Computação gráfica 1ªup

Prof.icaro ferreira

Ementa do curso

- Conceitos sobre Computação Gráfica.
- Computação Gráfica Bidimensional: primitivas 2D, atributos, transformações geométricas e animação.
- Computação Gráfica Tridimensional primitivas 3D, transformações espaciais, iluminação e animação.
- Introdução às subáreas da computação gráfica: síntese de imagens.
- Processamento de imagens e análise de imagens.

Ementa do curso (LIVROs)

Computer Graphics: Principles and Practice – John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, Kurt Akeley

Considerado a "bíblia" da computação gráfica, aborda desde conceitos básicos até técnicas avançadas de renderização, iluminação e animação.

Fundamentals of Computer Graphics – Peter Shirley, Michael Ashikhmin, Steve Marschner

Um livro acessível para iniciantes, cobrindo os princípios fundamentais de gráficos 2D e 3D.

Real-Time Rendering – Tomas Akenine-Möller, Eric Haines, Naty Hoffman

Foca em técnicas modernas para renderização em tempo real, essenciais para jogos e aplicações interativas.

Conceito de computação gráfica

- Computação Gráfica é uma área da computação que estuda a geração, manipulação e representação de imagens digitais.

- Ela é amplamente utilizada em jogos, simulações, animações, visualização científica, design gráfico, entre outras áreas.

Conceito de computação gráfica

1. Definição e Aplicações

Computação Gráfica envolve técnicas para representar imagens de forma eficiente e realista. Algumas de suas aplicações incluem:

- Jogos e entretenimento (renderização 3D em tempo real)
- Design gráfico e modelagem (Photoshop, Blender)
- Simulações médicas e científicas
- Realidade virtual e aumentada
- Interfaces gráficas (GUIs)

Conceito de computação gráfica

2. Representação de Imagens

As imagens digitais podem ser representadas de duas formas principais:

- **Rasterização (Bitmap):** Imagem formada por pixels, cada um com uma cor definida. Exemplo: PNG, JPEG.
- **Vetorial:** Imagem baseada em equações matemáticas, permitindo redimensionamento sem perda de qualidade. Exemplo: SVG.

Conceito de computação gráfica

3. Modelos de Cores

A cor é um elemento fundamental na Computação Gráfica, sendo representada por diferentes modelos:

- **RGB (Red, Green, Blue):** Usado em telas digitais, combina as cores para formar outras.
- **CMYK (Cyan, Magenta, Yellow, Black):** Utilizado em impressão.
- **HSV/HSL (Hue, Saturation, Value/Lightness):** Representação alternativa para facilitar manipulação de cores.

Conceito de computação gráfica

4. Primitivas Gráficas

Os gráficos são formados por primitivas básicas, como:

- Pontos
- Linhas
- Polígonos
- Curvas

Conceito de computação gráfica

4. Primitivas Gráficas

Os gráficos são formados por primitivas básicas, como:

- Pontos
- Linhas
- Polígonos
- Curvas

Conceito de computação gráfica

5. Transformações Geométricas

São operações usadas para manipular objetos gráficos:

- **Translação:** Move um objeto para uma nova posição.
- **Rotação:** Gira um objeto em torno de um ponto.
- **Escala:** Altera o tamanho do objeto.
- **Reflexão:** Espelha um objeto em relação a um eixo.

Conceito de computação gráfica

6. Renderização e Iluminação

A renderização é o processo de geração de imagens a partir de modelos 3D. Técnicas comuns incluem:

- **Ray Tracing:** Simula o caminho da luz para criar efeitos realistas.
- **Rasterização:** Método mais rápido, usado em jogos para renderização em tempo real.

Conceito de computação gráfica

7. Bibliotecas em Python para Computação Gráfica

Python oferece diversas bibliotecas para trabalhar com gráficos, como:

- **PIL/Pillow:** Manipulação de imagens raster.
- **Matplotlib:** Geração de gráficos 2D.
- **Pygame:** Desenvolvimento de jogos e gráficos 2D.
- **OpenCV:** Processamento de imagens e visão computacional.
- **PyOpenGL:** Interface para OpenGL, útil para gráficos 3D.

Conceito de computação gráfica

1. Primitivas Gráficas 2D

As primitivas gráficas são os elementos básicos para desenhar objetos em um espaço 2D. As principais primitivas incluem:

a) Ponto

A menor unidade gráfica, representada por coordenadas (x,y)(x, y)(x,y) em um plano cartesiano.

b) Linha

Segmento definido por dois pontos (x1,y1) e (x2,y2). Algoritmos comuns para desenhar linhas incluem:

- **Algoritmo de Bresenham:** Efetivo para desenhar linhas sem serrilhamento.
- **Equação da reta:** y=mx+b , ondem é a inclinação.

Conceito de computação gráfica

c) Polígonos

Formados por múltiplos segmentos de linha conectados. Exemplos: triângulos, quadrados, hexágonos.

d) Curvas

Podem ser definidas por funções matemáticas ou pontos de controle, como:

- **Curvas de Bézier:** Amplamente usadas para design gráfico e tipografia.
- **Splines:** Usadas para criar curvas suaves.

e) Círculos e Elipses

Desenhados com equações matemáticas ou algoritmos como:

- **Algoritmo de Bresenham para círculos**
- **Equação paramétrica do círculo:** $x=r\cos(t), y=r\sin(t)$

Conceito de computação gráfica

2. Atributos das Primitivas Gráficas

Cada primitiva pode ter atributos que afetam sua aparência, como:

a) Cor

Definida em modelos como RGB, CMYK ou HSV.

b) Espessura e Estilo de Linha

Linhas podem ser sólidas, tracejadas, pontilhadas, etc.

c) Preenchimento

Polígonos podem ser preenchidos com cores sólidas, gradientes ou texturas.

d) Transparência (Alpha)

Define a opacidade de um objeto (0 = totalmente transparente, 1 = opaco).

Conceito de computação gráfica

3. Transformações Geométricas

Permitem manipular a posição, tamanho e orientação dos objetos 2D.

a) Translação

Move um objeto de (x, y) para uma nova posição (x', y'):

$$x' = x + t_x$$

$$y' = y + t_y$$

Onde t_x e t_y são os deslocamentos nas direções x e y.

b) Escala

Ajusta o tamanho do objeto em relação a um ponto de referência:

$$x' = x \cdot s_x$$

$$y' = y \cdot s_y$$

Se $s_x, s_y > 1$, o objeto aumenta; se $0 < s_x, s_y < 1$, ele diminui.

Conceito de computação gráfica

3. Transformações Geométricas

Permitem manipular a posição, tamanho e orientação dos objetos 2D.

b) Escala

Ajusta o tamanho do objeto em relação a um ponto de referência:

$$x' = x \cdot s_x$$

$$y' = y \cdot s_y$$

Se $s_x, s_y > 1$, o objeto aumenta; se $0 < s_x, s_y < 1$, ele diminui.

c) Rotação

Gira um objeto em torno da origem por um ângulo θ:

$$x' = x\cos(\theta) - y\sin(\theta)$$

$$y' = x\sin(\theta) + y\cos(\theta)$$

Conceito de computação gráfica

3. Transformações Geométricas

Permitem manipular a posição, tamanho e orientação dos objetos 2D.

d) Reflexão

Espelha um objeto em relação a um eixo:

- Reflexão em x: $(x, y) \rightarrow (x, -y)$
- Reflexão em y: $(x, y) \rightarrow (-x, y)$

e) Cisalhamento

Deforma um objeto ao deslocar seus pontos proporcionalmente:

$$x' = x + sh_x \cdot y$$

$$y' = y + sh_y \cdot x$$

Aula 2 - Primitivas Gráficas 2D

- **Teoria**: Pontos, linhas, círculos, polígonos.
- **Prática**: Desenho de primitivas com Pygame.

Aula 2 - Primitivas Gráficas 2D

1. Introdução

- Definição de primitivas gráficas 2D
- Importância das primitivas na Computação Gráfica
- Exemplos de uso em aplicações reais

Aula 2 - Primitivas Gráficas 2D

2. Primitivas Gráficas Básicas

2.1. Ponto

- Representação de um ponto em um sistema de coordenadas
- Uso de pixels na tela

2.2. Linha

- Conceito de linha e sua representação digital
- Algoritmos de rasterização de linhas (Breseham, DDA)

Aula 2 - Primitivas Gráficas 2D

Primitivas gráficas 2D são formas geométricas básicas que servem como blocos de construção para a criação de imagens digitais. As principais primitivas incluem:

1. **Pontos**: Representam um único pixel na tela, com coordenadas (x, y).
2. **Linhas**: São definidas por dois pontos (início e fim) e podem ser retas ou curvas.

Aula 2 - Primitivas Gráficas 2D

2.3. Círculos e Elipses

- Definição matemática
- Algoritmo de Bresenham para círculos

2.4. Polígonos

- Representação de polígonos na tela
- Propriedades de polígonos convexos e côncavos

Aula 2 - Primitivas Gráficas 2D

1. **Polígonos**: Formas fechadas compostas por uma série de segmentos de linha conectados, como triângulos, retângulos e pentágonos.
2. **Círculos e Elipses**: Formas curvas definidas por um centro e um raio (no caso de círculos) ou por dois raios (no caso de elipses).

Aula 2 - Primitivas Gráficas 2D

1. **Arcos**: Segmentos de círculos ou elipses, definidos por um ângulo inicial e um ângulo final.
2. **Curvas**: Podem ser curvas de Bézier, splines, ou outras formas de curvas paramétricas.

Aula 2 - Primitivas Gráficas 2D

3. Representação Computacional

- Sistema de coordenadas na tela do computador
- Resolução e escala
- Conversão entre coordenadas matemáticas e coordenadas de tela

Aula 2 - Primitivas Gráficas 2D

Importância das Primitivas na Computação Gráfica

As primitivas gráficas são fundamentais na computação gráfica por várias razões:

1. **Base para Construção de Imagens**: Todas as imagens complexas são construídas a partir dessas formas básicas. Sem elas, seria impossível criar gráficos detalhados.

Aula 2 - Primitivas Gráficas 2D

1. **Design Gráfico**: Ferramentas de design, como Adobe Illustrator e CorelDRAW, usam primitivas para criar logos, ilustrações, e outros elementos visuais.
2. **Visualização Científica**: Gráficos de dados, como gráficos de barras, linhas e dispersão, são criados usando primitivas gráficas.

Aula 2 - Primitivas Gráficas 2D

1. **Sistemas CAD (Computer-Aided Design)**: Primitivas são usadas para desenhar modelos 2D de peças mecânicas, plantas arquitetônicas, e outros projetos técnicos.
2. **Interface do Usuário (UI)**: Botões, ícones, e outros elementos de interface são frequentemente renderizados usando primitivas gráficas.
3. **Simulações e Modelagem**: Em aplicações de simulação, como simulações de tráfego ou de fluidos, primitivas gráficas são usadas para representar objetos e fenômenos.

Aula 2 - Primitivas Gráficas 2D

1. vamos praticar, abram o vscode.
2. instalar https://stackoverflow.com/questions/54376745/how-to-import-pygame-in-visual-studio-code
3. vamos lá...

Aula 2 - Primitivas Gráficas 2D

Crie um programa em Pygame que desenhe uma cena simples contendo:

1. Um céu azul
2. Um sol representado por um círculo amarelo
3. Uma casa representada por um retângulo e um triângulo
4. Uma árvore representada por um retângulo e um círculo

Aula 3 - Transformações Geométricas 2D

1. Introdução

- Definição de transformações geométricas
- Importância das transformações na Computação Gráfica
- Aplicações em jogos, animações e design gráfico

Aula 3 - Transformações Geométricas 2D

1. Introdução

- Definição de transformações geométricas

  Transformações geométricas são operações matemáticas que alteram a posição, orientação, tamanho ou forma de objetos em um espaço bidimensional (2D) ou tridimensional (3D). Essas transformações são fundamentais para manipular objetos gráficos de maneira eficiente.

Aula 3 - Transformações Geométricas 2D

1. Introdução

- Importância das transformações na Computação Gráfica

  As transformações geométricas são essenciais na computação gráfica porque permitem a criação de cenas complexas a partir de objetos simples. Elas são usadas para mover, girar, escalar e distorcer objetos, o que é crucial em aplicações como jogos, animações e design gráfico.

Aula 3 - Transformações Geométricas 2D

1. Introdução

Aplicações em Jogos, Animações e Design Gráfico:

- **Jogos:** Movimentação de personagens, câmeras e objetos.
- **Animações:** Transições suaves entre cenas, movimentos de personagens.
- **Design Gráfico:** Manipulação de imagens, criação de layouts.

Aula 3 - Transformações Geométricas 2D

2. Tipos de Transformações

2.1. Translação

Definição e Matriz de Translação:

A translação move um objeto de uma posição para outra em um espaço 2D ou 3D. A matriz de translação para 2D é dada por:

$$T = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

Onde tx e ty são as distâncias de translação ao longo dos eixos X e Y, respectivamente.

Aula 3 - Transformações Geométricas 2D

2. Tipos de Transformações

Como Deslocar Objetos em um Espaço 2D:

Para deslocar um objeto, multiplicamos cada ponto do objeto pela matriz de translação.

Aula 3 - Transformações Geométricas 2D

2.2. Rotação

Rotação em Torno da Origem:

A rotação gira um objeto em torno da origem (0,0) por um ângulo θ. A matriz de rotação para 2D é:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Aula 3 - Transformações Geométricas 2D

2.2. Rotação

Rotação em Torno da Origem:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotação em Torno de um Ponto Arbitrário:

Para rotacionar em torno de um ponto arbitrário, primeiro translada-se o ponto para a origem, aplica-se a rotação e depois translada-se de volta.

Matriz de Rotação:

A matriz de rotação é usada para girar objetos em torno de um eixo.

UMJ
Centro Universitário Mario Pontes Jucá

Aula 3 - Transformações Geométricas 2D

2.2. Rotação

2.3. Escala

Definição e Matriz de Escala:

A escala altera o tamanho de um objeto. A matriz de escala para 2D é:

$$S = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Aula 3 - Transformações Geométricas 2D

2.2. Rotação

2.3. Escala

$$S = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Onde sx e sy são os fatores de escala ao long respectivamente.

Escalando Objetos Uniformemente e Não Uniformemente:

Uniformemente:

sx=sy

Não Uniformemente:

sx≠sy

UMJ
Centro Universitário Mario Pontes Jucá

Aula 3 - Transformações Geométricas 2D

3. Composição de Transformações

Aplicação de Múltiplas Transformações:

Múltiplas transformações podem ser aplicadas sequencialmente para alcançar efeitos complexos.

Ordem das Transformações e Seus Efeitos:

A ordem em que as transformações são aplicadas afeta o resultado final. Por exemplo, rotacionar e depois transladar produz um resultado diferente de transladar e depois rotacionar.

Aula 3 - Transformações Geométricas 2D

Uso de Multiplicação de Matrizes:

Para combinar transformações, multiplicamos as matrizes correspondentes. A ordem da multiplicação deve refletir a ordem desejada das transformações.

Aula 4 - Animação 2D

Introdução à Animação 2D

- O que é animação?
- Como a animação é usada em jogos, interfaces gráficas e simulações
- Diferença entre animação baseada em tempo e baseada em quadros

Aula 4 - Animação 2D

Introdução à Animação 2D

- O que é animação?

  Animação é a técnica de criar a ilusão de movimento através da exibição sequencial de imagens ou quadros. No contexto da computação gráfica, a animação 2D envolve o deslocamento de elementos gráficos em um espaço bidimensional.

Aula 4 - Animação 2D

Introdução à Animação 2D

Como a animação é usada em jogos, interfaces gráficas e simulações?

- **Jogos:** Animações são essenciais para personagens, efeitos visuais, movimentos de câmera e interações do usuário. Exemplo: personagens correndo ou pulando.
- **Interfaces gráficas (GUI):** Animações tornam a experiência mais fluida, como botões que mudam de cor ao serem clicados ou janelas que deslizam suavemente.
- **Simulações:** Usadas em aplicações científicas, médicas e de engenharia para representar fenômenos físicos, como fluidos em movimento ou simulações de tráfego.

Aula 4 - Animação 2D

Diferença entre animação baseada em tempo e baseada em quadros

- **Baseada em quadros:** A animação é atualizada a cada ciclo do jogo, sem considerar o tempo real. Se o jogo rodar a uma taxa de quadros diferente, a animação pode ficar mais rápida ou mais lenta.
- **Baseada em tempo:** O movimento dos objetos é ajustado com base no tempo decorrido. Isso garante que a animação ocorra na mesma velocidade, independentemente do FPS.

Aula 4 - Animação 2D

2. Fundamentos da Animação

2.1. Quadros por Segundo (FPS)

- Definição de FPS (Frames Per Second)
- Impacto do FPS na fluidez da animação
- Exemplo prático de diferentes taxas de FPS

Aula 4 - Animação 2D

Definição de FPS (Frames Per Second)
FPS (Frames Per Second) refere-se ao número de imagens ou quadros exibidos por segundo em uma animação.

Impacto do FPS na fluidez da animação

- Um FPS baixo (< 30) pode resultar em animações truncadas e menos suaves.
- Um FPS médio (30-60) proporciona uma experiência fluida na maioria dos casos.
- Um FPS alto (> 60) melhora a fluidez, mas pode exigir mais poder computacional.

Aula 4 - Animação 2D

Exemplo prático de diferentes taxas de FPS

- 15 FPS: Movimento perceptivelmente travado.
- 30 FPS: Movimento razoavelmente suave, comum em vídeos e jogos casuais.
- 60 FPS: Experiência fluida, comum em jogos modernos.
- 120+ FPS: Suavidade extrema, usada em monitores de alta taxa de atualização.

Aula 4 - Animação 2D

2.2. Técnicas de Animação

- Animação quadro a quadro
- Interpolação linear e suavização de movimento
- Uso de spritesheets

2.3. Movimentação e Física Básica

- Velocidade e aceleração
- Movimento retilíneo uniforme e uniformemente acelerado
- Detecção de colisões básica

Aula 4 - Animação 2D

Animação quadro a quadro

Cada quadro contém uma imagem ligeiramente diferente da anterior, criando a ilusão de movimento. É a técnica usada em desenhos animados clássicos e sprites de jogos 2D.

Interpolação linear e suavização de movimento

- **Interpolação linear:** Movimento entre dois pontos onde a posição muda uniformemente ao longo do tempo.
- **Suavização de movimento:** Técnicas como easing (aceleração/desaceleração) tornam a animação mais natural.

Uso de spritesheets

Spritesheets são imagens contendo múltiplos quadros de animação. Em vez de carregar várias imagens separadas, um jogo pode exibir diferentes partes do spritesheet para criar animação.

Aula 4 - Animação 2D

Velocidade e aceleração

- **Velocidade:** Determina o deslocamento de um objeto por unidade de tempo.
- **Aceleração:** Mede a taxa de variação da velocidade ao longo do tempo.

Movimento retilíneo uniforme e uniformemente acelerado

- **MRU (Movimento Retilíneo Uniforme):** O objeto se move em linha reta com velocidade constante. Exemplo: um personagem deslizando no gelo.
- **MRUA (Movimento Retilíneo Uniformemente Acelerado):** O objeto acelera ou desacelera de forma constante. Exemplo: um objeto caindo devido à gravidade.

Aula 4 - Animação 2D

Detecção de colisões básica

A detecção de colisões é usada para evitar que objetos atravessem uns aos outros. Métodos comuns incluem:

- **Colisão baseada em bounding box (caixa delimitadora):** Verifica se os retângulos ao redor dos objetos se sobrepõem.
- **Colisão pixel-perfect:** Verifica colisão exata com base nos pixels visíveis.

Aula 4 - Animação 2D

Exercício Prático

1. Modifique o programa para adicionar gravidade e fazer o círculo quicar no chão.
2. Adicione controle de velocidade usando as setas do teclado.
3. Utilize sprites para animar um personagem andando na tela.

Aula 5 - Visualização 2D

O que é visualização em computação gráfica?

Visualização é o processo de exibir imagens gráficas na tela de um computador a partir de dados matemáticos e geométricos. Em gráficos 2D, isso envolve a conversão de coordenadas do mundo para um espaço de exibição (viewport)

Aula 5 - Visualização 2D

Diferença entre coordenadas do mundo real e coordenadas da tela

- **Coordenadas do Mundo (World Coordinates):** Representam a posição de um objeto em um espaço abstrato de modelagem, podendo ser definidas em qualquer escala.
-
- **Coordenadas da Tela (Screen Coordinates):** Representam a posição do objeto na tela física (por exemplo, um monitor), onde o canto superior esquerdo normalmente é a origem (0,0).
-
- **Conversão:** Para exibir corretamente um objeto, é necessário transformar as coordenadas do mundo para coordenadas da tela usando a transformação **Window-to-Viewport**.

Aula 5 - Visualização 2D

Viewport e Window

Window (Janela de visualização)

A **window** é uma região no espaço do mundo que define o que será visível na tela. Tudo que estiver fora dessa região será descartado ou recortado.

Viewport

A **viewport** é a região na tela onde a **window** é mapeada. O tamanho da viewport pode ser ajustado para controlar a escala e posição da exibição.

Aula 5 - Visualização 2D

Transformação Window-to-Viewport

Essa transformação converte as coordenadas do mundo para coordenadas da tela, garantindo que um objeto seja corretamente posicionado na viewport.

Fórmula para conversão:

$$V_x = \left(\frac{X - X_{min}}{X_{max} - X_{min}} \right) \times (V_{maxX} - V_{minX}) + V_{minX}$$

$$V_y = \left(\frac{Y - Y_{min}}{Y_{max} - Y_{min}} \right) \times (V_{maxY} - V_{minY}) + V_{minY}$$

Aula 5 - Visualização 2D

Transformações de Visualização

Transformações de visualização são usadas para alterar a forma como os objetos aparecem na tela sem modificar suas propriedades fundamentais. Algumas das principais transformações são:

Translação

Desloca um objeto de um ponto a outro sem alterar sua escala ou orientação.

Fórmula:

$$X' = X + T_x, \quad Y' = Y + T_y$$

Onde T_x e T_y são os deslocamentos nas direções X e Y.

Aula 5 - Visualização 2D

Como ajustar a visualização sem alterar os objetos

Essas transformações podem ser aplicadas à **window** em vez dos objetos, permitindo zoom, rotação e movimentação da cena sem modificar os objetos originais.

Bora codar!!!!!!!!!!!!!!!!!!!

Aula 5 - Visualização 2D

Recorte (Clipping) de Primitivas Gráficas

O que é clipping?

O recorte (clipping) é a técnica usada para cortar partes de objetos que estão fora da área visível (window). Isso melhora a eficiência do processamento gráfico, garantindo que apenas elementos dentro da área visível sejam renderizados.

Aula 5 - Visualização 2D

Algoritmos de recorte

1. Cohen-Sutherland (para linhas)

Este algoritmo divide o espaço em 9 regiões e usa códigos binários para determinar quais partes de uma linha devem ser desenhadas.

Passos:

1. Determina se a linha está completamente dentro da window (totalmente visível).
2. Se estiver completamente fora, ela é descartada.
3. Se estiver parcialmente dentro, é recortada usando os limites da window.

Aula 5 - Visualização 2D

Algoritmos de recorte

2. Liang-Barsky (para linhas)

Esse algoritmo é mais eficiente do que Cohen-Sutherland porque calcula diretamente as interseções da linha com a window sem dividir o espaço em regiões.

3. Sutherland-Hodgman (para polígonos)

Um algoritmo para recorte de polígonos contra uma região retangular.

Aula 5 - Visualização 2D

Aplicação prática do recorte em gráficos 2D

O recorte pode ser implementado em Pygame ao verificar se um objeto está dentro da viewport antes de desenhá-lo.

```
def clip_rectangle(rect, clip_area):
    if rect.right < clip_area.left or rect.left > clip_area.right:
        return None  # Totalmente fora
    if rect.bottom < clip_area.top or rect.top > clip_area.bottom:
        return None  # Totalmente fora
    return rect.clip(clip_area)  # Retorna a parte visível
```

Aula 5 - Visualização 2D

https://docs.google.com/document/d/1qR4mrdPvbJf6Ueik2hFCBc8LWWqHAt8vl8jdFUySyAA/edit?usp=sharing

Aula 6 - Modelos de Cores e Transformação de Cores

1. Fundamentos das Cores

O que é cor e como a percebemos?

- A cor é uma percepção visual baseada na forma como nossos olhos e cérebro interpretam diferentes comprimentos de onda da luz.
- A luz branca é composta por diversas cores, como demonstrado no experimento de Newton com o prisma.

Aula 6 - Modelos de Cores e Transformação de Cores

Síntese Aditiva e Subtrativa de Cores

- **Síntese Aditiva (RGB - Red, Green, Blue)**:
    - Usada em monitores, telas e iluminação digital.
    - A combinação de todas as cores gera **branco**.
    - Exemplo: (255,0,0) = vermelho, (0,255,0) = verde, (0,0,255) = azul.
- **Síntese Subtrativa (CMY/CMYK - Cyan, Magenta, Yellow, Black)**:
    - Usada em impressoras e pigmentos.
    - A combinação de todas as cores gera **preto**.
    - Exemplo: (0,255,255) = azul, (255,0,255) = roxo.

Aula 6 - Modelos de Cores e Transformação de Cores

2. Modelos de Cores

Os modelos de cores ajudam a representar e manipular cores em computação gráfica.

- **RGB (Red, Green, Blue)**:
    - Modelo usado em telas e monitores.
    - Representado como (R, G, B) com valores entre 0 e 255.
- **CMY/CMYK (Cyan, Magenta, Yellow, Black)**:
    - Utilizado na impressão de imagens.
    - A conversão entre RGB e CMY ocorre subtraindo as cores de 255.

Aula 6 - Modelos de Cores e Transformação de Cores

HSV/HSB (Hue, Saturation, Value/Brightness):

- **Hue (Matiz):** Representa a cor principal (ex: vermelho, azul, verde).
- **Saturation (Saturação):** Define a intensidade da cor (de 0 a 100%).
- **Value/Brightness (Brilho):** Indica a claridade da cor.

HSL (Hue, Saturation, Lightness):

- Similar ao HSV, mas com um cálculo diferente para claridade.

YUV:

- Modelo usado para compressão de imagens e vídeos.
- Separa informações de **luminância (Y)** e crominância (**U e V**).

Aula 6 - Modelos de Cores e Transformação de Cores

Transformação entre Modelos de Cores (30 min)

Conversão RGB ↔ CMY

Fórmulas básicas:

$$C = 255 - R, \quad M = 255 - G, \quad Y = 255 - B$$

Conversão RGB ↔ HSV

1. Normaliza R, G e B para valores entre 0 e 1.
2. Calcula o **máximo e mínimo** dos valores.
3. Usa as fórmulas para encontrar **Matiz (H), Saturação (S) e Brilho (V)**.

Aula 6 - Modelos de Cores e Transformação de Cores

Bora ver na prática!!

Aula 7 - Representação e Modelagem de Objetos 2D

O que é modelagem em Computação Gráfica?

A **modelagem** em computação gráfica refere-se ao processo de criar e representar objetos visuais em um ambiente digital. Em um sistema computacional, a modelagem permite que objetos sejam definidos por pontos, linhas e superfícies matemáticas para serem manipulados, transformados e renderizados.

Aula 7 - Representação e Modelagem de Objetos 2D

Característica	Modelagem 2D	Modelagem 3D
Dimensões	Apenas largura e altura (X, Y)	Inclui profundidade (X, Y, Z)
Objetos	Linhas, polígonos, curvas	Malhas tridimensionais
Transformações	Rotação, translação, escala	Rotação, translação, escala + projeção
Aplicações	Design gráfico, jogos 2D, interfaces gráficas	Animação, CAD, renderizações fotorrealistas

Aula 7 - Representação e Modelagem de Objetos 2D

Como representar objetos 2D de forma eficiente?

A eficiência da modelagem 2D depende da representação usada. Alguns fatores incluem:

✅ **Uso de primitivas gráficas básicas** (pontos, linhas, polígonos) para reduzir cálculos.

✅ **Representação vetorial em vez de bitmap**, pois permite escalabilidade sem perda de qualidade.

✅ **Uso de coordenadas homogêneas** para facilitar transformações geométricas.

Aula 7 - Representação e Modelagem de Objetos 2D

Representação de Objetos 2D

Primitivas gráficas: pontos, linhas, polígonos

Os elementos básicos para modelagem 2D são:

- **Pontos (x, y)**: Coordenadas individuais.
- **Linhas**: Conexão entre dois pontos, representada por uma equação linear.
- **Polígonos**: Formas fechadas compostas por múltiplas linhas conectadas.

Aula 7 - Representação e Modelagem de Objetos 2D

Representação vetorial vs. representação matricial (bitmap)

- **Vetorial**: Define objetos como equações matemáticas (exemplo: SVG).
    - ✅ Escalável sem perder qualidade.
    - ✅ Ideal para gráficos e tipografia digital.
- **Matricial (Bitmap)**: Representa imagens como uma matriz de pixels (exemplo: PNG, JPG).
    - ❌ Perde qualidade ao redimensionar.
    - ✅ Melhor para fotos e texturas realistas.

Aula 7 - Representação e Modelagem de Objetos 2D

Estruturas de Dados para Modelagem 2D

Lista de vértices e arestas

Para modelar polígonos, utilizamos listas de **vértices (pontos)** conectados por **arestas (linhas)**. Exemplo:

poligono = [(100, 100), (200, 100), (200, 200), (100, 200)]

Essa estrutura permite flexibilidade na manipulação dos objetos.

Aula 7 - Representação e Modelagem de Objetos 2D

Estruturas de malha para polígonos

- **Malha de vértices**: Define objetos através de uma malha conectando vários pontos.

- **Lista de adjacências**: Estrutura que armazena conexões entre os vértices para rápida manipulação.

Hierarquia de objetos e agrupamento

Objetos podem ser organizados hierarquicamente para facilitar manipulações.
Exemplo: Um **robô 2D** pode ter seus braços e pernas como subobjetos do corpo, permitindo mover cada parte independentemente.

Aula 7 - Representação e Modelagem de Objetos 2D

Curvas e Superfícies em 2D

Curvas de Bézier e suas propriedades

- Criadas usando **pontos de controle** que definem a curvatura.
- Exemplo: Uma curva de Bézier quadrática usa 3 pontos (inicial, intermediário e final).
- Muito usada em **fontes vetoriais e gráficos interativos**.

A curva Bézier é definida pela equação:

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2$$

onde P0,P1,P2 são pontos de controle e t varia de 0 a 1.

Aula 7 - Representação e Modelagem de Objetos 2D

Curvas B-Spline e NURBS

- **B-Spline**: Suaviza curvas com mais pontos de controle, útil para modelagem complexa.

- **NURBS (Non-Uniform Rational B-Spline)**: Adiciona pesos aos pontos de controle, permitindo modelagem mais flexível.

Aplicação na construção de formas suaves e fontes vetoriais

Essas curvas são essenciais na modelagem de:

✅ **Fonte TrueType (TTF)**.
✅ **Ferramentas de design gráfico (Adobe Illustrator, Inkscape)**.
✅ **Modelagem de estradas e superfícies em CAD**.

UMJ
Centro Universitário Mario Pontes Jucá

Aula 7 - Representação e Modelagem de Objetos 2D

Transformações e Manipulação de Objetos 2D

Como modificar formas 2D com transformações geométricas

As transformações geométricas permitem manipular objetos sem alterar seus dados base.
Principais transformações:

- **Translação**: Move o objeto para outra posição.
- **Escalonamento**: Aumenta ou reduz o tamanho do objeto.
- **Rotação**: Gira o objeto em torno de um ponto.

Essas operações podem ser representadas com **matrizes de transformação**.

Aula 7 - Representação e Modelagem de Objetos 2D

$$\text{Matriz de translação} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{Matriz de rotação} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Aula 7 - Representação e Modelagem de Objetos 2D

Edição interativa e controle sobre vértices e arestas

Para editores gráficos, os objetos devem ser manipuláveis com:

✅ **Seleção e movimentação de pontos** com o mouse.

✅ **Escalonamento e rotação em tempo real**.

✅ **Edição direta de vértices e subdivisão de arestas**.

Aula 8 - Rasterização e Preenchimento de Polígonos

Introdução à Rasterização

O que é rasterização?

Rasterização é o processo de converter gráficos vetoriais (baseados em equações matemáticas) em uma imagem rasterizada composta por pixels. É um dos passos mais importantes da computação gráfica, pois define como os objetos serão desenhados na tela.

Quando desenhamos uma linha ou um polígono na tela, estamos **preenchendo pixels** de forma que representem essa forma geométrica corretamente.

Aula 8 - Rasterização e Preenchimento de Polígonos

Característica	Gráficos Vetoriais	Gráficos Rasterizados
Definição	Baseado em equações matemáticas (linhas, curvas).	Baseado em uma grade de pixels.
Escalabilidade	Pode ser redimensionado sem perder qualidade.	Perde qualidade ao ser ampliado.
Uso comum	Logos, fontes, ilustrações.	Fotos, texturas, telas de jogos.

Aula 8 - Rasterização e Preenchimento de Polígonos

O pipeline de rasterização em gráficos 2D

O pipeline de rasterização é o processo pelo qual um objeto 2D ou 3D é convertido em pixels. Ele segue estas etapas:

1. **Transformação geométrica** – Converte coordenadas do mundo real para coordenadas de tela.

2. **Clipping (Recorte)** – Objetos fora da tela são removidos para otimizar o desempenho.

3. **Rasterização** – Conversão dos objetos para pixels na tela.

4. **Interpolação de cores e shading** – Aplica cores, texturas e iluminação nos pixels.

Aula 8 - Rasterização e Preenchimento de Polígonos

Algoritmos de Rasterização de Linhas e Círculos

Algoritmo de Bresenham para linhas

O **algoritmo de Bresenham** é uma técnica eficiente para desenhar linhas usando apenas operações inteiras, evitando cálculos com números decimais.

Ele determina quais pixels devem ser ativados para aproximar ao máximo a reta ideal. Para isso, usa um erro acumulado para decidir se o próximo pixel será desenhado na mesma linha ou deslocado para a linha adjacente.

✅ **Vantagens:**

- Rápido e eficiente, pois usa apenas operações inteiras.
- Fácil de implementar em sistemas gráficos.

Aula 8 - Rasterização e Preenchimento de Polígonos

Algoritmo de rasterização de círculos (Midpoint Circle Algorithm)

Esse algoritmo, baseado no método do ponto médio, desenha um círculo a partir de um ponto central e um raio. Ele aproveita a **simetria do círculo** para calcular apenas 1/8 dos pontos e espelhar os outros.

Ele funciona avaliando a posição do ponto médio entre dois pixels candidatos e ajustando o próximo pixel com base nesse cálculo.

✅ **Vantagens:**

- Reduz cálculos, pois desenha apenas 1/8 do círculo e reflete os pontos.
- Usa apenas adições e subtrações, tornando-o eficiente.

Aula 8 - Rasterização e Preenchimento de Polígonos

Preenchimento de Polígonos

Preenchimento por varredura (Scanline Fill)

O **algoritmo Scanline Fill** percorre as linhas da tela **horizontalmente** e preenche os pixels dentro do polígono.

1. Identifica **arestas do polígono**.
2. Para cada linha horizontal, determina **quais segmentos devem ser preenchidos**.
3. Desenha os pixels entre as interseções com as arestas.

✅ **Vantagens:**

- Rápido e eficiente para polígonos grandes.
- Usa apenas operações básicas de comparação.

Aula 8 - Rasterização e Preenchimento de Polígonos

Preenchimento com algoritmo do pintor (Flood Fill)

O **Flood Fill** funciona como um "balde de tinta", preenchendo regiões conectadas com a mesma cor. Ele pode ser implementado de duas formas:

1. **Recursivo** – Preenche a região propagando para pixels vizinhos.

2. **Baseado em pilha (iterativo)** – Usa uma estrutura de pilha para evitar estouro de recursão.

Aula 8 - Rasterização e Preenchimento de Polígonos

✅ Vantagens:

- Funciona bem para áreas fechadas e irregulares.
- Fácil de entender e implementar.

🚨 Desafios:

- Pode ser ineficiente em polígonos grandes devido ao número de chamadas recursivas.
- Necessita de técnicas para evitar vazamentos no preenchimento (ex: preenchimento de borda).

Aula 8 - Rasterização e Preenchimento de Polígonos

Estratégias para evitar vazamentos no preenchimento

Quando usamos o Flood Fill, podem ocorrer falhas no preenchimento devido a bordas mal definidas ou erros na varredura de pixels. Algumas soluções incluem:

- **Definir corretamente a borda do polígono** para que todos os pixels internos sejam cobertos.

- **Usar um buffer de profundidade (Z-buffer)** para garantir que pixels sobrepostos sejam desenhados corretamente.

- **Implementar tolerância de cor** para evitar vazamentos devido a variações sutis na tonalidade.

Aula 8 - Rasterização e Preenchimento de Polígonos

Interpolação de Cores e Shading

Gouraud Shading e interpolação linear

O **Gouraud Shading** é uma técnica de suavização de cores para polígonos, calculando cores nos vértices e interpolando os valores ao longo da superfície.

1. **Define cores nos vértices do polígono**.

2. **Interpola os valores de cor ao longo das arestas**.

3. **Preenche o interior do polígono interpolando os valores das arestas**.

Aula 8 - Rasterização e Preenchimento de Polígonos

✅ Vantagens:

- Reduz o efeito de "degraus" na iluminação de polígonos.
- É computacionalmente barato.

🚨 Desvantagem:

- Não lida bem com reflexos especulares, pois a interpolação linear pode perder detalhes.

Aula 8 - Rasterização e Preenchimento de Polígonos

Phong Shading (introdução)

O **Phong Shading** é uma melhoria do Gouraud Shading que interpola **normais** em vez de cores. Isso resulta em uma iluminação mais precisa e realista.

Passos:

1. **Define normais nos vértices**.
2. **Interpola as normais ao longo da superfície**.
3. **Aplica o modelo de iluminação Phong para cada pixel**.

Aula 8 - Rasterização e Preenchimento de Polígonos

✅ **Vantagens:**

- Melhora a qualidade dos reflexos especulares.
- Produz superfícies suaves e realistas.

🚨 **Desvantagem:**

- Mais caro computacionalmente do que Gouraud Shading.

Aula 8 - Rasterização e Preenchimento de Polígonos

Aplicação prática de preenchimento de polígonos coloridos

Podemos usar **interpolação linear de cores** para criar efeitos visuais suaves em polígonos preenchidos.

📌 **Exemplo de interpolação de cores em um triângulo:**

- Definir cores diferentes para os três vértices.
- Interpolar os valores de cor ao longo das arestas.
- Preencher os pixels internos usando interpolação.

Obrigado!!!