



# GEOMETRIA COMPUTACIONAL ALGORITMO DE GRAHAM

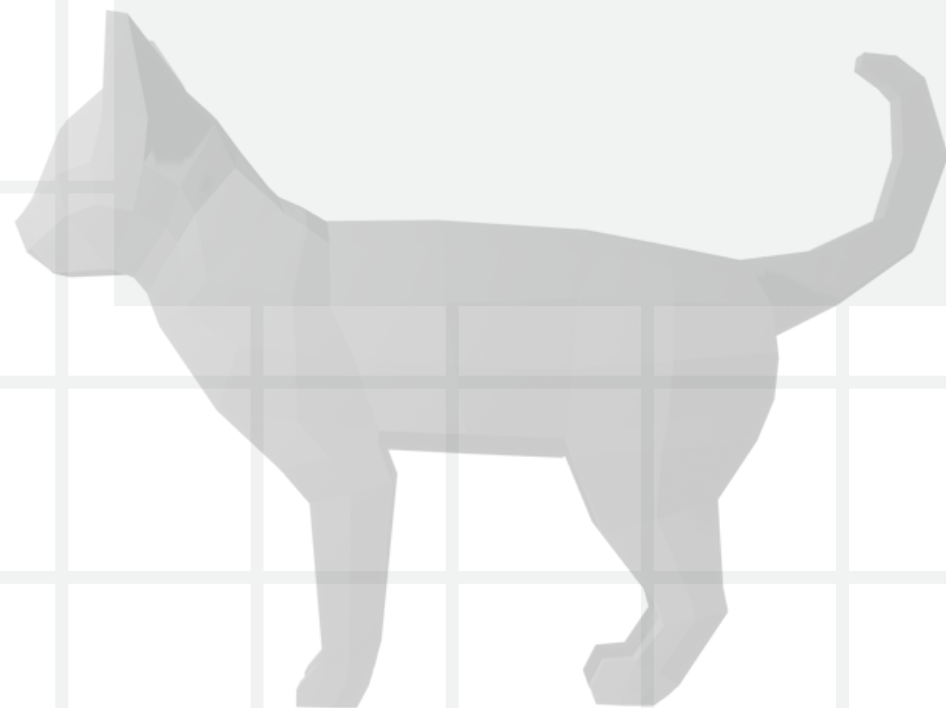
**Luis Fernando Bastos Rego - 470043**

**Samuel Vieira de Paula Farias - 498844**

# INTRODUÇÃO


O algoritmo de Graham é uma evolução do algoritmo de Jarvis, realizando o fecho convexo em tempo de ordem  $n \log n$ .

Nessa apresentação iremos mostrar como foi o processo da implementação e como foi feito o fecho convexo para o nosso tema.

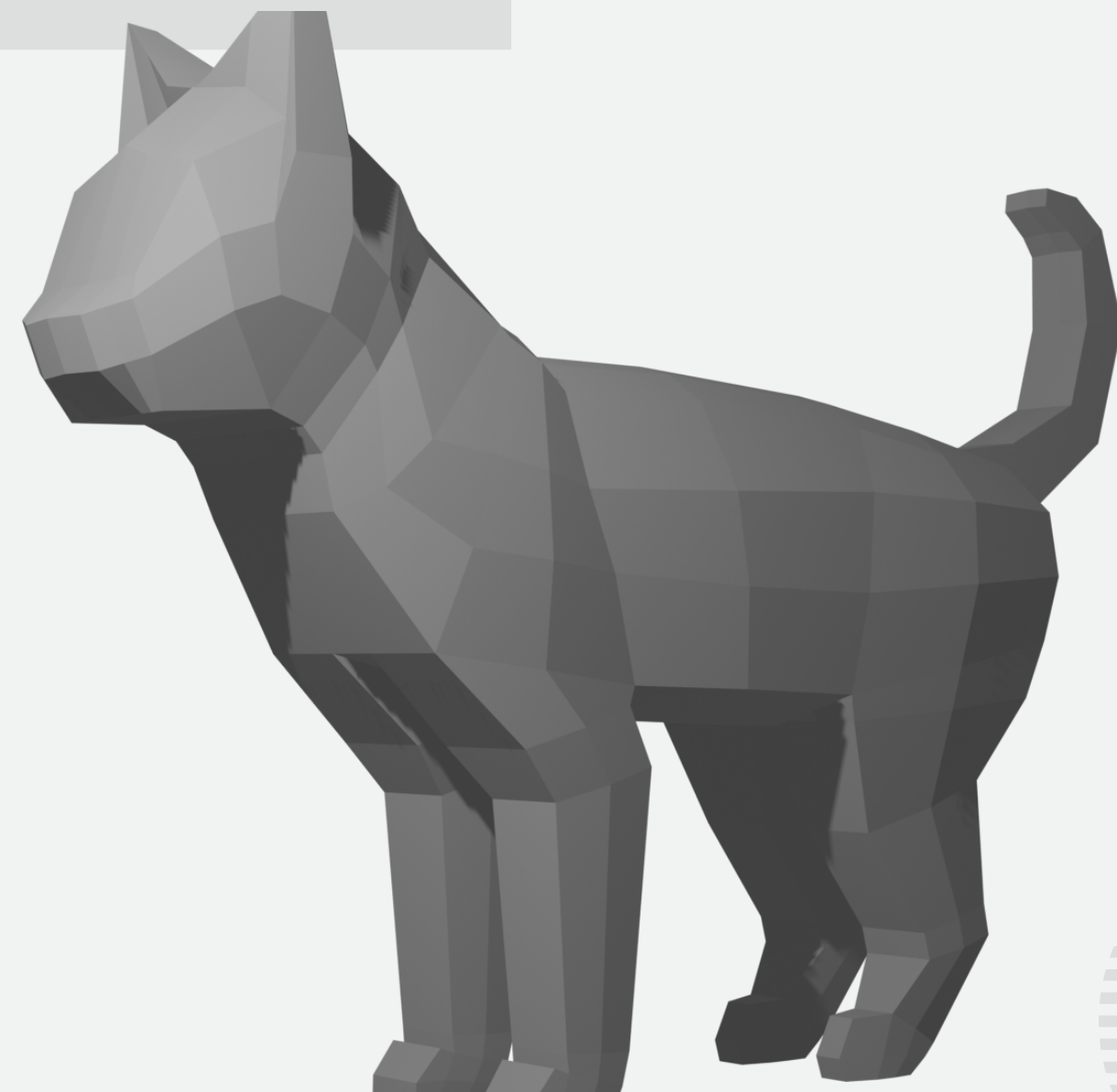
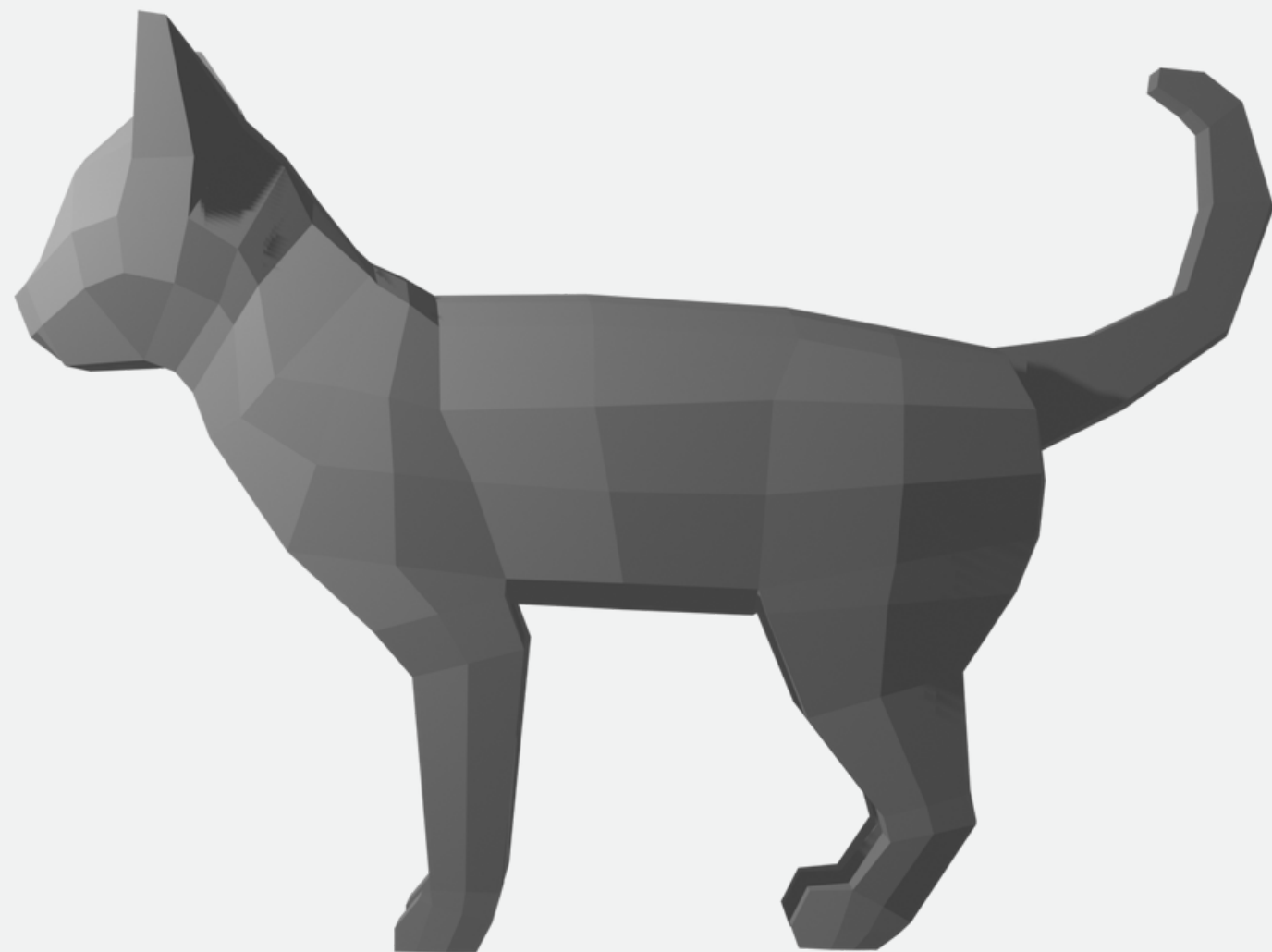




# **METODOLOGIA**

- Foram usados os pseudocódigos e explicações disponibilizados nos slides da disciplina;
  - Códigos escritos na linguagem C++;
  - Blender utilizado para renderização e modelagem;
  - Tema é uma projeção de um modelo 3D, disponibilizado na bibliografia deste slide;
- 

# TEMA



# PLANEJAMENTO

```
graph TD; A[PLANEJAMENTO] --- B[PARTE 1  
Divisão em partes convexas]; A --- C[PARTE 2  
Produção do fecho convexo com o algoritmo de Graham]; A --- D[PARTE 3  
União dos fechos produzidos];
```

## PARTE 1

Divisão em  
partes  
convexas

## PARTE 2

Produção do  
fecho convexo  
com o  
algoritmo de  
Graham

## PARTE 3

União dos  
fechos  
produzidos

# PRÉ-PROCESSAMENTO

## TRANSIÇÃO DO 3D PARA O 2D

Foi utilizado o Blender para realizar o "flattening" do modelo em torno do plano sagital, ou seja, a forma do gato em perfil

## REMOÇÃO E MODIFICAÇÃO DE VÉRTICES INDESEJÁVEIS

Para facilitar a divisão entre partes convexas, foram removidos ou modificados alguns vértices indesejáveis

# PSEUDOCÓDIGO

**GrahamHull(nuvemDePontos):**

centroide ← pegaCentroide(nuvemDePontos)

angulosRelativos ← Calcula os ângulos relativos à centroide

vetorPontos ← ordenação(angulosRelativos, nuvemDePontos)

deque ← double ended queue com a seguinte tupla:(ponto, contador),  
onde contador é o número de vezes que o ponto esteve na posição  
"next" e teve sua aresta com "current" validada , ou seja, foi  
verificado.

atual ← deque.inicio()

anterior ← deque.fim()

prox ← deque.segundoElemento()

# PSEUDOCÓDIGO

```
Enquanto (deque.front < 2 ou deque.back < 2):
```

```
    se verifica(anterior, atual, prox, deque):
```

```
        temp = deque.popFront()
```

```
        deque.pushBack(temp)
```

```
        deque.front.contador+=1
```

```
    senão:
```

```
        deck.popFront()
```

```
        temp = deque.popBack()
```

```
        deck.pushFront(temp)
```

```
retorna deque
```

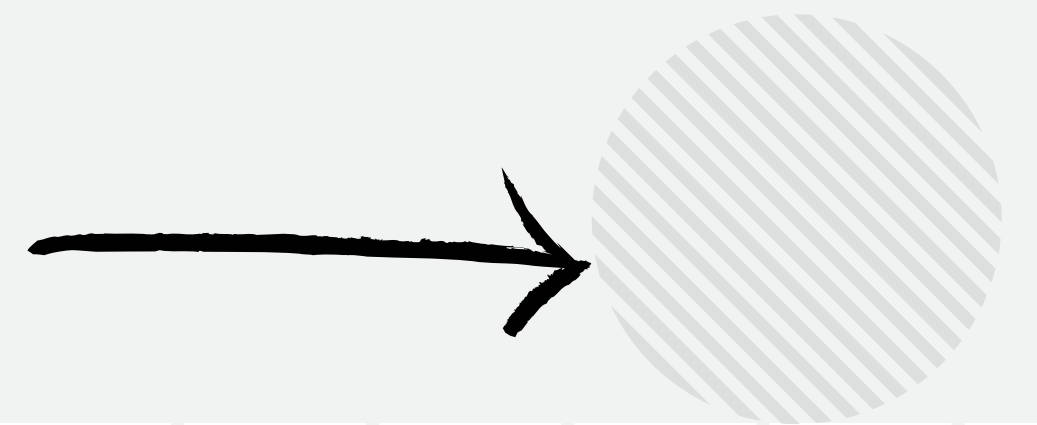


# PSEUDOCÓDIGO

```
verifica(anterior, atual, prox, v):  
  s ← prodvec((ponto[anterior], ponto[atual]), (ponto[atual], ponto[prox]))  
  se s ≥ 0:  
    retorna true  
  senão:  
    retorna false
```

# COLAR

O algoritmo correspondente ao "colar" consiste em criar um vetor de vértices únicos, o qual é preenchido com a leitura de vértices de cada objeto após a execução do algoritmo do fecho. Os vértices de cada objeto são referências ao índice desse vetor. Após isso, é construída uma matriz que segue a seguinte lógica para eliminar arestas internas.



# COLAR

- Se  $\text{matrix}[i][j] == F$ ,  $\text{matriz}[i][j]$  e  $\text{matriz}[j][i] = V$   
(A aresta ainda não existe, logo, adiciona)
- Se  $\text{matrix}[i][j] == V$ ,  $\text{matriz}[i][j]$  e  $\text{matriz}[j][i] = F$   
(A aresta já existe e foi encontrada uma igual, logo, remove, pois ela é interna)

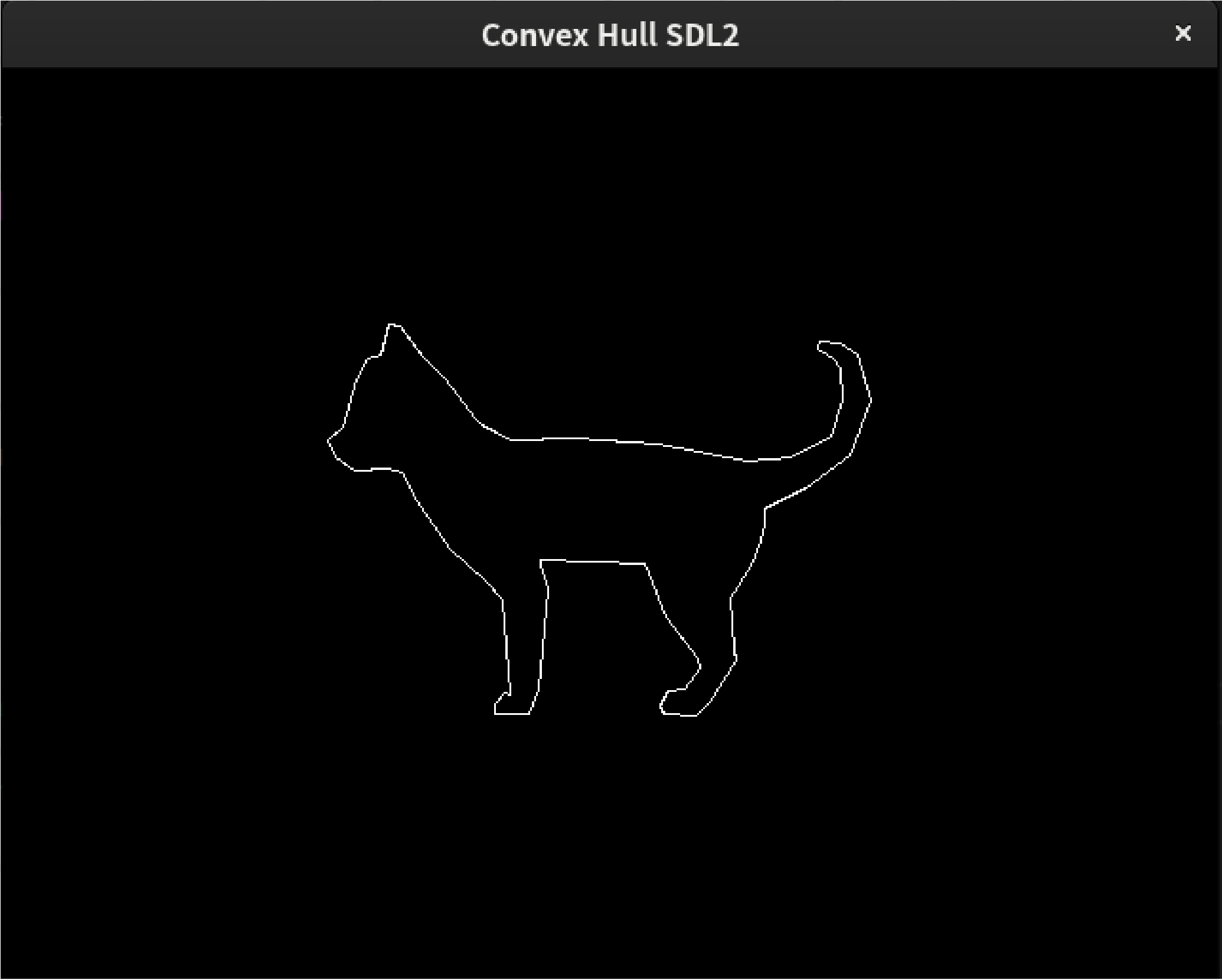


# RESULTADOS



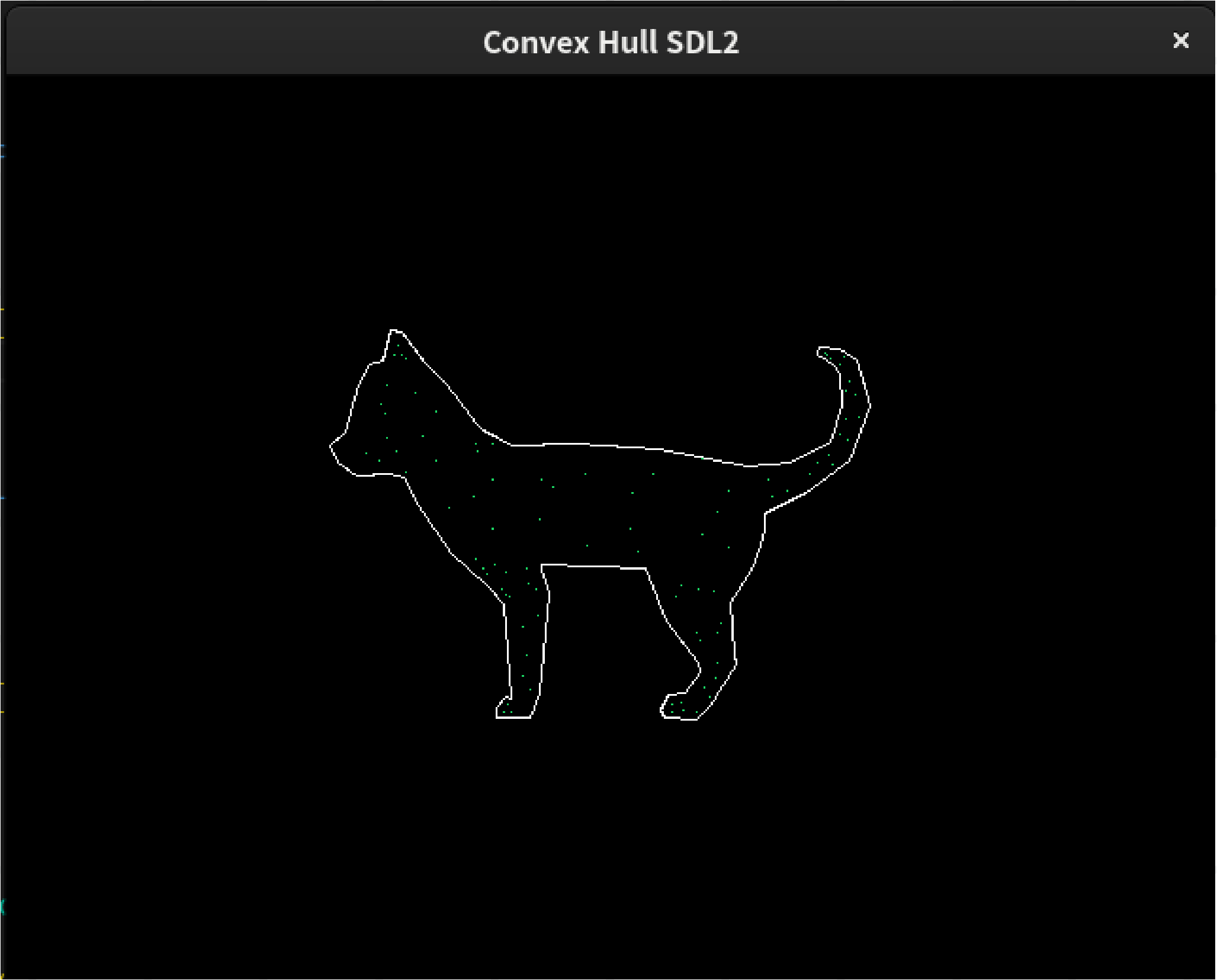


# RESULTADOS





# RESULTADOS





# BIBLIOGRAFIA

- 🔍 Slides da disciplina disponibilizados no email.
- 🔍 CARVALHO, Paulo C. P., FIGUEIREDO, L.H. Introdução à Geometria Computacional, Rio de Janeiro: IMPA, 1991
- 🔍 Modelo utilizado no tema: [link](#)

# OBRIGADO



Apresentação feita por Luis Fernando e Samuel Vieira

Repositório em <https://github.com/samue1v/computational-Geometry/>