

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

2.1	Exemplo de objetos	4
2.2	Gramática livre de contexto	6

1 Introdução

Desenhos de superfícies costumam ser feitos a partir de um ponto de vista do espaço ambiente 3D. Esse projeto implementa uma visualização de superfícies que não depende de um espaço ambiente.

A visualização pode ser comparada ao que um ser bidimensional interno à superfície observaria: simula-se raios de luz partindo da posição do ser, e os pontos iluminados são observados. Os raios de luz devem seguir caminhos em ‘linha reta’, que minimizam distância. Para uma superfície qualquer, esses caminhos são chamados de geodésicos, e são estudados na geometria diferencial. A visualização, chamada de *geodesic tracing*, obtém uma transformação de uma imagem original sobre a superfície.

A implementação feita nesse projeto é feita em três partes: compilador, método numérico e interface gráfica.

O compilador fornece uma maneira do usuário definir as superfícies e outros objetos. O usuário escreve um texto, seguindo algumas regras gramaticais, que então é processado. A teoria de compiladores é essencial para essa etapa, principalmente a análise léxica e a análise sintática ([AHO, 1986](#)). O compilador está descrito no capítulo 2.

O método numérico se refere à simulação dos raios de luz na superfície. Um raio de luz é determinado pela posição e direção inicial, que são as condições iniciais. Um sistema de equações diferenciais ordinárias (equação geodésica ([PRESSLEY, 2012](#))) determina a curva que a luz traça. Uma solução aproximada da equação é calculada pelo método de Runge-Kutta de ordem 4 ([BURDEN, 2001](#)). O método está descrito no capítulo 3.

A interface gráfica é simples e é construída usando *ImGui* ([CORNUT, s.d.](#)), uma ferramenta de interface gráfica fácil de usar. A linguagem de programação escolhida para a implementação desse projeto é *C++*, e para desenhar a interface e os objetos, *OpenGL* é usado.

2 Compilador

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

Código 2.1 – 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];

```

A linguagem permite comentários no estilo da linguagem Python, usando #.

O programa declara os seguintes objetos:

Objetos	Descrição
r e o	parâmetros que podem ser alterados na interface. Seus valores devem estar nos intervalos indicados
c	uma curva parametrizada por t . O domínio da parametrização é o intervalo indicado. A curva depende do parâmetro r , que foi definido anteriormente.
k	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.
k2	uma constante, e não pode ser alterada na interface como os parâmetros. Esse tipo de objeto pode ser usado para deixar o programa mais legível.
p	o ponto da curva c de parâmetro t = k2 . Esse objeto depende indiretamente de k , então é instanciado 8 vezes.
v	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.

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`.

2.1 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 2.2 (AHO, 1986). A sintaxe das expressões matemáticas foi baseada na gramática da linguagem C (UNIVERSITY, s.d.).

Código 2.2 – 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 | id ":" GRID ;
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 | "-" UNARY | "*" UNARY | "/" UNARY | APP ;
27 APP        = FUNC UNARY | POW ;
28 FUNC       = FUNC2 "^" UNARY | FUNC2 ;
29 FUNC2      = FUNC2 "_" var | FUNC2 "'" | func ;
30
31 POW        = COMP "^" UNARY | COMP ;
32 COMP       = COMP "_" num | FACT ;
33 FACT       = const | num | var
34             | "(" TUPLE ")" | "[" TUPLE "]" | "{" TUPLE "}";
35 TUPLE      = ADD "," TUPLE | ADD ;

```

Os termos em maiúsculo(não-terminais) representam variáveis gramaticais. O lado direito de uma igualdade especifica as possíveis formas sentenciais 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 **PRÓG** 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 **PRÓG**. Enquanto houver não-terminais, deve-se substituí-los por uma de suas formas.

Para extrair o significado de um programa, o processo contrário deve ser feito. É necessário encontrar uma maneira de se obter o programa a partir de **PRÓG**. 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.](#))

A tabela 1 descreve as operações e suas ordens de precedência, com base na gramática.

Tabela 1 – Ordem das operações

Operações	Aridade	Associatividade	Exemplo	Descrição
() [] {}	Unário		(<i>expr</i>)	Isola a expressão interna
,	Binário	Esquerda	(<i>a</i> , <i>b</i> , <i>c</i>)	Adiciona uma elemento à tupla(dentro de parênteses)
+ -	Binário	Esquerda	<i>a</i> + <i>b</i>	Soma e subtração usuais
<i>juxtaposição</i>	Binário	Esquerda	<i>ab</i>	Multiplicação
* /	Binário	Esquerda	<i>a</i> * <i>b</i>	Multiplicação e Divisão
+ - * /	Unário		- <i>x</i> , * <i>v</i>	Positivo, Negativo, Quadrado e Recíproco
<i>aplicação</i>	Binário	Esquerda	sin <i>x</i>	Soma e subtração usuais
^	Binário	Direita	<i>a</i> ^ <i>b</i>	Potenciação
_	Unário		(1, 2, 3)_2	Elemento da tupla
’, _	Unário		sin’ <i>x</i> + f_ <i>z</i> (3)	Derivada Total e Parcial

3 Curvas e Superfícies

O objetivo primário do projeto é visualizar curvas e superfícies. As curvas são visualizadas apenas em espaço 3D. As superfícies são visualizadas em espaço 3D e em *geodesic tracing*.

3.1 Curvas

Há duas principais maneiras de se definir uma curva na geometria analítica: por parametrização e por equação. Esse trabalho apenas considera curvas paramétricas.

Uma curva pode ser parametrizada por um número real. Formalmente, uma parametrização é uma função $\gamma : I \rightarrow \mathbb{R}^n$, onde I é um intervalo real. Nesse trabalho, o intervalo é fechado, e $n = 3$.

As curvas são desenhadas através de vários segmentos. Dada uma partição de I de k pontos, pode-se aproximar a curva pelos segmentos de extremidade $\gamma(t_i)$ e $\gamma(t_{i-1})$ para $i < k$, onde t_i é o i -ésimo ponto da partição. Nesse trabalho, a partição depende apenas de k e é uniforme.

O vetor tangente pode ser calculado com $\gamma'(t)$.

3.2 Superfícies

Assim como as curvas, apenas superfícies parametrizadas serão consideradas nesse trabalho: $\sigma : I_1 \times I_2 \rightarrow \mathbb{R}^3$, onde I_1 e I_2 são intervalos fechados reais.

As superfícies são desenhadas através de vários triângulos, a partir de partições dos intervalos I_1 e I_2 . Juntas, as partições formam uma grade de retângulos, e cada retângulo pode ser dividido em 2 triângulos. Esses são os triângulos desenhados.

Os vetores tangentes nas direções coordenadas são as derivadas parciais $\sigma_u(u, v)$ e $\sigma_v(u, v)$, onde os parâmetros são u e v . Nesse projeto, os parâmetros podem ter nomes quaisquer.

3.2.1 Primeira forma fundamental

Supondo que a superfície seja diferenciável e com vetores tangentes linearmente independentes, a primeira forma fundamental no ponto paramétrico (u, v) é definida como

$$\begin{bmatrix} \sigma_u \cdot \sigma_u & \sigma_u \cdot \sigma_v \\ \sigma_v \cdot \sigma_u & \sigma_v \cdot \sigma_v \end{bmatrix} = \begin{bmatrix} E & F \\ F & G \end{bmatrix}$$

onde as funções são todas aplicadas no ponto (u, v) .

Os vetores σ_u e σ_v formam uma base do espaço tangente. O produto escalar de dois vetores tangentes $x = x_1\sigma_u + x_2\sigma_v$ e $y = y_1\sigma_u + y_2\sigma_v$ pode ser calculado da seguinte forma:

$$\begin{aligned} x \cdot y &= (x_1\sigma_u + x_2\sigma_v) \cdot (y_1\sigma_u + y_2\sigma_v) \\ x \cdot y &= x_1y_1E + x_1y_2F + x_2y_1F + x_2y_2G \end{aligned}$$

O produto depende apenas dos coeficientes e da primeira forma fundamental. Isso significa que distâncias e ângulos podem ser calculados sem se referir ao espaço ambiente da parametrização, ou seja, de forma intrínseca.

3.2.2 Equação geodésica

3.2.3 Solução Numérica

3.2.4 Geodesic Tracing

Referências

- AHO, Alfred V. **Compilers: Principles, Techniques, & Tools**. [S.l.]: Pearson, 1986. Syntax Analysis.
- BURDEN, Richard L. **Numerical Analysis**. [S.l.]: Cengage, 2001. Initial-Value Problems for Ordinary Differential Equations.
- CALGARY, University of. **The Context Free Grammar Checker**. [S.l.: s.n.]. <http://smlweb.cpsc.ucalgary.ca/>. Acessado em 2022-09-28.
- CORNUT, Omar. **ImGui**. [S.l.: s.n.]. <https://github.com/ocornut/imgui>. Acessado em 2022-10-04.
- PRESSLEY, Andrew. **Elementary Differential Geometry**. [S.l.]: Springer, 2012. Geodesics.
- 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.