Computação Gráfica I - MAB122 (2021-1)
Professor: João Vitor de Oliveira Silva

Primeira Tarefa Prática

Leia o enunciado todo desta tarefa antes de "colocar a mão na massa".

Seu objetivo nesta tarefa de implementação é realizar a **rasterização** de primitivas gráficas. Dentro do esqueleto desta atividade, há diferentes arquivos `.json` contendo a descrição de um gráfico a ser convertido. Por exemplo, no arquivo `polygon.json`, temos:

```
{
  "width": 320,
  "height": 200,
  "scene":[
   {  "shape": "polygon",
      "vertices": [ [50,25], [300, 25], [300, 140], [160, 180] ],
      "color": [121, 67, 91]
   }
  ]
}
```

Neste caso, temos uma cena contendo apenas um polígono, com seus vertices e cor (no espaço RGB) informados. Por enquanto, ao abrir qualquer um destes arquivos em seu navegador, apenas verá uma tela branca. Para que o gráfico seja renderizado, é necessário que termine a implementação das funções/métodos incompletos no esqueleto do Google Colab da atividade.
Se achar necessário, pode criar classes e/ou funções auxiliares. Coloque uma descrição em cada nova classe/função criada nos moldes das presente no esqueleto.

Sua solução deve ao menos ser capaz de resolver o problema de rasterização para as seguintes primitivas gráficas: `triangle`, `polygon` (caso convexo) e `circle`. Veja como cada uma dessas primitivas gráficas está definida nos arquivos `.json` para que saiba manuseá-las corretamente (abra o arquivo num editor de texto de sua preferência).

Para acelerar o processo de renderização, será necessário que se construa uma *bounding box* para cada primitiva gráfica presente em sua cena. Uma *bounding box* é um retângulo que contêm a sua primitiva gráfica, de forma que seja realizado o teste de interseção sobre a *bounding box* antes de se testar interseção sobre sua primitiva gráfica. Na Figura 1, é possível ver a *bounding box* associada a uma primitiva gráfica do tipo triângulo. Uma vez criada as bounding boxes, atualize também loop de pixels, de modo que não sejam realizados testes de interseção desnecessários.

Além disso, é possível que alguma primitiva gráfica possua uma transformação afim associada, como no arquivo `xform_polygon.json`:

```
{
  "width": 320,
  "height": 200,
  "scene":[
   {  "shape": "polygon",
      "vertices": [ [50,25], [300, 25], [300, 140], [160, 180] ],
      "color": [121, 67, 91],
      "xform": [[0.5, 0, 15], [0, 0.75, 20], [0, 0, 1]]
   }
  ]
}
```

Nestes casos, sua solução deverá renderizar a primitiva após sofrer a transformação afim especificada. *Dica: É possível usar o método* `get` ou o comando `in` *para verificar a existência de uma chave denominada* `xform` em um dicionário no Python.

Você deve também realizar um processo conhecido como *anti-aliasing*. Este processo tem o intuito de produzir bordas sem o efeito de serrilhado, que é bastante característico nas bordas. Por padrão o que fazemos é testar se o centro de um pixel $(p_x + 0.5, p_y + 0.5)$ está no interior de alguma primitiva ou não, atribuindo ou não a cor desta primitiva no processo (veja Figura 2). O anti-aliasing com o método chamado de *supersampling* realiza este teste em mais localizações do que só o centro do pixel, realizando um cálculo de cor média ao final do processo. Você deve considerar o caso em que 4 pontos são considerados, com a configuração uniforme a rotacionada em 45^o. Isso está ilustrado na Figura 3. Para mais informações, veja `https://www.inf.unioeste.br/~adair/CG/Notas%20Aula/Anti-aliasing/Algoritmos%20de%20Anti-aliasing.pdf` e `https://newbedev.com/what-are-the-differences-between-the-different-anti-aliasing\-multisampling-settings`.

Por fim, deverá implementar uma das seguintes funcionalidades adicionais na sua solução:

1. Implemente a rasterização de polígonos não convexos encontrando uma triangulação usando *ear clipping* ou *sweep line*.

2. Implemente a rasterização de polígonos não convexos com teste de interseção via *winding number*.

3. Implemente a rasterização de círculos usando triangulação. Para isso, você deve gerar um número de pontos adequado para que visualmente os triângulos se pareçam com o círculo original. Na Figura 4, é possível ver um círculo e duas possíveis triangulações (com 8 e 20 pontos, respectivamente). Caso decida por esta funcionalidade, não é necessário implementar a lógica do teste de interseção para círculos (i.e usando a eq. implicita da circunferência).

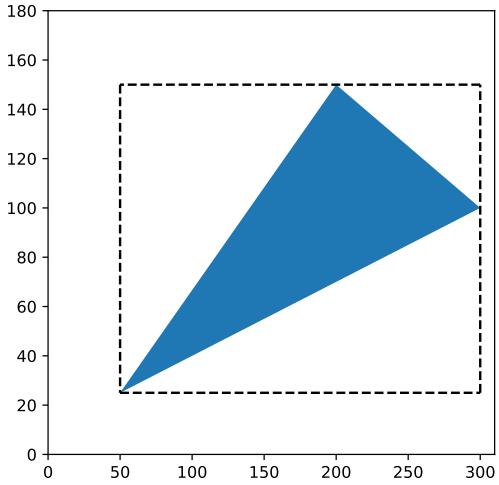

Figura 1: *Bouding box* de uma primitiva do tipo triângulo.

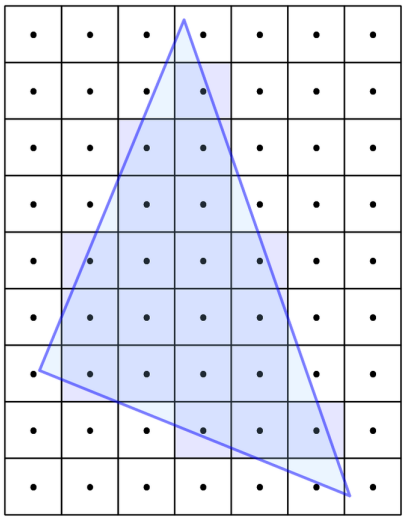

Figura 2: Processo de rasterização sem *anti-aliasing*, em que se considera apenas o centro do pixel.

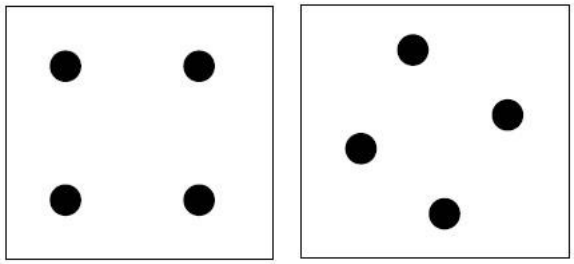

Figura 3: Pontos usados usando *supersampling*: à esquerda modo uniforme e à direita modo rotacionado.

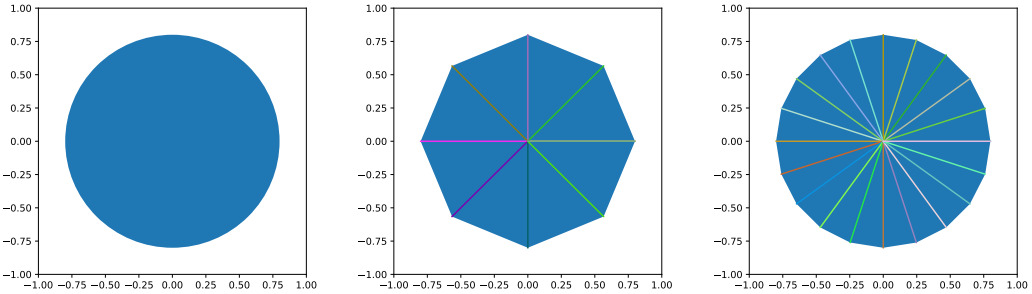

Figura 4: Círculo de centro $(0, 0)$ e raio $r = 0.8$, e duas de suas possíveis triangulações (com 8 e 20 pontos, respectivamente).

Considerações finais

- O trabalho pode ser feito de forma **individual** ou em **dupla**.
- A entrega deve ser feita pelo formulário do Google Forms da atividade. Pode-se enviar o arquivo `.ipynb` ou um link do repositório com a solução desenvolvida.

Prazo para entrega: 05/09.