Ray Tracing

João Vitor Farias Silva (jvfsilva@inf.ufpel.edu.br)
Graziele Fagundes Martins (gfmartins@inf.ufpel.edu.br)

Aplicação escolhida

Para este trabalho escolhemos a aplicação gráfica Ray Tracing Recursivo. Ray tracing é uma técnica de renderização que simula o comportamento da luz para gerar imagens realistas em gráficos 3D. O algoritmo traça os caminhos dos raios de luz a partir da câmera (ou olho) para determinar a cor dos pixels na imagem final, levando em consideração como esses raios interagem com os objetos na cena.

Na implementação recursiva do ray tracing, cada vez que um raio atinge um objeto, novos raios são gerados para simular efeitos como reflexões e refrações. A recursão ocorre porque, a partir de cada ponto de interseção, o algoritmo lança novos raios que podem, por sua vez, gerar mais interseções e novos raios, até que um limite de profundidade de recursão seja atingido.

Linguagens

Para a interface visual, utilizamos Python, que oferece simplicidade e facilidade na criação de GUIs. E para os cálculos complexos necessários para a geração de imagens, optamos por C, que proporciona maior desempenho e eficiência no processamento de operações matemáticas intensivas.

Interface entre as linguagens

Para integrar as linguagens Python e C em nosso projeto, utilizamos a biblioteca "*ctypes*" do Python para facilitar a comunicação entre essas linguagens. O processo envolve compilar o código-fonte C como uma biblioteca compartilhada e depois acessá-la a partir do código Python.

1. Compilação do Código C

O código C é compilado para gerar uma biblioteca compartilhada no formato .so (shared object). O comando utilizado para essa tarefa é:

```
gcc -fPIC -shared -o raytracing.so raytracing.c utils.c
```

Neste comando:

- -fPIC instrui o compilador a gerar código independente de posição, necessário para criar bibliotecas compartilhadas.

- -shared indica ao compilador que o objetivo é criar uma biblioteca compartilhada em vez de um executável.
- -o raytracing.so especifica o nome do arquivo da biblioteca compartilhada gerada.

Após a execução deste comando, o resultado é um arquivo raytracing.so, que contém o código compilado de raytracing.c e utils.c.

2. Utilização da Biblioteca Compartilhada em Python

Para utilizar a biblioteca compartilhada no código Python, carregamos o arquivo .so usando a biblioteca ctypes:

```
PATH = './raytracing.so'
raytracing = ctypes.CDLL(PATH, winmode=0)
```

A função ctypes.CDLL(PATH, winmode=0) carrega a biblioteca compartilhada e retorna um objeto CDLL que permite interagir com as funções definidas na biblioteca C. Depois de carregar a biblioteca, é necessário definir os tipos dos argumentos e o tipo de retorno das funções que serão utilizadas. No nosso caso, a função PerPixel é configurada da seguinte forma:

```
raytracing.PerPixel.argtypes = [ctypes.c_float, ctypes.c_float]
raytracing.PerPixel.restype = ctypes.c_int
```

- argtypes define que PerPixel espera dois argumentos do tipo float (ctypes.c_float).
- restype especifica que PerPixel retorna um valor do tipo int (ctypes.c_int).

Com essas configurações, a função PerPixel, que realiza cálculos complexos de ray tracing no código C, pode ser chamada a partir do Python. Isso nos permite gerar imagens e manipular dados de forma eficiente, aproveitando a combinação das capacidades das duas linguagens.

Resultado

Após a conclusão, o resultado esperado é mostrado na imagem abaixo (Figura 1). É importante ressaltar que a qualidade da imagem gerada depende do valor da variável “*numSimples"* no código Python, que representa o número de *samples* (amostras) por

pixel. Ao aumentar esse valor, uma imagem mais nítida será produzida, entretanto isso resultará em um maior tempo de execução.

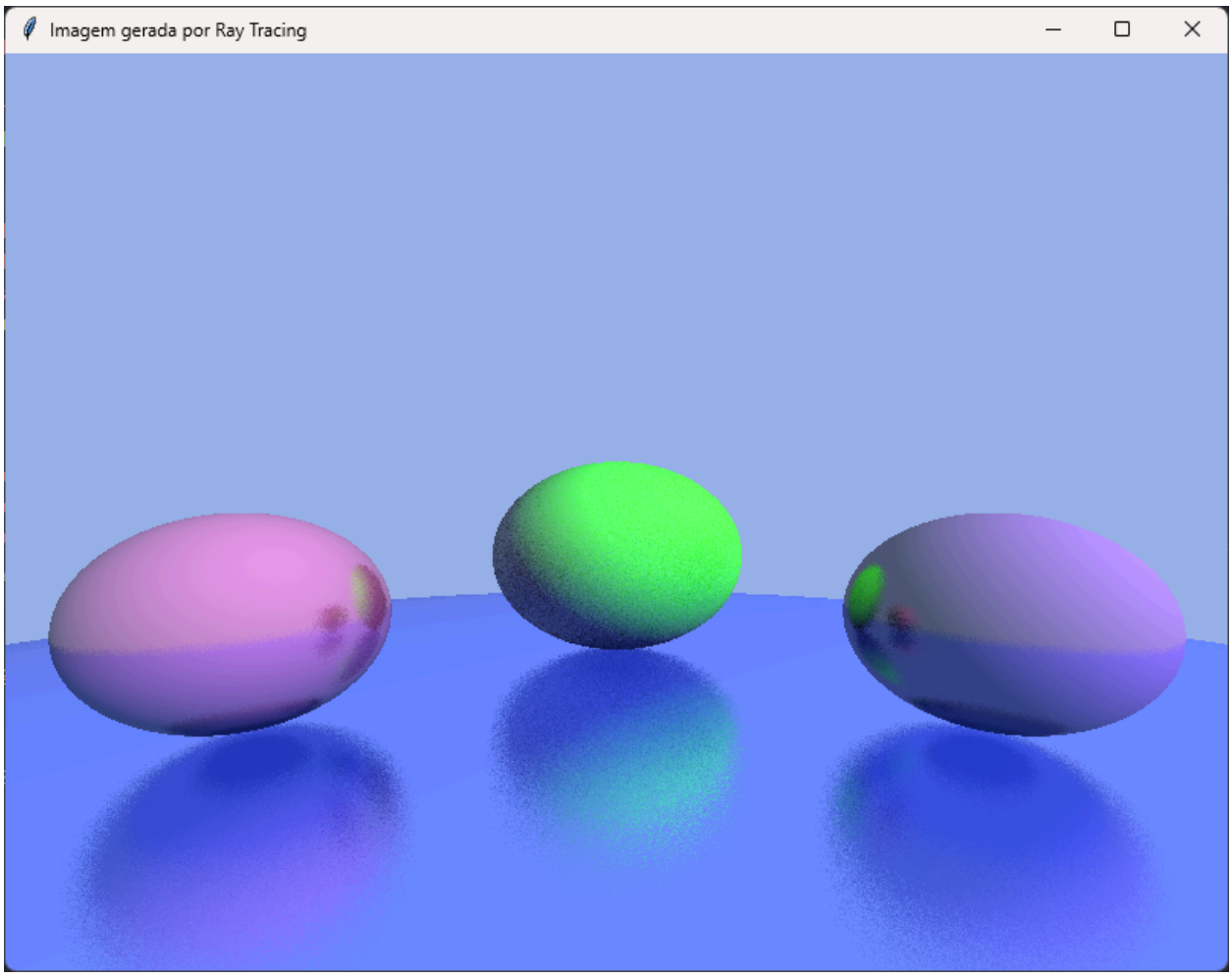

Figura 1: Imagem gerada pela aplicação.