UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ-UNIOESTE
Centro de Ciências Exatas e Tecnológicas –
CCET Curso de Bacharelado em Ciência da Computação

Matheus Bueno Pereira
Kauan Campos
Ryan hideky
Sammuel

Trabalho 2 CG: Preenchimento de Polígonos com Interpolação de Cores (Gouraud)

Prof. Adair Santa Catarina
Disciplina: Computação Gráfica

CASCAVEL – PR

1. Introdução

O código foi implementado em Python utilizando a biblioteca Pygame. O objetivo foi desenvolver um sistema de preenchimento de polígonos utilizando interpolação incremental de cores (Gouraud), aplicando o cálculo tanto entre arestas quanto entre pixels na mesma scanline. A aplicação também permite ao usuário interagir por meio de uma interface lateral para seleção de cores e controle do desenho dos vértices.

2. Explicação do código

```python
# --- Funções de pré-renderização da Interface ---
def create_color_wheel(radius):
    wheel_surface = pygame.Surface((radius * 2, radius * 2), pygame.SRCALPHA)
    for y in range(radius * 2):
        for x in range(radius * 2):
            dx, dy = x - radius, y - radius
            distance = math.hypot(dx, dy)
            if distance <= radius:
                angle = math.atan2(dy, dx)
                hue = (angle / (2 * math.pi)) % 1.0
                saturation = min(distance / radius, 1.0)
                rgb_normalized = colorsys.hsv_to_rgb(hue, saturation, 1.0)
                rgb_color = tuple(int(c * 255) for c in rgb_normalized)
                wheel_surface.set_at((x, y), rgb_color)
    return wheel_surface
```

A create_color_wheel(radius) gera uma imagem (pygame.Surface) que representa um *color wheel* (um seletor circular de cores onde a hue varia com o ângulo ao redor do centro e a saturação varia com a distância ao centro). Essa superfície é usada depois na UI para o usuário escolher tonalidade e saturação clicando no círculo.

```python
def create_brightness_slider(width, height):
    slider_surface = pygame.Surface((width, height))
    for y in range(height):
        value = 1.0 - (y / height)
        color = tuple(int(c * 255) for c in colorsys.hsv_to_rgb(0, 0, value))
        pygame.draw.line(slider_surface, color, (0, y), (width, y))
    return slider_surface
```

Essa função create_brightness_slider(width, height) cria uma **barra vertical de brilho** (brightness slider) que vai do branco (em cima) ao preto (embaixo).

Ela é usada junto com o *color wheel* para permitir ao usuário escolher o **Value** (V) do modelo de cores HSV.

```python
# --- Função de Preenchimento Incremental ---
def gouraud_fill_polygon(surface, vertices, vertex_colors):
    if len(vertices) < 3: return
    y_min = min(v[1] for v in vertices)
    y_max = max(v[1] for v in vertices)
    edge_table = [[] for _ in range(y_max + 1)]
    for i in range(len(vertices)):
        p1, c1 = vertices[i], vertex_colors[i]
        p2, c2 = vertices[(i + 1) % len(vertices)], vertex_colors[(i + 1) % len(vertices)]
        if p1[1] > p2[1]: p1, p2, c1, c2 = p2, p1, c2, c1
        if p1[1] == p2[1]: continue
        delta_y = p2[1] - p1[1]
        inv_slope = (p2[0] - p1[0]) / delta_y
        delta_color = [(c2[j] - c1[j]) / delta_y for j in range(3)]
        edge_table[p1[1]].append({
            "y_max": p2[1], "x": float(p1[0]), "inv_slope": inv_slope,
            "color": list(c1), "delta_color": delta_color
        })
    active_edges = []
    for y in range(y_min, y_max + 1):
        for edge in edge_table[y]:
            active_edges.append(edge)
        active_edges = [edge for edge in active_edges if edge["y_max"] > y]
        active_edges.sort(key=lambda edge: edge["x"])
        for i in range(0, len(active_edges), 2):
            if i + 1 < len(active_edges):
                e1, e2 = active_edges[i], active_edges[i+1]
                x_start, x_end = int(e1["x"]), int(e2["x"])
                color_start, color_end = e1["color"], e2["color"]
                if x_start >= x_end: continue
                delta_x = x_end - x_start
                span_delta_color = [(color_end[j] - color_start[j]) / delta_x for j in range(3)] if delta_x != 0 else [0,0,0]
                current_pixel_color = list(color_start)
                for x in range(x_start, x_end):
                    # **NOVO**: Verifica se o pixel está dentro do canvas
                    if UI_PANEL_WIDTH <= x < SCREEN_WIDTH:
                        surface.set_at((x, y), tuple(int(c) for c in current_pixel_color))
                    for j in range(3):
                        current_pixel_color[j] += span_delta_color[j]
        for edge in active_edges:
            edge["x"] += edge["inv_slope"]
            for j in range(3):
                edge["color"][j] += edge["delta_color"][j]
```

Essa função gouraud_fill_polygon(surface, vertices, vertex_colors) implementa o preenchimento de polígonos com interpolação de cores de Gouraud usando o método incremental, ou seja, calculando cores ponto a ponto enquanto percorre o polígono.

Logo no início, é feita uma verificação sobre a validade do polígono, ou seja, a função verifica se há pelo menos 3 vértices. Se não houver, retorna imediatamente, já que não é possível formar um polígono.

Ademais, é feita a determinação do intervalo vertical, são obtidos os valores mínimos (y_min) e máximos (y_max) das coordenadas y dos vértices, definindo o intervalo vertical que o algoritmo vai percorrer.

Cria-se uma lista de listas (edge_table) com tamanho y_max + 1, onde cada índice representa uma linha horizontal (scanline).

O loop que percorre os vértices monta as arestas:

- Para cada par de vértices consecutivos (p1, p2) e suas cores (c1, c2), garante que p1 seja o ponto mais alto (menor y).
- Se a aresta for horizontal (p1[1] == p2[1]), ela é ignorada.
- Calcula-se: delta_y: diferença vertical da aresta; inv_slope: inverso da inclinação, ou seja, quanto x deve mudar por unidade de y; delta_color: variação da cor por unidade vertical (para interpolação de cor ao longo da aresta).
- Essas informações são guardadas na edge_table na posição de p1[1], com: y_max: até onde essa aresta é válida, x: posição x inicial, color: cor inicial, delta_color: incremento da cor por passo vertical.

Essa etapa representa a **interpolação fora da scanline**, ou seja, calcula os

Incrementos que serão usados para atualizar posição e cor a cada linha.

Posteriormente é feita uma varredura por scanline, cria-se uma lista active_edges para armazenar as arestas ativas na linha atual.

Para cada linha y de y_min até y_max:

- Adiciona à lista active_edges todas as arestas que começam nessa linha (vindas da edge_table).
- Remove da lista as que já chegaram ao seu y_max.
- Ordena as arestas ativas pelo valor x (da esquerda para a direita).

Enfim, é feito a **interpolação dentro da scanline**, o preenchimento é feito de duas em duas arestas (e1 e e2), formando um segmento horizontal. Obtém-se x_start e x_end (limites horizontais) e as cores correspondentes (color_start e color_end). Calcula-se delta_x (comprimento do segmento) e span_delta_color, que é a variação de cor por pixel horizontal. Inicia-se current_pixel_color como a cor no ponto inicial, e para cada x até x_end:

- Se o pixel estiver dentro da área do canvas (UI_PANEL_WIDTH <= x < SCREEN_WIDTH), desenha-se (surface.set_at) com a cor atual.
- Incrementa-se cada componente da cor (R, G, B) usando span_delta_color.

Por fim, é feita uma atualização para a próxima linha, sendo assim, cada aresta ativa tem seu x atualizado somando inv_slope, e sua cor ajustada somando delta_color.

```python
def draw_text(surface, text, pos, font, color=WHITE, center=False):
    text_surface = font.render(text, True, color)
    if center:
        text_rect = text_surface.get_rect(center=pos)
        surface.blit(text_surface, text_rect)
    else:
        surface.blit(text_surface, pos)
```

A função draw_text desenha um texto na interface gráfica usando Pygame. Ela recebe a superfície onde o texto será desenhado, a string do texto, a posição, a fonte, a cor e uma opção para centralizar o texto na posição informada. Primeiro, a função cria uma imagem do texto com a cor especificada usando font.render(). Se a opção de centralizar estiver ativada, o texto é posicionado de forma que seu centro fique na posição dada; caso contrário, o texto é desenhado a partir do canto superior esquerdo na posição passada. Por fim, o texto é desenhado na superfície usando o método blit.

Em suma, ela centraliza a tarefa de renderizar texto na interface gráfica, posicionando o texto no local desejado e aplicando o alinhamento centralizado ou padrão

Finalmente, o objetivo é tratar sobre a função main:

```python
def main():
    pygame.init()
    screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
    pygame.display.set_caption("Criador de Polígonos Incremental com Interface Separada")
    clock = pygame.time.Clock()
    font = pygame.font.Font(None, FONT_SIZE)
    font_title = pygame.font.Font(None, 32)
    font_button = pygame.font.Font(None, 28)

    # Definição das áreas da tela
    canvas_rect = pygame.Rect(UI_PANEL_WIDTH, 0, SCREEN_WIDTH - UI_PANEL_WIDTH, SCREEN_HEIGHT)

    # Reposicionam (constant) UI_PANEL_WIDTH: Literal[280]  squerdo
    UI_CENTER_X = UI_PANEL_WIDTH // 2

    WHEEL_RADIUS = 85
    WHEEL_POS = (UI_CENTER_X, 120)
    color_wheel_surface = create_color_wheel(WHEEL_RADIUS)

    SLIDER_WIDTH, SLIDER_HEIGHT = 30, 150
    SLIDER_POS = (UI_CENTER_X - SLIDER_WIDTH // 2, WHEEL_POS[1] + WHEEL_RADIUS + 20)
    brightness_slider_surface = create_brightness_slider(SLIDER_WIDTH, SLIDER_HEIGHT)
    slider_rect = pygame.Rect(SLIDER_POS, (SLIDER_WIDTH, SLIDER_HEIGHT))

    PREVIEW_SIZE = 80
    PREVIEW_POS = (UI_CENTER_X - PREVIEW_SIZE // 2, SLIDER_POS[1] + SLIDER_HEIGHT + 20)
    preview_rect = pygame.Rect(PREVIEW_POS, (PREVIEW_SIZE, PREVIEW_SIZE))

    reset_button_rect = pygame.Rect(0, 0, 150, 50)
    reset_button_rect.center = (UI_CENTER_X, SCREEN_HEIGHT - 60)

    # --- Variáveis de estado ---
    vertices, vertex_colors = [], []
    is_polygon_closed = False
    current_hsv = (0.0, 1.0, 1.0)
    dragging_wheel, dragging_slider = False, False
```


Você pode colar a imagem a partir da área de transferência.


```python
def main():
    running = True
    while running:
        rgb_normalized = colorsys.hsv_to_rgb(*current_hsv)
        current_color = tuple(int(c * 255) for c in rgb_normalized)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

            mx, my = pygame.mouse.get_pos()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                    if reset_button_rect.collidepoint(mx, my):
                        vertices.clear(); vertex_colors.clear(); is_polygon_closed = False
                    elif math.hypot(mx - WHEEL_POS[0], my - WHEEL_POS[1]) <= WHEEL_RADIUS:
                        dragging_wheel = True
                    elif slider_rect.collidepoint(mx, my):
                        dragging_slider = True
                    # Adicionar vértice somente se o clique for DENTRO do canvas
                    elif canvas_rect.collidepoint(mx, my) and not is_polygon_closed:
                        vertices.append(event.pos)
                        vertex_colors.append(current_color)

                elif event.button == 3 and canvas_rect.collidepoint(mx, my):
                    if not is_polygon_closed and len(vertices) >= 3:
                        is_polygon_closed = True

            if event.type == pygame.MOUSEBUTTONUP and event.button == 1:
                dragging_wheel, dragging_slider = False, False

            if event.type == pygame.MOUSEMOTION:
                if dragging_wheel:
                    dx, dy = mx - WHEEL_POS[0], my - WHEEL_POS[1]
                    dist = math.hypot(dx, dy)
                    hue = (math.atan2(dy, dx (constant) WHEEL_RADIUS: Literal[85]
                    saturation = min(dist / WHEEL_RADIUS, 1.0)
                    current_hsv = (hue, saturation, current_hsv[2])

                if dragging_slider:
                    value = 1.0 - (my - slider_rect.top) / slider_rect.height
                    value = max(0.0, min(1.0, value))
                    current_hsv = (current_hsv[0], current_hsv[1], value)

        # --- Lógica de Desenho ---
        screen.fill(UI_PANEL_COLOR)
        pygame.draw.rect(screen, CANVAS_COLOR, canvas_rect)
```

```
    # Desenha Polígono no Canvas
    if len(vertices) > 0:
        if is_polygon_closed:
            gouraud_fill_polygon(screen, vertices, vertex_colors)
        if len(vertices) > 1:
            pygame.draw.lines(screen, WHITE, is_polygon_closed, vertices, 2)
        for i, vertex in enumerate(vertices):
            pygame.draw.circle(screen, vertex_colors[i], vertex, 7)
            pygame.draw.circle(screen, BLACK, vertex, 7, 1)

    # Desenha a UI no Painel
    draw_text(screen, "Seletor de Cores", (UI_CENTER_X, 20), font_title, WHITE, center=True)
    screen.blit(color_wheel_surface, (WHEEL_POS[0] - WHEEL_RADIUS, WHEEL_POS[1] - WHEEL_RADIUS))
    screen.blit(brightness_slider_surface, SLIDER_POS)
    h, s, v = current_hsv
    angle = h * 2 * math.pi
    indicator_x = WHEEL_POS[0] + math.cos(angle) * s * WHEEL_RADIUS
    indicator_y = WHEEL_POS[1] + math.sin(angle) * s * WHEEL_RADIUS
    pygame.draw.circle(screen, WHITE, (indicator_x, indicator_y), 5, 2)
    slider_indicator_y = slider_rect.top + (1.0 - v) * slider_rect.height
    pygame.draw.line(screen, WHITE, (slider_rect.left, slider_indicator_y), (slider_rect.right, slider_indicator_y), 3)
    pygame.draw.rect(screen, current_color, preview_rect)
    pygame.draw.rect(screen, WHITE, preview_rect, 2)

    # Desenha o Botão de Reset no Painel
    pygame.draw.rect(screen, RED_BUTTON, reset_button_rect, border_radius=12)
    pygame.draw.rect(screen, WHITE, reset_button_rect, 2, border_radius=12)
    draw_text(screen, "Resetar", reset_button_rect.center, font_button, WHITE, center=True)

    # Desenha Instruções no Painel
    inst_y_start = preview_rect.bottom + 40
    draw_text(screen, "Instruções:", (UI_CENTER_X, inst_y_start), font_title, WHITE, center=True)
    draw_text(screen, "1. Escolha a cor no seletor.", (20, inst_y_start + 40), font)
    draw_text(screen, "2. Clique esquerdo no quadro", (20, inst_y_start + 65), font)
    draw_text(screen, "   para adicionar um vértice.", (20, inst_y_start + 85), font)
    draw_text(screen, "3. Clique direito no quadro", (20, inst_y_start + 110), font)
    draw_text(screen, "   para fechar o polígono.", (20, inst_y_start + 130), font)

    pygame.display.flip()
    clock.tick(FPS)

pygame.quit()
```

A função main é o núcleo da aplicação gráfica em Pygame que permite ao usuário criar polígonos coloridos incrementalmente em uma área de desenho.

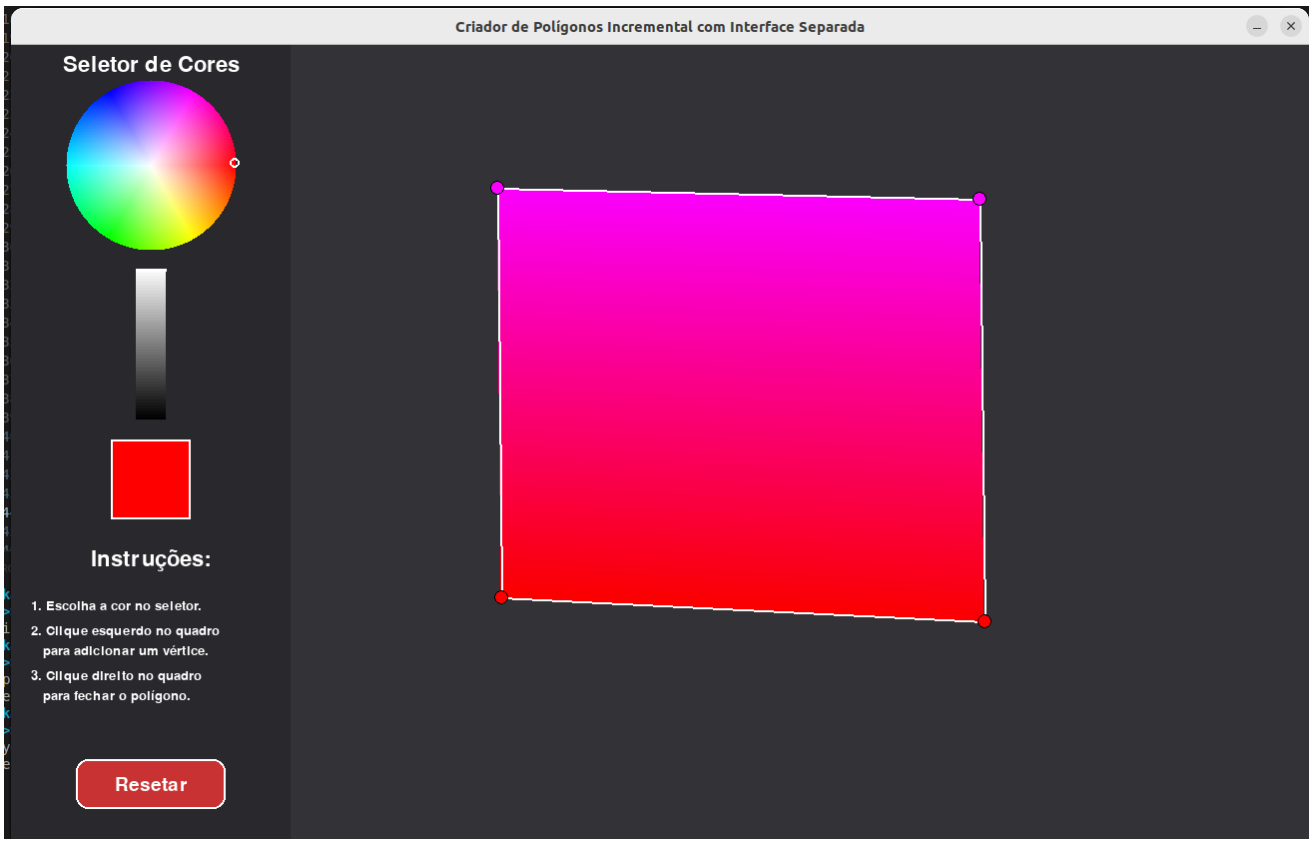
Em primeiro lugar, Inicializa o Pygame, configura a janela, define fontes para texto, e define as áreas principais: o painel de controle e o canvas (área de desenho) e cria superfícies pré-renderizadas para o seletor de cor.

Posteriormente, define variáveis de estado, cria listar para os vértices e cores associadas a cada um, controla se o polígono está ou não fechado, current_hsv mantém a cor atual selecionada pelo usuário no espaço HSV para atualizar conforme ele interage com o seletor e dragginf_wheel e dragging_slider controlam se o usuário está interagindo com o seletor de cor, arrastando na roda ou no slider.

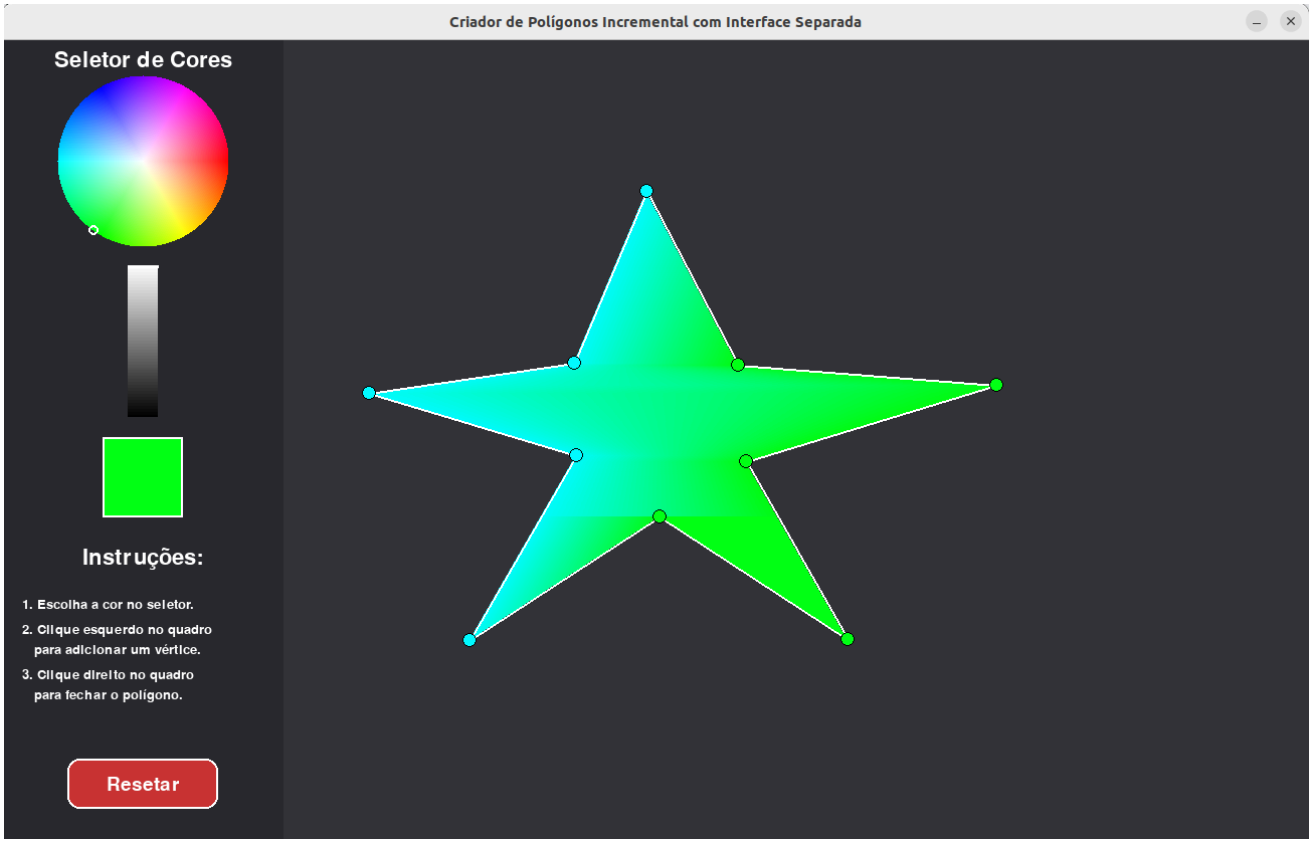
O loop principal controla e processa os eventos que ocorrem na interface gráfica, quando o polígono fecha ele chama a função gouraud_fill_polygon.

3. Exemplos de utilização

Polígono de 4 vértices, dois de cor rosa e dois de cor vermelha:



Polígono com 10 vértices, à esquerda ciano, e à direita verde:



Polígono com 9 faces, 4 brancas e 5 pretas.