Computação Gráfica I - MAB122 (2020-2)
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 `.html` contendo a descrição de um gráfico a ser convertido. Por exemplo, no arquivo `polygon.html`, temos:

```
var 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 arquivo `src/basic.renderer.js`.
Se achar necessário, pode criar classes e/ou funções auxiliares.

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 `.html` 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.

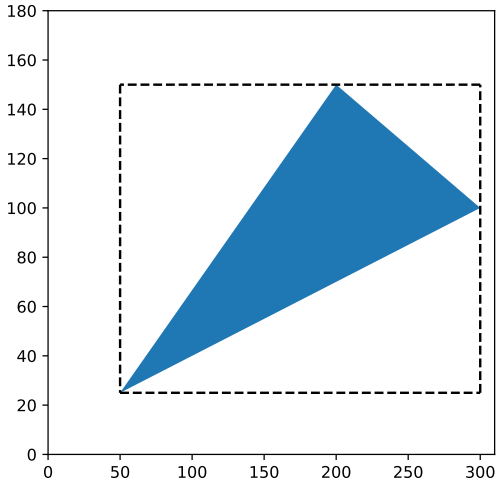

Figura 1: *Bouding box* de uma primitiva do tipo triângulo.

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

```
var 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* `hasOwnProperty` *para verificar a existência de uma chave denominada* `xform`.

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

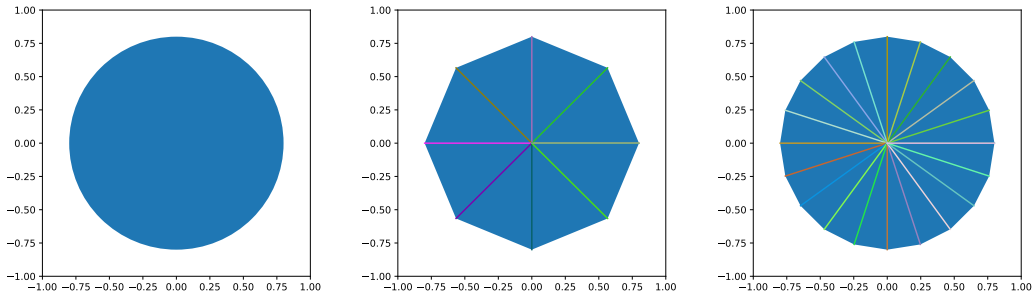

Figura 2: Círculo de centro $(0, 0)$ e raio $r = 0.8$, e duas de suas possíveis triangulações (com 8 e 20 pontos, respectivamente).

com o círculo original. Na Figura 2, 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).

Considerações finais

- É recomendável o uso do Google Chrome quando abrir os arquivos `.html`.
- O trabalho pode ser feito de forma **individual** ou em **dupla**.
- A entrega deve ser feita pela plataforma Google Classroom. Pode-se enviar um arquivo .zip ou um link do repositório com a solução desenvolvida.

Prazo para entrega: 18/04.