Documentação: Trabalho sobre o conceito de Tombamento (ICF)

Alunos: Gabriel Papa Alcântara Andrade;
Tales Noujain Silveira;
Nicolas Simonetto

Sumário:

1 - Sobre o Projeto:

O nosso código foi criado com o intuito de explorar mais o conceito de Tombamento, o qual, achamos que foi menos explorado do que deveria. Usando da linguagem de programação do Blender 3.6 em conjunto com a linguagem de programação Python e o uso das bibliotecas bpy e, math, desenvolvemos um programa que simula o Tombamento em gravidades diferentes, angulações diferentes, alturas diferentes e com materiais diferentes.

2 - Sobre o código

O código em si é razoavelmente simples. Primeiramente importamos as bibliotecas que precisaríamos usar durante o processo de realização do projeto, e assim começamos a escrever funções não tão simples obtidas através da biblioteca bpy.

Primeiramente, *"setamos"* uma função para limpar a cena anterior, essa função faz com que os corpos gerados na simulação anterior sejam excluídos e não haja interferência com a simulação que está acontecendo:

```
def limpar_cena():
    bpy.ops.object.select_all(action='DESELECT')
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()
```

Agora criamos uma função que criará os objetos que utilizaremos na simulação, o corpo que sofre o tombamento, seu nome, o local em que vai ser colocado, sua rotação e o seu tamanho. Realizamos então o comando "type = ACTIVE" para que ele seja o objeto que sofrerá a ação das forças que colocamos mais tarde no código:

```
def adicionar_objeto_primitivo(nome, tipo, location, rotation=(0, 0, 0), scale=(1, 1, 1)):
    if tipo == 'uv_sphere':
        bpy.ops.mesh.primitive_uv_sphere_add(segments=32, ring_count=16, radius=2.0, location=location, rotation=rotation)
    elif tipo == 'cube':
        bpy.ops.mesh.primitive_cube_add(size=2, location=location, rotation=rotation)
        bpy.context.object.scale = scale
    elif tipo == 'rectangle':
        bpy.ops.mesh.primitive_cube_add(size=2, location=location, rotation=rotation)
        bpy.context.object.scale = scale

    obj = bpy.context.active_object
    obj.name = nome

    bpy.ops.rigidbody.object_add(type='ACTIVE')
    return obj
```

A palavra "size" significa tamanho e media o tamanho do objeto que desejamos colocar, e colocamos o nome do corpo de "obj" para que seja mais fácil implementarmos funções a ele mais tarde no código.

Escrevemos um código bem semelhante para a criação do nosso chão que servirá como rampa e será alocado um pouco abaixo do nosso "obj":

```
def adicionar_chao(location, rotacao):
    bpy.ops.mesh.primitive_plane_add(size=25, location=location, rotation=rotacao)
    chao = bpy.context.active_object
    chao.name = "Chao"

    bpy.ops.rigidbody.object_add(type='PASSIVE')
    return chao
```

Outra função extremamente semelhante para construir um chão passivo na simulação:

```
def adicionar_geral(location):
    bpy.ops.mesh.primitive_plane_add(size=100, location=(0,0,-2.1))
    geral = bpy.context.active_object
    geral.name = "floor"

    bpy.ops.rigidbody.object_add(type='PASSIVE')
    return chao
```

O termo “PASSIVE” que existe em ambas as funções indica que o objeto é meramente passivo, ou seja, ele possui atrito e ele só entrará em contato com o corpo ativo, mas este não sofre com as ações das forças que colocaremos mais tarde no código.

A última função do código, e talvez a mais importante server para que possamos aplicar física aos objetos, e mais importante, que haja atrito entre os objetos que variará conforme um dicionário que faço mais para frente. Ela recebe o tipo de objeto, sua massa, seu atrito/fricção e sua resistência (que não importa muito para o nosso código).

```
def configurar_fisica(objeto, tipo, massa, friccao, restituicao):
    objeto.rigid_body.type = tipo
    objeto.rigid_body.mass = massa
    objeto.rigid_body.friction = friccao if friccao is not None else 0.0
    objeto.rigid_body.restitution = restituicao if restituicao is not None else 0.0
```

Agora acionamos as funções dentro do código a partir do texto “nome da função()”, que executará a função sem que nada precise acontecer para que ela seja acionada. Como exemplo:

```
# Limpar a cena
limpar_cena()
```

Lidamos com algumas adversidades ao utilizar o blender, uma vez que este não aceita a utilização de “input” em seu código. Todas as vezes que eu colocava esse comando no código, o aplicativo falhava e me obrigava a fechá-lo. Logo, usamos um jeito interativo dentro do próprio código para que o usuário possa usá-lo. Explicaremos melhor no próximo capítulo.

```
# Escolher o objeto diretamente
objeto_escolhido = 'esfera'  # Pode ser 'esfera', 'quadrado' ou 'retangulo'

# Adicionar objeto selecionado
if objeto_escolhido.lower() == 'esfera':
    objeto_ativo = adicionar_objeto_primitivo("Esfera", 'uv_sphere', (0, 0, 5), rotation=(0, 0, 0))
elif objeto_escolhido.lower() == 'quadrado':
    objeto_ativo = adicionar_objeto_primitivo("Quadrado", 'cube', (0, 0, 5), rotation=(0, 0, 0))
elif objeto_escolhido.lower() == 'retangulo':
    objeto_ativo = adicionar_objeto_primitivo("Retangulo", 'rectangle', (0, 0, 5), rotation=(0, 0, 0), scale=(1, 1, 3))
else:
    print("Escolha inválida. Por favor, ajuste o valor de 'objeto_escolhido' para 'esfera', 'quadrado' ou 'retangulo'.")
    bpy.ops.wm.quit_blender()
```

Nessas linhas, eu peço que a pessoa digite o objeto, e com um simples uso de “if”, “elif” e “else”, faço com que a função “adicionar_objeto_primitivo” construa um objeto conforme a vontade da pessoa que utiliza esse código. Caso a opção não esteja dentro dos conformes, o código vai pedir que você digite um objeto que está dentro das possibilidades.

Nas próximas linhas, eu aciono as outras funções que citei anteriormente e faço algumas nomeações para facilitar o dinamismo do código, como chamar o ângulo do surgimento do objeto e do chão de “angulo” para não repetir o mesmo código diversas vezes durante a escrita. Faço o mesmo com a massa do objeto para que esse tenha Peso:

```
# Angulação do Chão:
angulo = math.radians(20)

# Adicionar chão
chao = adicionar_chao((0, 0, 0), (angulo, 0, 0))

# Adicionar o floor geral:
floor = adicionar_geral((0,0,0))
# Definir altura inicial
# A altura mínima muda de acordo com o objeto em questão:
altura_inicial = 4.4  # Substitua pelo valor desejado

# Configurar posição inicial
objeto_ativo.location.z = altura_inicial

# Massa do objeto:
m = float(10) #Substitua pelo valor desejado
```

Os comentários ajudam o usuário a entender o que ele deve fazer, seja por substituir o valor, ou mesmo entender o que os códigos fazem.

Agora, escrevemos um dicionário com possibilidades de atrito que o usuário pode explorar, o material escolhido ditará a cor dos objetos, o coeficiente de atrito entre eles e ajudará a pessoa que observa o código a entender como funcionaria a simulação na vida real com materiais reais.

```
# Configurar física para os objetos com base no material
material_objeto = 'Borracha Asfalto'  # Substitua pelo material real do objeto
coeficientes_atrito = {
    'Alumínio e Aço Carbono': 0.61,
    'Borracha Asfalto': 0.4,
    'Cobre e Ferro Fundido': 1.1,
    'Grafite e Vidro': 0.1,
    'Atrito Nulo': 0.0,
    'Atrito Muito': 50.0,
}

if material_objeto in coeficientes_atrito:
    coeficiente = coeficientes_atrito[material_objeto]
    configurar_fisica(objeto_ativo, 'ACTIVE', 1.0, coeficiente, None)

    # Adicionar ou modificar para definir a cor do material do objeto
    bpy.ops.object.material_slot_remove({'object': objeto_ativo, 'confirm': False})
    bpy.ops.object.material_slot_add({'object': objeto_ativo})
    material = bpy.data.materials.new(name="ObjectMaterial")
    objeto_ativo.data.materials[0] = material

    # Mapear materiais para cores específicas
    cores_mapeadas = {
        'Alumínio e Aço Carbono': (0.3, 0.3, 0.5),      # Vermelho
        'Borracha Asfalto': (0.3, 0.3, 0.3),            # Preto
        'Cobre e Ferro Fundido': (0.904, 0.498, 0.196),     # Cobre
        'Grafite e Vidro': (0.1, 0.1, 0.1),             # Cinza
        'Atrito Nulo': (1, 1, 1),                   # Branco
        'Atrito Muito': (1, 0, 1),                  # Magenta
    }

    if material_objeto in cores_mapeadas:
        cor = cores_mapeadas[material_objeto]
        material.diffuse_color = cor + (10,)  # Adiciona um componente alpha (opacidade)
    else:
        print(f"Cor para o material '{material_objeto}' não mapeada.")
```

Fazemos então outro dicionário para que o usuário possa selecionar o planeta em que ele gostaria de simular o tombamento, ou seja, definir a gravidade do ambiente de simulação.

```
# Configurar gravidade
gravidade_universal = 'Terra'  # Substitua pela gravidade real
gravidade_usada = {
    'Lua': -1.62,
    'Terra': -9.81,
    'Marte': -3.71,
    'Sol': -274,
}

if gravidade_universal in gravidade_usada:
    gravidade = gravidade_usada[gravidade_universal]
    bpy.context.scene.gravity = (0, 0, gravidade)
else:
    print(f"Gravidade '{gravidade_universal}' não encontrado na tabela de gravidade.")
    bpy.ops.wm.quit_blender()
```

“bpy.context.scene.gravity” é a função que faz com que a gravidade seja aplicada à cena e, caso a pessoa não selecione uma gravidade possível, o programa retornará que a gravidade não foi encontrada.

Para finalizar, aplicamos então o início da simulação e a sua quantia de frames que implicam em quanto tempo será a duração da simulação.

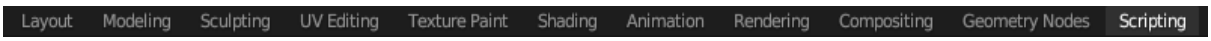
```
# Configurar animação
bpy.context.scene.frame_start = 1
bpy.context.scene.frame_end = 100

# Adicionar rotação ao objeto
objeto_ativo.rotation_euler = (angulo, 0, 0)

# Reproduzir a animação
bpy.ops.screen.animation_play()
```

3 - Como usar:

Para que o usuário possa usufruir da nossa simulação, ele necessita da versão 3.6 do Blender baixada e deve colocar o código na aba “Scripting” do aplicativo:



Assim feito, para que o usuário possa modificar a simulação, por ausência do input, ele deve modificar o número no código para que haja a mudança na simulação.
Como por exemplo:

```
# Configurar física para os objetos com base no material
material_objeto = 'Borracha Asfalto'  # Substitua pelo material real do objeto
coeficientes_atrito = {
    'Alumínio e Aço Carbono': 0.61,
    'Borracha Asfalto': 0.4,
    'Cobre e Ferro Fundido': 1.1,
    'Grafite e Vidro': 0.1,
    'Atrito Nulo': 0.0,
    'Atrito Muito': 50.0,
```

Assim, deve “clicar” no botão para iniciar o script:

(O botão em forma de seta, apontada para a direita). E para melhor visualização, pode selecionar a opção “Layout no aplicativo e ver a animação de simulação do código,

4 - Dificuldades do Código:

Primeiramente, o código utiliza uma biblioteca não usual, o que dificulta o seu entendimento no início, mas a biblioteca não possui um nível de dificuldade elevado. Um dos maiores problemas que enfrentamos é a não utilização de “input” que torna nosso programa menos dinâmico e mais manual.

Outra dificuldade é o pouco conteúdo disponível na internet sobre essa biblioteca, já que a sua utilização não é muito usual. Mas tirando os contratempos, o nosso código é simples e qualquer um pode se divertir ao usá-lo.

5 - Fontes:

- https://docs.blender.org/api/current/info_quickstart.html
- https://www.blender.org
- https://www.youtube.com/watch?v=cyt0O7saU4Q