MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DO NORTE DE MINAS GERAIS – IFNMG
Criado pela Lei no.11.892, de 29/12/2008
Campus Montes Claros

SARAH EMANUELLE ALVES LINO
JOÃO KENNEDY SOUZA SOARES

RELATÓRIO : TRABALHO PRÁTICO 2
Visualização de Objetos 2D- Clipping e Transformações Interativas

Relatório apresentado ao Professor Wagner Ferreira de Barros como parte das exigências de avaliação da disciplina de Computação Gráfica do Curso de Bacharelado em Ciência da Computação do Instituto Federal do Norte de Minas Gerais, Campus Montes Claros.

MONTES CLAROS - MG

2025

Relatório de Desenvolvimento: Visualizador de Objetos 2D com Transformada de Viewport

A computação gráfica é uma área focada na geração de imagens a partir de modelos computacionais. No primeiro trabalho prático, exploramos o conceito fundamental da Transformada de Viewport, que mapeia coordenadas de um sistema de "mundo" para um dispositivo de exibição. Este segundo trabalho avança sobre essa base, introduzindo um pipeline de visualização mais complexo e robusto, centrado no conceito de clipping (recorte).

O clipping é o processo de identificar e remover as porções de objetos geométricos que estão fora da área de visualização, definida pela "janela" (window). Essa operação é crucial para que apenas o conteúdo contido na window seja efetivamente renderizado na viewport, otimizando o processamento e garantindo a correta visualização da cena.

Este relatório descreve a evolução do "Visualizador de Objetos 2D", detalhando a implementação de transformações interativas na window (translação, rotação e escala) e a aplicação de algoritmos de clipping canônicos para pontos, retas e polígonos.

.

2. Visão Geral e Objetivos do Projeto

O objetivo central deste projeto foi expandir a aplicação gráfica, dada no trabalho, para permitir a manipulação livre da window sobre o mundo e implementar um de clipping completo.

As novas funcionalidades principais adicionadas na implementação incluem:

- **Transformações Interativas da Window:**
    - **Movimentação:** Navegação para cima, baixo, esquerda e direita através de botões e teclas direcionais.
    - **Rotação:** Giro da window em torno de seu centro em passos pré-definidos.
    - **Zoom:** Ampliação e redução da window em 10% por clique, mantendo o centro como pivô.
- **Clipping:**
    - Implementação de um sistema que transforma objetos do mundo para um Sistema de Coordenadas do Plano de Projeção (PPC) antes do recorte.

    - **Recorte de Retas:** Implementação dos algoritmos Cohen-Sutherland e Liang-Barsky, com uma opção na interface para o usuário selecionar qual utilizar.
    - **Recorte de Polígonos:** Implementação do algoritmo de Weiler-Atherton, capaz de tratar polígonos côncavos e convexos.
- **Suporte a Cores:** O sistema agora lê e aplica cores aos objetos (pontos, retas e polígonos) definidas no arquivo XML de entrada.
- **DisplayList Avançada:** Criação de uma janela de depuração que exibe, em tempo real, as coordenadas dos objetos em cada etapa do pipeline (Mundo, PPC, Viewport) e seu status de visibilidade.

3. Desenvolvimento e Tecnologias

Para a construção do projeto, foram mantidas as ferramentas, com foco na simplicidade e na clareza do código.

- Linguagem de Programação: Python 3.12.
- Interface Gráfica: A biblioteca Tkinter, que é o toolkit padrão de GUI para Python, foi utilizada para construir a interface do usuário, incluindo a janela principal, os menus e as áreas de desenho (canvas).
- Análise de Dados (Parsing): O módulo nativo do Python xml.etree.ElementTree foi empregado para analisar os arquivos de entrada em formato XML.

A estrutura do código foi centralizada na classe Visualizador, que foi expandida para as novas funcionalidades:

- *init:* Além dos componentes anteriores, agora inicializa os novos botões de transformação (zoom, rotação), a caixa de seleção para os algoritmos de recorte de retas e realiza o binding de novas teclas (*plus, minus, period, comma*) para controle via teclado.
- *zoom, rotacionar:* Novos métodos que manipulam as dimensões e o ângulo da window, recalculam a matriz de transformação e disparam o redesenho da cena.
- *world_to_ppc*: Método que agora incorpora a lógica de rotação, transladando o objeto para a origem da window, aplicando a rotação e então o mapeando para o PPC.

- *realizar_clipping:* Método central que gerencia o processo de recorte. Ele chama os algoritmos específicos para cada tipo de objeto (clip_ponto, clip_reta_cohen_sutherland, clip_reta_liang_barsky, clip_poligono_weiler_atherton) e atualiza o estado de visibilidade (visivel: bool) de cada objeto.
- *desenhar_viewport*: O método que renderiza foi modificado para primeiro invocar realizar_clipping e, em seguida, iterar sobre os objetos, desenhando apenas aqueles cujo atributo visivel é True.

4. Implementação da Transformada de Viewport

Uma das principais diferenças em relação ao trabalho anterior é a introdução do Sistema de Coordenadas do Plano de Projeção (PPC). A transformação agora ocorre em múltiplos estágios: Mundo → PPC → Clipping → Viewport.

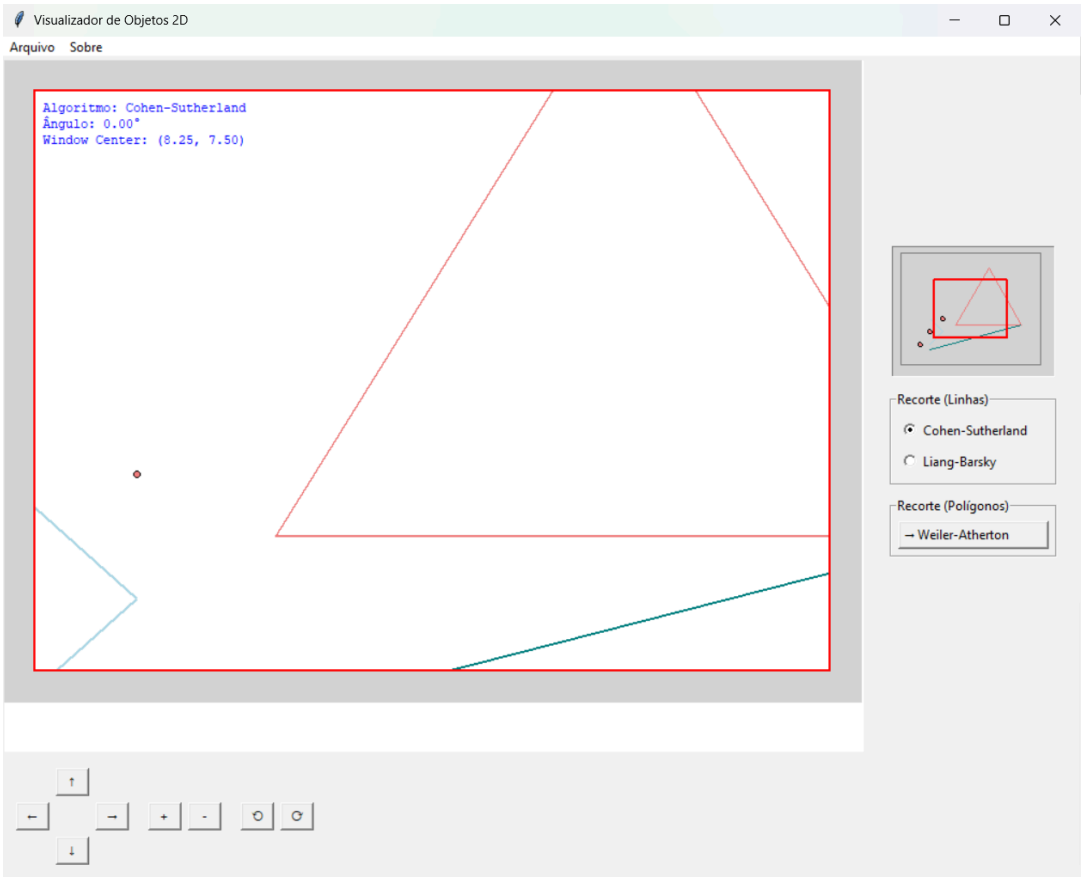
1. **Transformação Mundo para PPC:** As coordenadas de cada objeto são transformadas para um sistema onde a window está centralizada na origem e alinhada com os eixos. Esta etapa inclui a translação e a rotação inversa da window, simplificando os cálculos de recorte.
2. **Recorte (Clipping):** Com os objetos no PPC, os algoritmos de recorte são aplicados contra os limites da window (que no PPC são um retângulo alinhado aos eixos).
3. **Transformação PPC para Viewport:** Apenas os pontos dos objetos (ou segmentos de objetos) que sobreviveram ao recorte são mapeados para as coordenadas finais da viewport para serem desenhados.

4.1. Algoritmos de Recorte Implementados

- **Recorte de Pontos:** Um ponto é considerado visível se suas coordenadas no PPC estiverem dentro dos limites x e y da window.
- **Recorte de Retas (Cohen-Sutherland):** Este algoritmo associa um "código de região" de 4 bits a cada extremo da reta. Os códigos determinam se os pontos estão dentro, à esquerda, à direita, acima ou abaixo da window. O recorte é feito de forma iterativa, descartando trivialmente retas totalmente fora e aceitando trivialmente as que estão totalmente dentro. Para os demais casos, a interseção da reta com uma das bordas da window é calculada, e o processo se repete com o segmento menor.

- Recorte de Retas (Liang-Barsky): Um algoritmo mais eficiente que representa a reta de forma paramétrica. Ele calcula as interseções com as quatro bordas da window e determina se o segmento de reta visível existe. A interface permite ao usuário alternar entre este e o Cohen-Sutherland para fins de visualização e comparação.
- Recorte de Polígonos (Weiler-Atherton): Um algoritmo robusto que funciona para polígonos convexos e côncavos. Ele trata as listas de vértices do polígono e da window como estruturas de dados circulares. O algoritmo encontra todas as interseções entre as arestas de ambos, classificando-as como "de entrada" ou "de saída". Em seguida, ele atravessa as bordas, alternando entre as listas do polígono e da window a cada interseção para construir os novos polígonos resultantes do recorte.

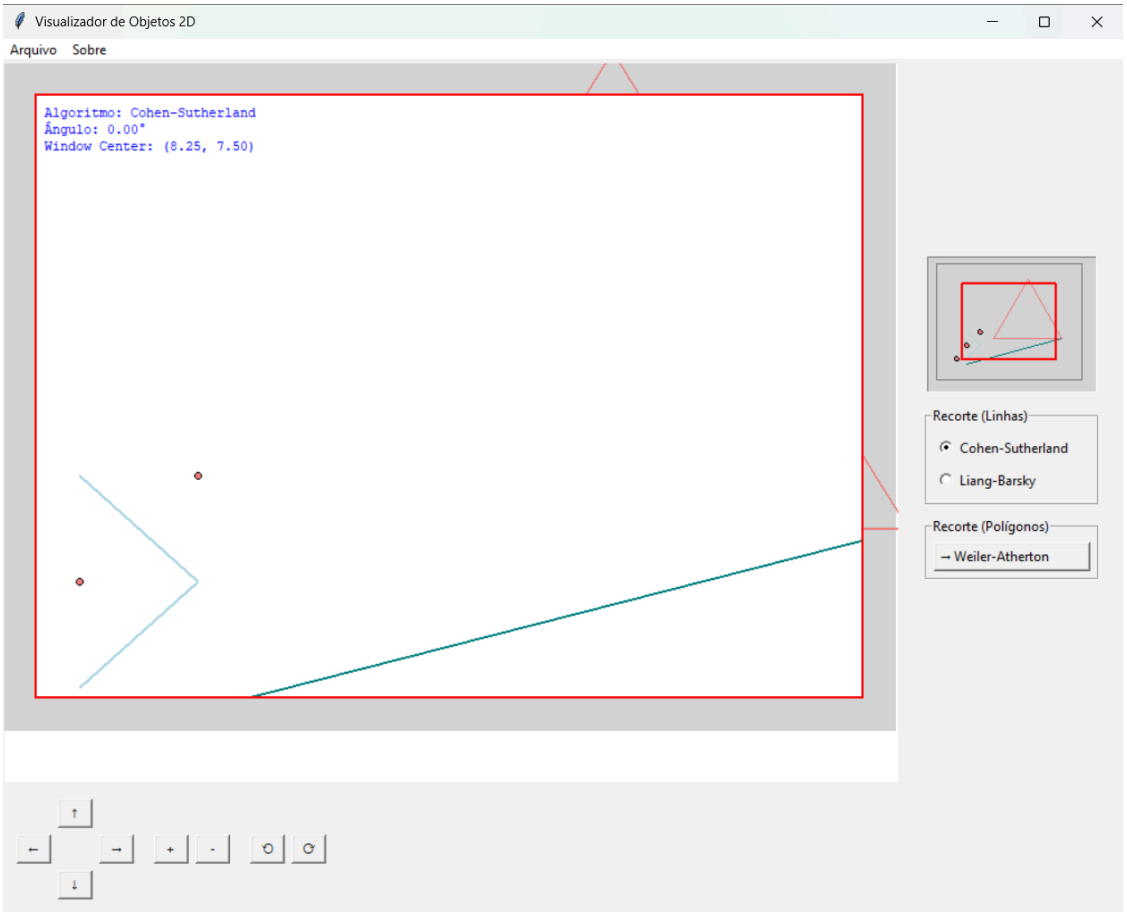
Assim, ao abrir o arquivo .xml, o limite da viewport fica preto, porém ao encontrar a alguma intercessão, a borda da viewport passa para a cor vermelha, gerando um resultado igual a imagem abaixo:



Para garantir o recorte dos polígonos, podemos ver o restante do polígono que ultrapassou a viewport, na *class Node*, função *def make_circular_nodes(pts, owner),* troca o sinal de < para > no *entering = prod < 0.*

```
# Calcula o vetor da aresta do polígono e o vetor normal da borda do clip
            vx, vy = (p2[0]-p1[0]), (p2[1]-p1[1])
            nx, ny = (q2[1]-q1[1]), (q1[0]-q2[0])
            prod = vx*nx + vy*ny
            entering = prod < 0 # troca entering = prod > 0 para teste
```

Gerando uma a saída somente com as partes do polígono fora da viewport:



5. Recursos Adicionais e Diferenças

Comparado ao primeiro trabalho, a versão atual representa um salto em funcionalidade e complexidade:

- **Interatividade Total:** O usuário não está mais restrito a mover a window, podendo também rotacioná-la e dar zoom, oferecendo uma exploração muito mais dinâmica e completa do "mundo".
- **Processamento Inteligente:** Em vez de transformar e tentar desenhar todos os objetos, o sistema agora recorta de forma inteligente, processando para a etapa final de renderização apenas os dados estritamente necessários.
- **Fidelidade Visual e de Dados:** O suporte a cores, lido do arquivo *entrada.xml*, permite cenas mais informativas. A estrutura de dados de cada objeto foi aprimorada para armazenar suas coordenadas em diferentes sistemas (mundo e PPC) e seu estado de visibilidade, conforme exigido.
- **Ferramenta de Depuração (DisplayList):** A nova janela "DisplayList" é um recurso poderoso que torna o pipeline de transformação transparente. Ao exibir as coordenadas de cada objeto no mundo, no PPC e na viewport, juntamente com o status de visibilidade, ela facilita a verificação e a compreensão de cada etapa do processo.

6. Conclusão

O desenvolvimento da segunda fase do "Visualizador de Objetos 2D" permitiu a aplicação prática de conceitos avançados de computação gráfica, como o pipeline de clipping e transformações geométricas complexas. Assim, o projeto cumpriu com sucesso todos os objetivos propostos, resultando em uma ferramenta interativa que não apenas renderiza cenas 2D, mas também demonstra visualmente o funcionamento e a importância dos algoritmos de recorte.

7. Git Hub

Repositório GitHub