função **f**, a lista de argumentos é uma sequência de tokens do tipo **id**. Ao ler um argumento, o parser o coloca na tabela de símbolos como **var\_id**, impedindo a repetição de argumentos e possibilitando a referência no corpo da função. Após a declaração, os símbolos que serviram como argumento são removidos da tabela, pois são dummies.

O método de parsing usado é o *recursive descent*, onde cada não-terminal é uma rotina que lê uma sentença de seu tipo. Essas rotinas possuem efeitos colaterais, por isso não são chamadas de funções. Por exemplo, a rotina **FACT** examina o token atual, e consegue determinar qual produção gramatical deve ser utilizada. Caso seja **TUPLE**, sua rotina é chamada para ler uma tupla. Caso seja outra produção, a rotina consome e processa o token, mandando o lexer passar para o próximo.