

UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

# Trabalho Prático

## Computação Gráfica

Catarina Machado (a81047) Gonçalo Faria (a86264)  
João Vilaça (a82339)

4 de Março de 2019

# Conteúdo

<b>1 Fase 1 - Primitivas Gráficas</b>	<b>2</b>
1.1 Gerador . . . . .	2
1.1.1 Cone . . . . .	3
1.1.2 Esfera . . . . .	4
1.1.3 Plano . . . . .	5
1.1.4 Caixa . . . . .	5
1.2 Engine . . . . .	9
<b>2 Extras</b>	<b>10</b>

# 1 Fase 1 - Primitivas Gráficas

O objectivo da primeira fase do trabalho prático foi criar um gerador de vértices de quatro diferentes primitivas gráficas: plano, caixa, esfera e cone, tendo em consideração diferentes parâmetros, tais como a altura, a largura, a profundidade e o número de divisões. Também foi necessário criar uma aplicação que lê o ficheiro XML (gerado pelo gerador) e exibe a respetiva primitiva gráfica.

## 1.1 Gerador

Para simplificar a construção das diferentes formas, foi criado um módulo *CoordinateFrame* que representa um dado sistema de coordenadas. Este módulo mantém como estado, numa matriz, a informação referente à base e à origem, do sistema de coordenadas em questão, assim como uma lista onde se encontram registados sequencialmente os pontos que foram desenhados relativos a um sistema de coordenadas inicial.

Este módulo, possibilita a criação de funções simples e eficientes evitando a necessidade de calcular analiticamente as coordenadas de todos os pontos, que compreendem as formas geométricas desenhadas. A geração do modelo que comprehende as figuras geométricas especificada é também implementada neste módulo, através da biblioteca *tinyxml2*, e corresponde, em essência, apenas à escrita de todos os pontos constantes na dada instância de *CoordinateFrame*.

As funções implementadas comprehendem as operações de rotação, translação, escala e também três outras funções genéricas que conjuntamente permitem construir um diversificado conjunto de figuras geométricas. Estas funções são denominadas *frameTriangle*, *frameRegularPolygon*, *frameStacker*.

A função *frameTriangle* permite a construção de um triângulo isósceles. À custa desta é implementada a função *frameRegularPolygon* que permite a construção de um polígono regular com um arbitrário número de lados. A função *frameStacker*, através da função de construção de polígonos regulares, permite construir figuras geométricas por meio da amontoação de prismas, com bases paralelas semelhantes e com os lados formados por retângulos. Esta função é parametrizada com o número de lados das bases, o número de camadas de prismas e por uma função que descreve a distância dos vértices do polígono regular à origem em função da distância euclidiana na direção do vetor normal do primeiro polígono desenhado. Para efeitos de optimização, as bases interiores, as que não são visíveis no modelo final, não são desenhadas.

### 1.1.1 Cone

O cone é implementado, essencialmente, por via do uso da função *frameStacker* do módulo *CoordinateFrame*. Esta suporta já a indicação das *stacks* e das *slices*, sendo no entanto introduzida a função defina pela equação em (1) como parâmetro. Para que à medida que se afasta do primeiro polígono base a distância dos vértices dos subsequentes polígonos à origem decresça linearmente, partindo de 1 quando a altura é 0 e termine em 0 quando a altura é 1, a função em (1) foi usada. Para que o cone fique com as dimensões especificadas antes do desenho é aplicada uma escala.

$$r(x) = 1 - x \quad (1)$$

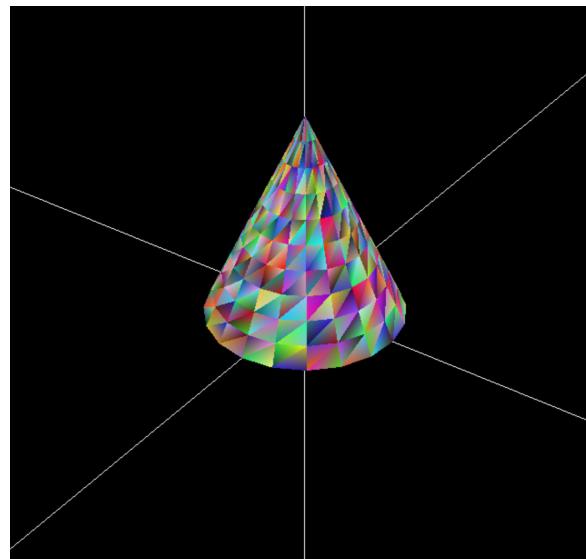


Figura 1: Cone com raio 1, altura 2, 20 slices e 10 stacks.

### 1.1.2 Esfera

A esfera, tal como o cone, é construída à custo da função *frameStacker* do módulo *CoordinateFrame*, no entanto a função que especifica o raio dos polígonos em função da altura é a indicada em (2).

$$r(x) = \sqrt{1 - (1 - x)^2} \quad (2)$$

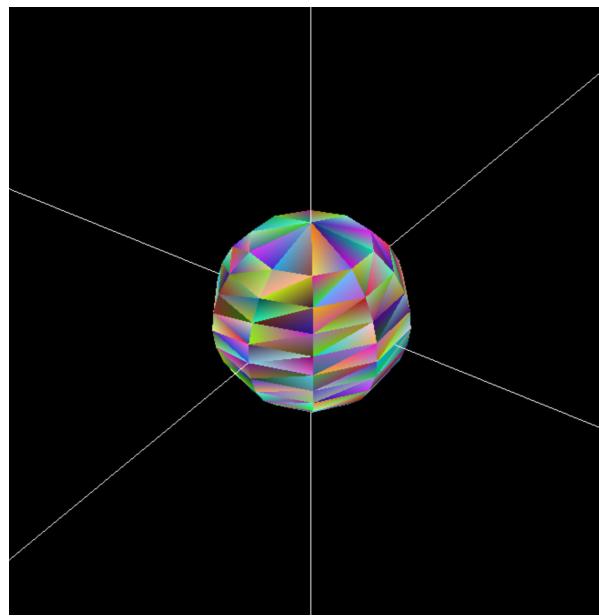


Figura 2: Esfera com raio 1, 10 slices e 10 stacks.

### 1.1.3 Plano

O plano é composto por dois triângulos, sendo os 3 pontos de cada um deles construídos pela função *framePoint*.

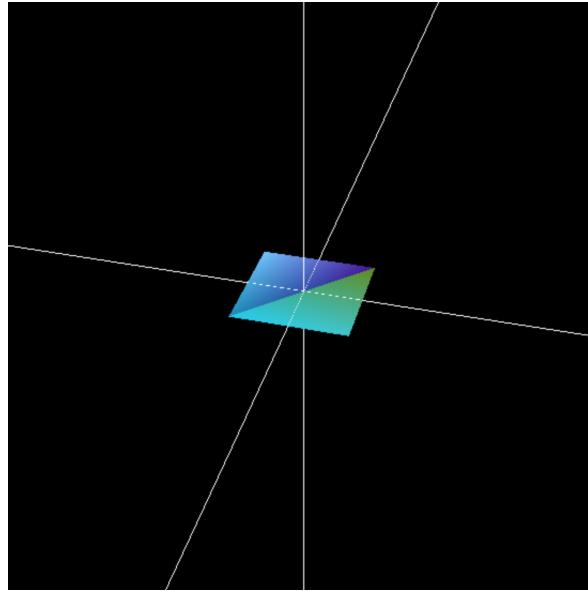


Figura 3: Plano XZ, centrado na origem.

### 1.1.4 Caixa

Para construção de cada face da caixa foi utilizada a função do plano anteriormente construída. Para facilitar os cálculos e de modo a termos as funções mais gerais possíveis, este plano foi construído com as dimensões 1x1, e todas as translações e escalas que iremos mencionar em seguida foram idealizadas tendo em conta essas medidas. Assim, para que a caixa tenha as dimensões finais pretendidas, basta fazer uma escala ao referencial antes de efetivamente desenhar a caixa.

**1.** O primeiro objetivo ao construir a caixa foi desenhar uma face com as divisões passadas como parâmetro (Figura 6). Para tal, uma face passa então a ser composta por número de divisões\*número de divisões de planos. A função responsável por isso é a *frameHyperplane*.

Assim, nesta função, numa fase inicial é feita uma escala com base no número de divisões pedidas (para que os “mini plano” fiquem com as dimensões pretendidas) e é feita uma translação de modo a que, no resultado final, a caixa se encontre centrada no centro do referencial (ponto (0,0,0)).

Consequentemente, o algoritmo passa por desenhar efetivamente um “mini plano” (já com a devida escala) e é feita uma translação para a direita. Isto é feito de forma iterativa até ficar construída uma linha com as divisões pedidas, tal como se pode ver na Figura 4.

Para se desenhar a face completa, foi necessário, também de forma iterativa, fazer uma translação no eixo do z e no eixo do x, de modo a que a próxima linha possa ser construída imediatamente abaixo da primeira, tal como ilustrado na Figura 5. O processo é repetido (consoante o número de divisões), até se obter o resultado pretendido (Figura 6).

**2.** Após termos uma face construída, o objetivo foi, através de translações e rotações à face anteriormente mencionada, construir a caixa completa, tal como se pode ver na Figura 7. A função responsável tem o nome de *frameCube*.

**3.** A função *box* é a função responsável pela construção efetiva da caixa, visto que é ela que chama a função *frameCube*. Para que a caixa tenha as dimensões (altura, largura e profundidade) passadas como parâmetro, antes de chamar a função *frameCube*, a função *box* aplica a devida escala ao referencial.

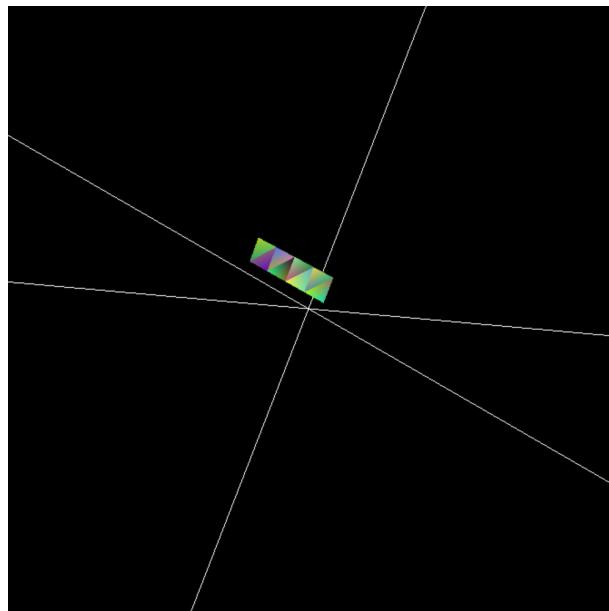


Figura 4: Linha com 4 divisões.

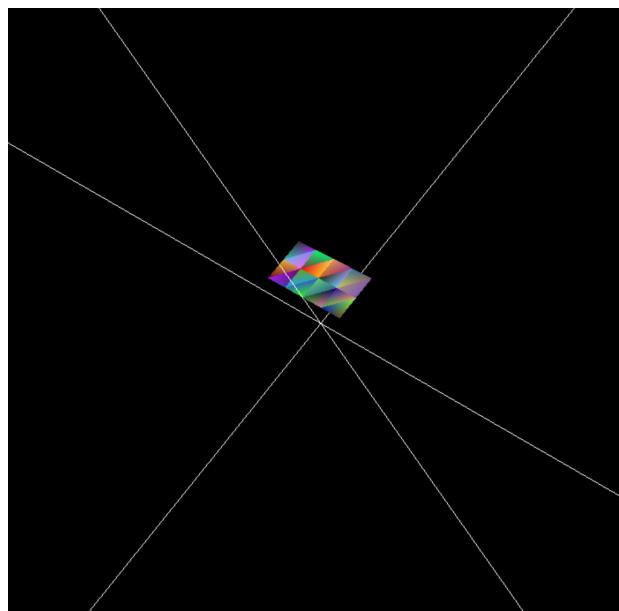


Figura 5: 2 linhas com 4 divisões.

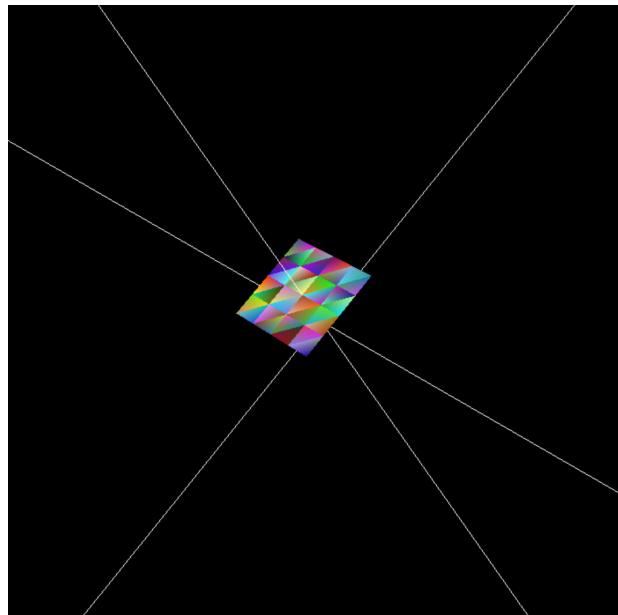


Figura 6: Face com 4 divisões.

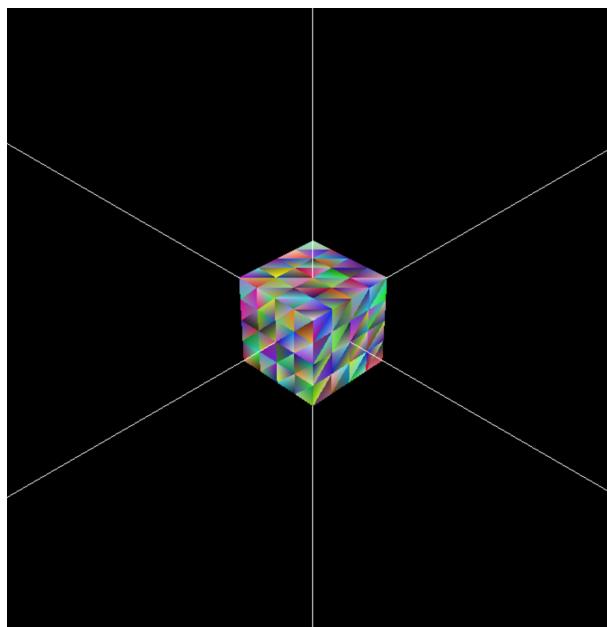


Figura 7: Caixa com dimensão 1x1x1, centrada na origem e com 4 divisões.

**Outros exemplos de caixas (Figuras 8 e 9):**

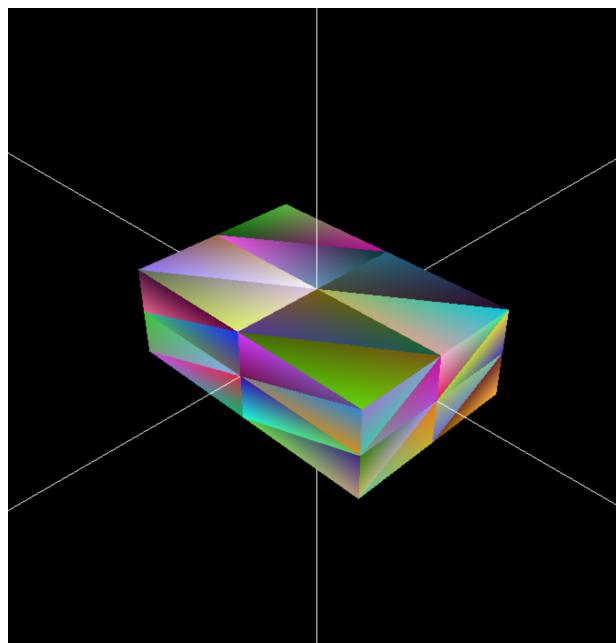


Figura 8: Caixa com dimensão  $3 \times 2 \times 1$ , centrada na origem e com 2 divisões.

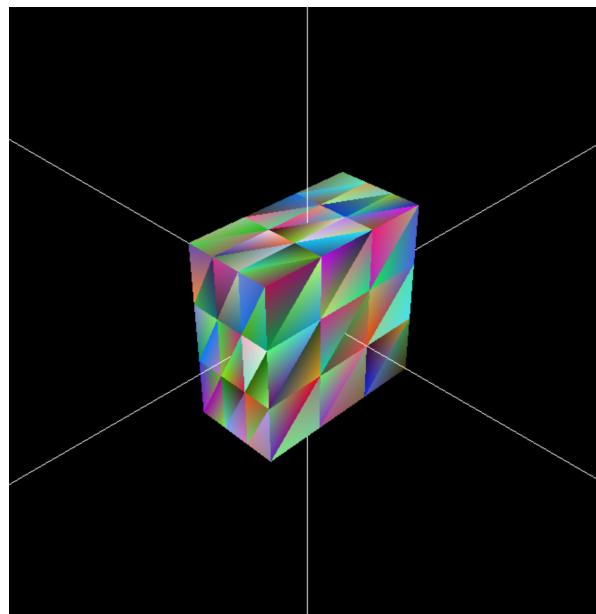


Figura 9: Caixa com dimensão  $1 \times 2 \times 2$ , centrada na origem e com 3 divisões.

## 1.2 Engine

O Engine é a aplicação responsável por receber o ficheiro de configuração de uma scene em XML com todos os ficheiros que contém os modelos a serem carregados para serem gerados através do OpenGL.

Cada ficheiro com modelos presentes no ficheiro de configuração é analisado pelo programa para dele serem extraídos os vários pontos que são depois adicionados a uma estrutura de dados. Esses pontos são depois sempre lidos dessa mesma estrutura pelo render do OpenGL.

```
<scene>
    <model file='/Users/user/cg/box3d' />
    <model file='/Users/user/cg/cone3d' />
</scene>
```

Figura 10: Exemplo de XML onde estão contidos os caminhos dos ficheiros com modelos.

```
<figure>
    <triangle>
        <point x="1" y="0" z="0"/>
        <point x="0" y="0" z="0"/>
        <point x="0.309" y="-1.164" z="-0.951"/>
    </triangle>
    <triangle>
        <point x="0.309" y="-1.164" z="-0.951"/>
        <point x="0" y="0" z="0"/>
        <point x="-0.809" y="-7.198" z="-0.587"/>
    </triangle>
    <triangle>
        <point x="-0.804" y="-7.197" z="-0.587"/>
        <point x="0" y="0" z="0"/>
        <point x="-0.806" y="7.197" z="0.587"/>
    </triangle>
    <triangle>
        <point x="-0.809" y="7.198" z="0.587"/>
        <point x="0" y="0" z="0"/>
        <point x="0.309" y="1.164" z="0.951"/>
    </triangle>
</figure>
```

Figura 11: Exemplo de XML com os triângulos que compõe o modelo e respetivos pontos.

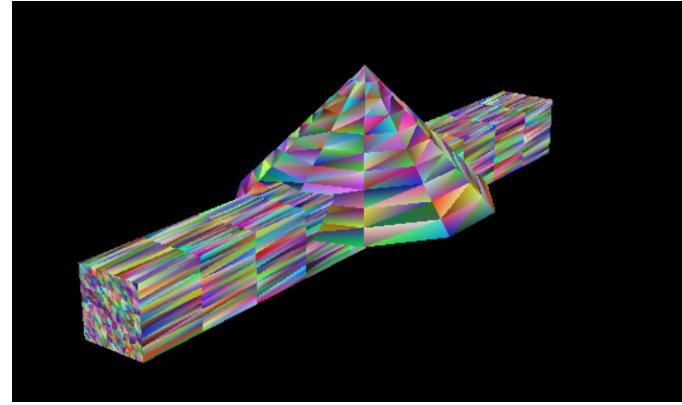


Figura 12: Resultado da aplicação do engine sobre o ficheiro da figura 10.

## 2 Extras

Em alternativa a modelar as diferentes formas geométricas através de funções específicas definidas, por exemplo, por coordenadas polares e esféricas nós concebemos funções genéricas que permitem o desenho de um enorme conjunto de formas. Para apresentar essas funcionalidades incluímos no relatório demonstrações do uso da função *frameStacker* com diferentes funções matemáticas como argumento. Nestes exemplos, o número de slices e de stacks é 10.

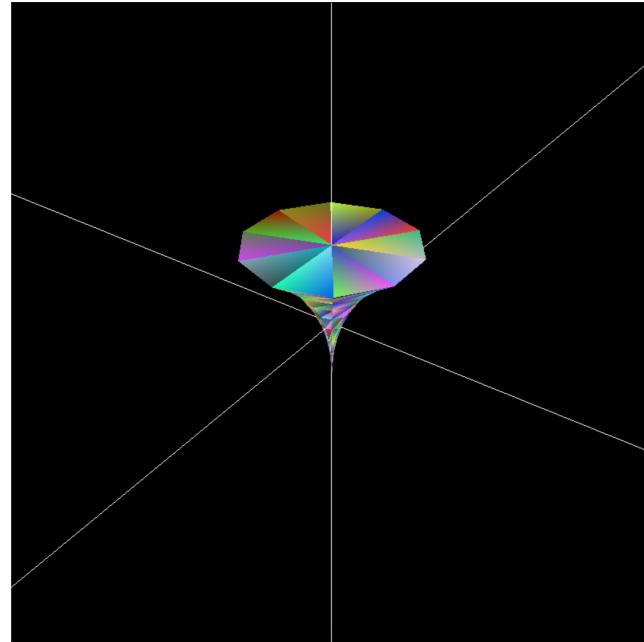


Figura 13:  $r(x) = x^3$

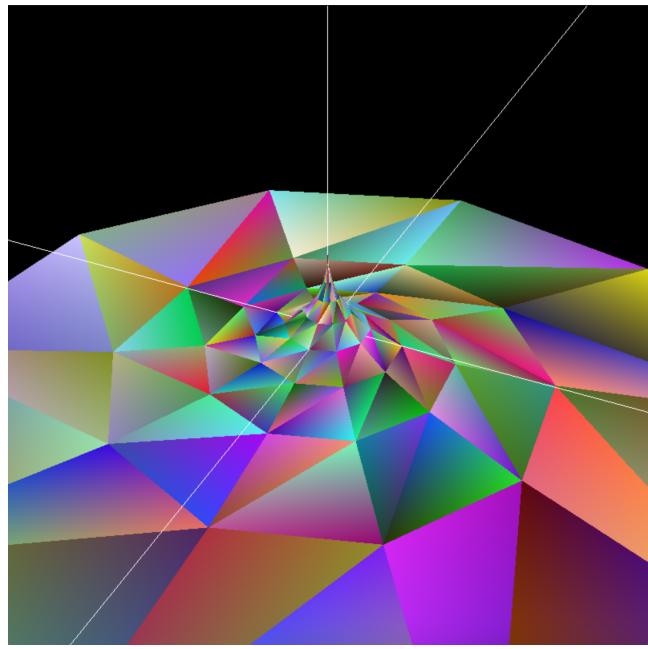


Figura 14:  $r(x) = \log(x)^2$

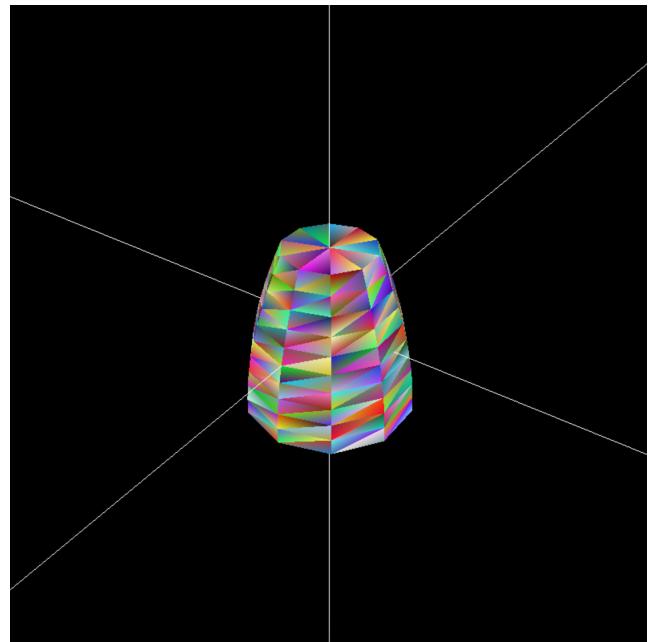


Figura 15:  $r(x) = \cos(x)$

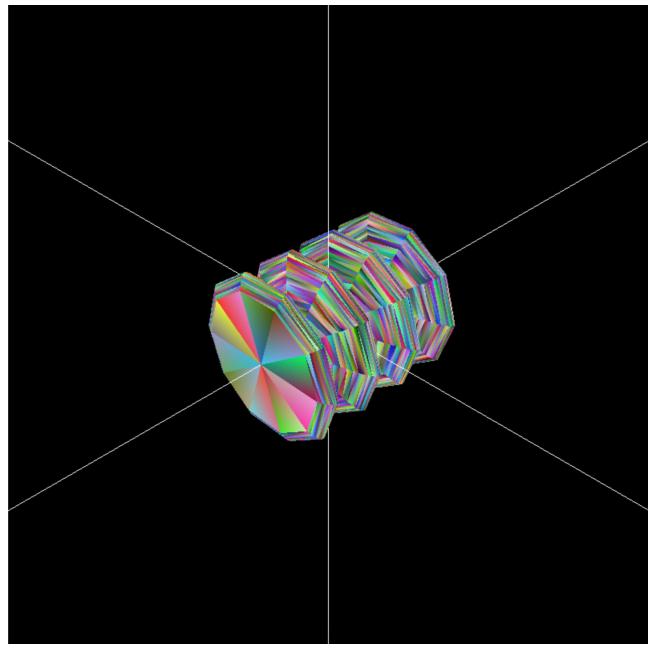


Figura 16:  $r(x) = \cos(10x)$ , com 100 stacks.

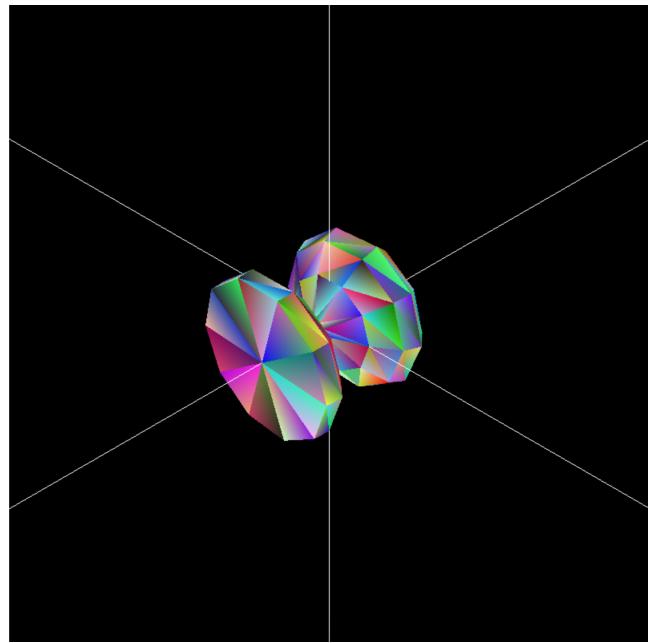


Figura 17:  $r(x) = \sin(5x)$

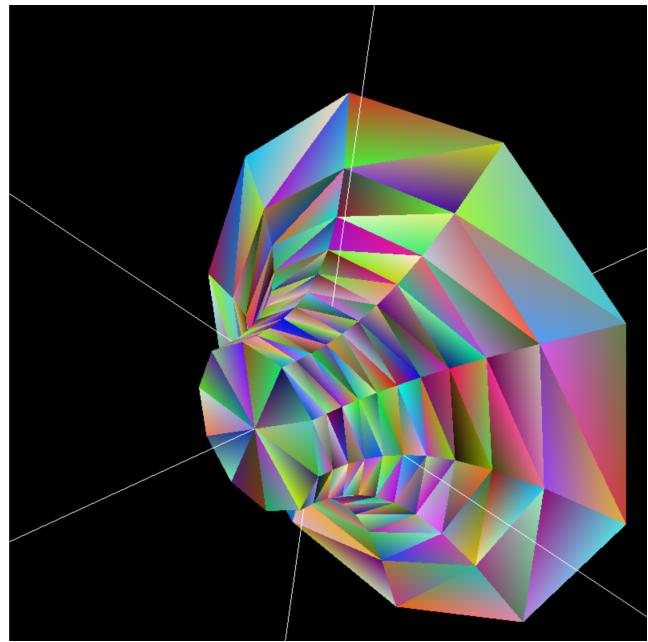


Figura 18:  $r(x) = 1 + \tan(x)^2$