Trabalho 2 - Computação Gráfica

Pedro Hiroshi Ely Ito

Departamento de Ciência da Computação
Instituto de Ciências Matéticas e de Computação
Universidade de São Paulo

13/12/2022

Componentes do código

O código se baseia nos códigos apresentados em aula, com modificações que permitem carregar modelos de uma pasta de forma simples, permitindo passar argumentos para transformações 3D, além das modificações necessárias para atender aos requisitos do trabalho.

Carregando modelos e texturas

```
def load_model(path):
    modelo = load_model_from_file(path)
    model_name = path.split('/')[-1]
    ### inserindo vertices do modelo no vetor de vertices
    print(f'Processando modelo {model_name}. Vertice
        inicial:',len(vertices_list))
    vertices_dict.append(len(vertices_list))
    for face in modelo['faces']:
        for vertice_id in face[0]:
            vertices_list.append( modelo['vertices'][
                vertice_id-1] )
        for texture_id in face[1]:
            textures_coord_list.append( modelo['texture'
                ][texture_id-1] )
        for normal_id in face[2]:
            normals_list.append( modelo['normals'][
                normal_id-1] )
    print(f'Processando modelo {model_name}. Vertice
        final:',len(vertices_list))
```

Carregando modelos e texturas

```
for i in sorted(os.listdir("./meshes")):
    load_model("./meshes"+'/'+i)
    print(35*'-')

### carregando textura equivalente e definindo um id (
    buffer): use um id por textura!
texture_path = sorted(os.listdir("./textures/"))
light_id = 0
for i in range(len(texture_path)):
    load_texture_from_file(i, "./textures/"+"/"+
        texture_path[i])
    if "luz" in texture_path[i]:
        light_id = i
    print("carregando textura:", texture_path[i])
    print(35*'-')
```

Carregando modelos e texturas

As texturas e modelos correspondentes têm mesmo nome, assim, ao ordenar as pastas, podemos criar os buffers de textura de forma automática.

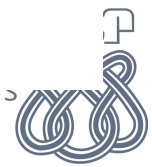
Renderizando

Para renderizar os modelos posteriormente, armazenamos o vértice inicial e final de cada modelo.

```
vertices_dict = np.unique(vertices_dict)
vertices_dict = list(zip(vertices_dict, vertices_dict
    [1:]))
```

Renderizando I

```
def draw_model(model_id, t_x=0., t_y=0., t_z=0., r_x =
    0.0, r_y = 1.0, r_z = 0.0):
    # aplica a matriz model
    angle = 0.0
    # escala
    s_x = 1.0; s_y = 1.0; s_z = 1.0;
    mat_model = model(angle, r_x, r_y, r_z, t_x, t_y, t_z
        , s_x, s_y, s_z)
    loc_model = glGetUniformLocation(program, "model")
    glUniformMatrix4fv(loc_model, 1, GL_TRUE, mat_model)
    #### define parametros de ilumincao do modelo
    ka = 0.3 # coeficiente de reflexao ambiente do modelo
    kd = 0.3 # coeficiente de reflexao difusa do modelo
    ks = 0.9 # coeficiente de reflexao especular do
        modelo
    ns = 64.0 # expoente de reflexao especular
```



Renderizando II

```
    loc_ka = glGetUniformLocation(program, "ka") #
        recuperando localizacao da variavel ka na GPU
    glUniform1f(loc_ka, ka) ### envia ka pra gpu
    loc_kd = glGetUniformLocation(program, "kd") #
        recuperando localizacao da variavel kd na GPU
    glUniform1f(loc_kd, kd) ### envia kd pra gpu
    loc_ks = glGetUniformLocation(program, "ks") #
        recuperando localizacao da variavel ks na GPU
    glUniform1f(loc_ks, ks) ### envia ks pra gpu
    loc_ns = glGetUniformLocation(program, "ns") #
        recuperando localizacao da variavel ns na GPU
    glUniform1f(loc_ns, ns) ### envia ns pra gpu
    if model_id == light_id:
        loc_light_pos = glGetUniformLocation(program, "
            lightPos") # recuperando localizacao da
            variavel lightPos na GPU
        glUniform3f(loc_light_pos, t_x, t_y, t_z) ###
            posicao da fonte de luz
```

Renderizando III

```
        ka = 1
        kd = 1
        ks = 1
        ns = 1000.0
    #define id da textura do modelo
    glBindTexture(GL_TEXTURE_2D, model_id)
    # desenha o modelo
    glDrawArrays(GL_TRIANGLES, vertices_dict[model_id
        ][0],
                 vertices_dict[model_id][1]-vertices_dict
                     [model_id][0]) ## renderizando
```

Loop principal I

```
glEnable(GL_DEPTH_TEST) ### importante para 3D

ang = 0.1
while not glfw.window_should_close(window):
    glfw.poll_events()
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glClearColor(0.2, 0.2, 0.2, 1.0)
    if polygonal_mode==True:
        glPolygonMode(GL_FRONT_AND_BACK,GL_LINE)
    if polygonal_mode==False:
        glPolygonMode(GL_FRONT_AND_BACK,GL_FILL)
    ang += 0.01
    draw_model(0, t_z=car_tz)#carro
    draw_model(1)#cadeira
    draw_model(2, t_y=3., t_z=np.cos(ang)*5, r_z=np.cos
        (5*ang))#caixa
    draw_model(3)#casa
```

Loop principal II

```
    draw_model(4, np.cos(ang)*5, 3.0 , np.sin(ang)*5)#luz
    draw_model(5)#caneca
    draw_model(6)#ceu
    draw_model(7)#mesa
    draw_model(8)#terreno
    draw_model(9)#arvores
    mat_view = view()
    loc_view = glGetUniformLocation(program, "view")
    glUniformMatrix4fv(loc_view, 1, GL_TRUE, mat_view)
    mat_projection = projection()
    loc_projection = glGetUniformLocation(program, "
        projection")
    glUniformMatrix4fv(loc_projection, 1, GL_TRUE,
        mat_projection)
    # atualizando a posicao da camera/observador na GPU
        para calculo da reflexao especular
```

Loop principal III

```
        loc_view_pos = glGetUniformLocation(program, "viewPos
            ") # recuperando localizacao da variavel viewPos
            na GPU
        glUniform3f(loc_view_pos, cameraPos[0], cameraPos[1],
             cameraPos[2]) ### posicao da camera/observador (
            x,y,z)
        glfw.swap_buffers(window)
glfw.terminate()
```