

A3 Computação Gráfica

Espada Curva Rúnica



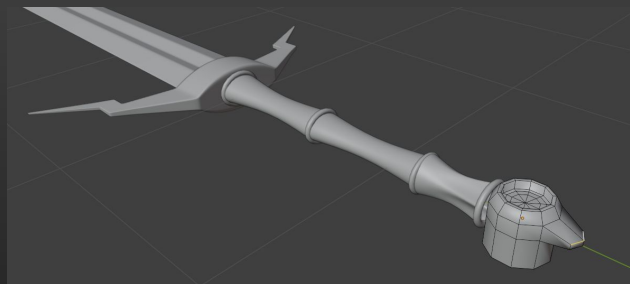
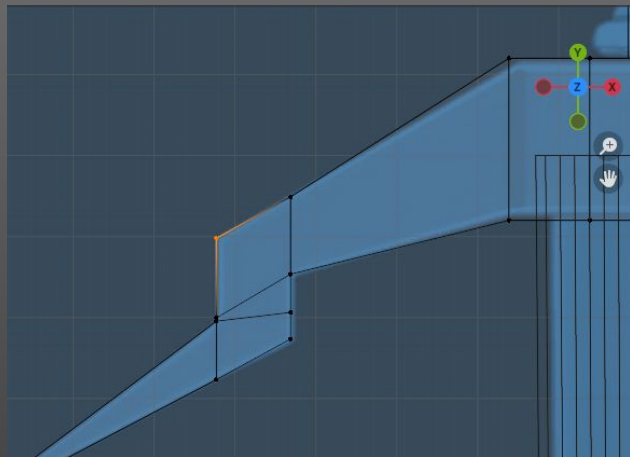
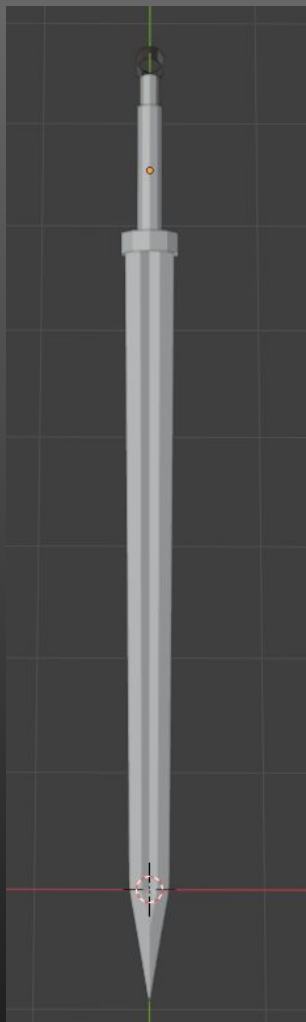
Matheus Pereira Silva
Nilso José Miguel da Silva Júnior

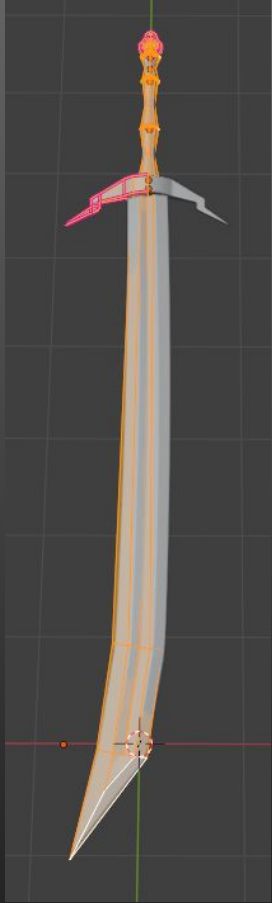
Mentores:
Adriana Neves
Vitor Leães

Criação do modelo 3D

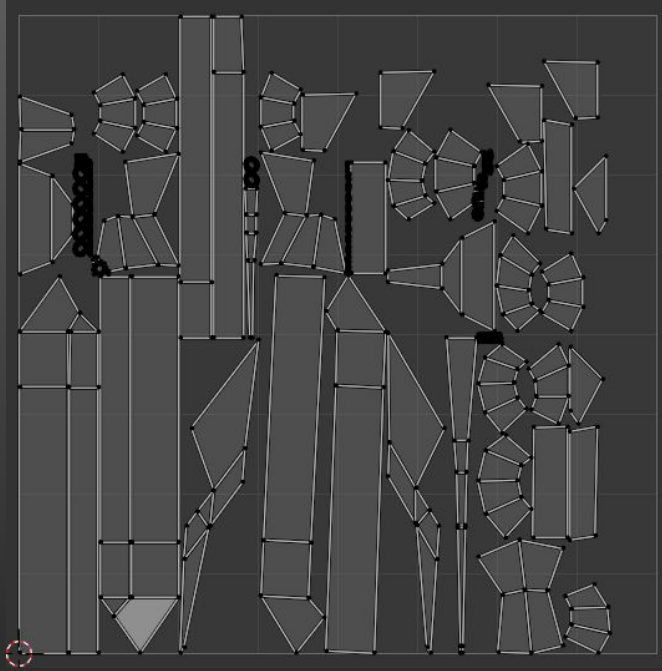
- ❖ Modelo criado do 0 no Blender
- ❖ Aplicação de texturas
- ❖ Iluminação Ray Tracing e Ambient occlusion
- ❖ UV Unwrapping
- ❖ Bake de texturas
- ❖ Exportação do modelo em .obj







UV Unwrap



Bake



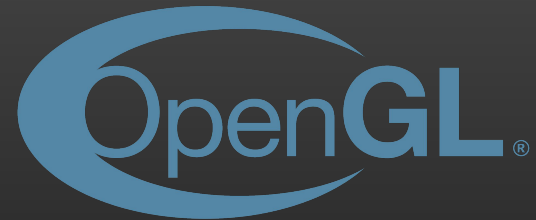
Objects	10 / 13
Vertices	27,003 / 27,003
Edges	53,816 / 53,816
Faces	26,816 / 26,816
Triangles	53,632 / 53,632



Renderização 3D com Pygame, OpenCV e OpenGL

Funcionalidades:

- ❖ Interatividade
- ❖ Transformações
- ❖ Rotação de camera
- ❖ Iluminação básica
- ❖ Textura

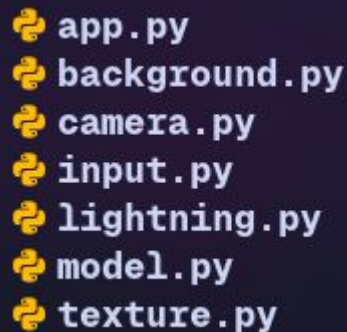


Estrutura do código

Classes:

- ❖ App
- ❖ ResourceManager
- ❖ Model
- ❖ Background
- ❖ Texture
- ❖ Camera
- ❖ Input
- ❖ Lightning

arquivos



app.py
background.py
camera.py
input.py
lightning.py
model.py
texture.py

app.py

```
from background import Background
from camera import Camera
from input import Input
from lightning import Lightning
from model import Model
```

texture.py

```
def load_texture(self, filename):  
    img = cv2.imread(filename, cv2.IMREAD_UNCHANGED)  
    if img is None:  
        raise Exception(f"Failed to load texture: {filename}")
```

input.py

```
def handle_events(self, event, model, camera):  
    if event.type in [pg.KEYDOWN]:  
        self.keyboard_input(event, model, camera)  
    elif event.type in [  
        pg.MOUSEWHEEL,  
        pg.MOUSEBUTTONDOWN,  
        pg.MOUSEBUTTONUP,  
        pg.MOUSEMOTION,  
    ]:  
        self.mouse_input(event, camera)
```

model.py

```
class Model:  
    def __init__(self, filename, texture_file):  
        self.vertices = []  
        self.texcoords = []  
  
        self.normals = []  
        self.faces = []  
        self.position = [0.0, 0.0, 0.0]  
        self.rotation = [0.0, 0.0, 0.0]  
        self.scale = 1.0  
        self.display_list = None  
  
        self.load_model(filename)  
        self.texture = Texture(texture_file)  
        self.create_display_list()
```


app.py

```
class App:
    def __init__(self):
        self.display = [1500, 800]
        self.running = False
        self.clock = pg.time.Clock()
        self.fps = 60
        self.frame_count = 0
        self.last_time = 0
        self.rotation_angle = 0

        self.resource_manager = ResourceManager()
        self.camera = Camera()
        self.input = Input()
        self.lightning = Lightning(intensity=1.0, color=(1.0, 1.0, 1.0))
```

```
# Destrutor
def cleanup(self):
    if self.background and hasattr(self.background, "texture"):
        self.background.texture.delete()

    for model in self.models.values():
        if hasattr(model, "texture"):
            model.texture.delete()
```

```
class ResourceManager:
    def __init__(self):
        self.models = {}
        self.textures = {}
        self.background = None
        self.assets_path = "assets"

    def load_resources(self):
        try:
            background_path = os.path.join(self.assets_path, "Background.png")
            self.background = Background(background_path)

            sword_obj_path = os.path.join(self.assets_path, "Sword.obj")
            sword_texture_path = os.path.join(self.assets_path, "Sword.png")
            self.models["sword"] = Model(sword_obj_path, sword_texture_path)

        except Exception as e:
            print(e)
            pg.quit()
            return False
        return True
```

app.py

OpenGL

```
def setup_opengl(display):
    glEnable(GL_DEPTH_TEST)
    glEnable(GL_CULL_FACE)
    glCullFace(GL_BACK)
    glShadeModel(GL_SMOOTH)

    # Vsync
    pg.display.gl_set_attribute(pg.GL_SWAP_CONTROL, 1)

    glMatrixMode(GL_PROJECTION)
    gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
    glMatrixMode(GL_MODELVIEW)
```

Pygame

```
def setup_display(self):
    pg.init()
    pg.display.gl_set_attribute(pg.GL_DEPTH_SIZE, 24)
    pg.display.set_mode(self.display, DOUBLEBUF | OPENGGL)

    setup_opengl(self.display)

    if not self.resource_manager.load_resources():
        return False

    self.input.print_controls()

    self.last_time = pg.time.get_ticks()
    return True
```

“The Main Loop: This stage repeats continuously (typically 60 times per second), while the application is running and consists of the following three substages:

- Process Input: Check if the user has performed any action that sends data to the computer, such as pressing keys on a keyboard or clicking buttons on a mouse.
- Update: Changing values of variables and objects.
- Render: Create graphics that are displayed on the screen.”

(STEMKOSKI; PASCALE, 2021, p.26)

```
def handle_events(self):
    for event in pg.event.get():
        if event.type == pg.QUIT:
            self.running = False
            return

    self.input.handle_events(
        event, self.resource_manager.models["sword"], self.camera
    )
```

```
def update(self):
    current_time = pg.time.get_ticks()

    # FPS display
    self.frame_count += 1
    if current_time - self.last_time > 1000:
        fps = self.frame_count
        pg.display.set_caption(f"A3 CG | FPS: {fps}")
        self.frame_count = 0
        self.last_time = current_time

    if self.input.rotation_xyz[0]:
        self.resource_manager.models["sword"].rotation[0] += 0.3
    elif self.input.rotation_xyz[1]:
        self.resource_manager.models["sword"].rotation[1] += 0.3
    elif self.input.rotation_xyz[2]:
        self.resource_manager.models["sword"].rotation[2] += 0.3

    # Atualizar iluminacao
    self.lightning.update()
```

```
def render(self):
    glClearColor(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    # Remover iluminacao para aplicar o background
    glDisable(GL_LIGHTING)
    if self.resource_manager.background:
        self.resource_manager.background.render()

    # Aplicar iluminacao de volta
    glEnable(GL_LIGHTING)





    # Aplicar iluminacao antes de renderizar o cenario
    self.lightning.apply()

    self.camera.apply()

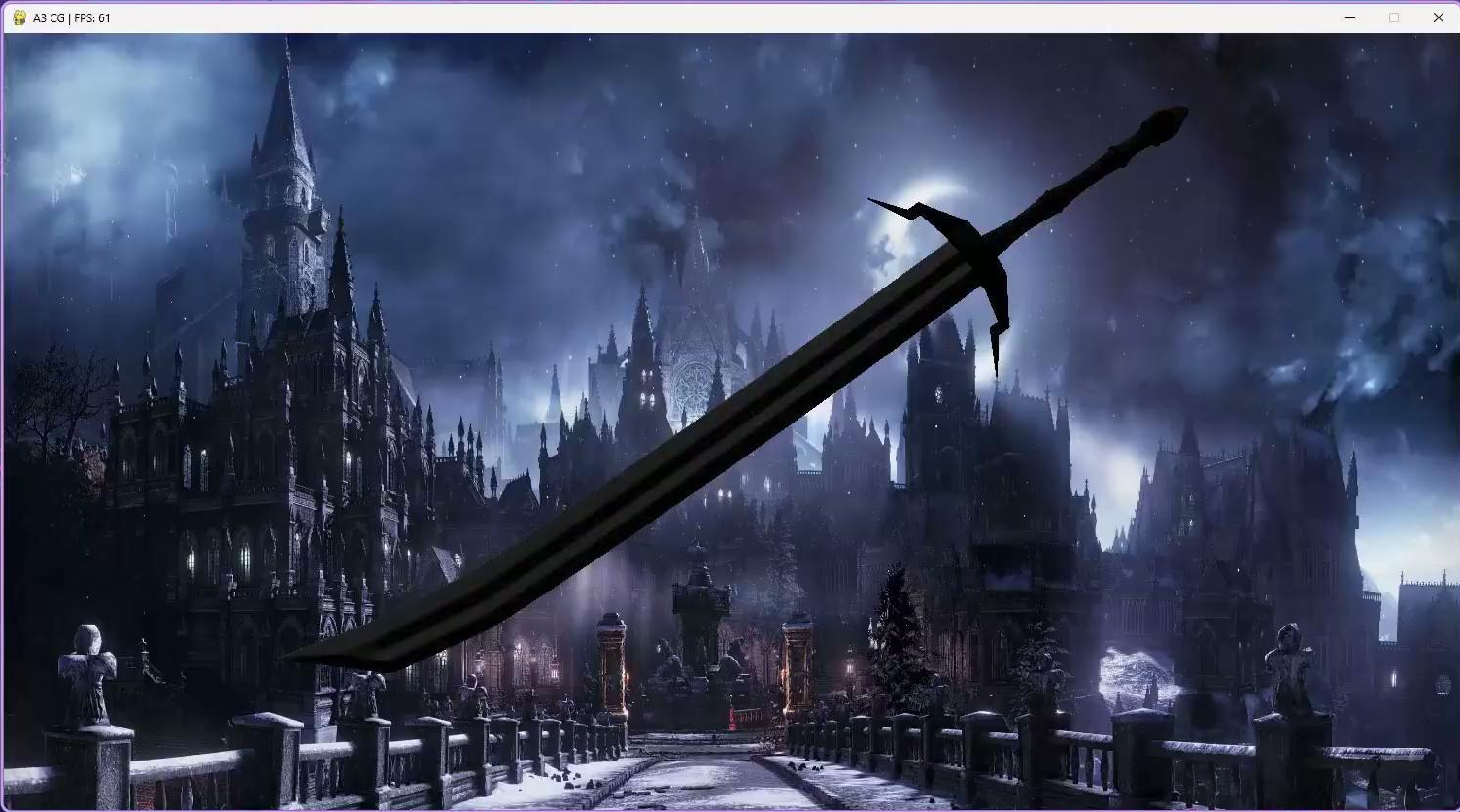
    glPushMatrix()
    glRotatef(self.rotation_angle, 0, 1, 0)
    self.resource_manager.models["sword"].render()
    glPopMatrix()

    pg.display.flip()
```

Comandos

- ❖ Rotação: WASD
- ❖ Aumentar escala: =
- ❖ Diminuir escala: -
- ❖ +Zoom: Q ou M3 para cima
- ❖ -Zoom: E ou M3 para baixo
- ❖ Translação:    
- ❖ Transladar para frente: F
- ❖ Transladar para atrás: B
- ❖ Alternar rotação de eixo: X, Y ou Z

```
# Rotacao
"rotate_left": pg.K_a,
"rotate_right": pg.K_d,
"rotate_up": pg.K_w,
"rotate_down": pg.K_s,
# Escala
"scale_up": pg.K_EQUALS,
"scale_down": pg.K_MINUS,
# Zoom
"zoom_in": pg.K_q,
"zoom_out": pg.K_e,
# Translacao
"translate_left": pg.K_LEFT,
"translate_right": pg.K_RIGHT,
"translate_up": pg.K_UP,
"translate_down": pg.K_DOWN,
"translate_forward": pg.K_f,
"translate_backwards": pg.K_b,
# Alternar rotacao de eixo
"toggle_rotation_x": pg.K_x,
"toggle_rotation_y": pg.K_y,
"toggle_rotation_z": pg.K_z,
```

Referências

- DEMES, Lennart. The Public Domain 3D Library. **ambientCG**. Disponível em: <https://ambientcg.com/>. Acesso em: 27 nov. 2024.
- STEMKOSKI, Lee; PASCALE, Michael. **Developing Graphics Frameworks with Python and OpenGL**. 1. ed. Boca Raton: CRC Press, 2021. 25-82 p. ISBN 9781003181378.
- BATOČANIN, Vladimir. Advanced OpenGL in Python with PyGame and PyOpenGL. **Stack Abuse**, 2019. Disponível em: <https://stackabuse.com/advanced-opengl-in-python-with-pygame-and-pyopengl/>. Acesso em: 29 nov. 2024.
- KINSLEY, Harrison. PyOpenGL Basics: OpenGL with PyOpenGL introduction and creation of Rotating Cube. **Python Programming Tutorials**, 2014. Disponível em: <https://pythonprogramming.net/opengl-rotating-cube-example-pyopengl-tutorial/>. Acesso em: 30 dez. 2024.