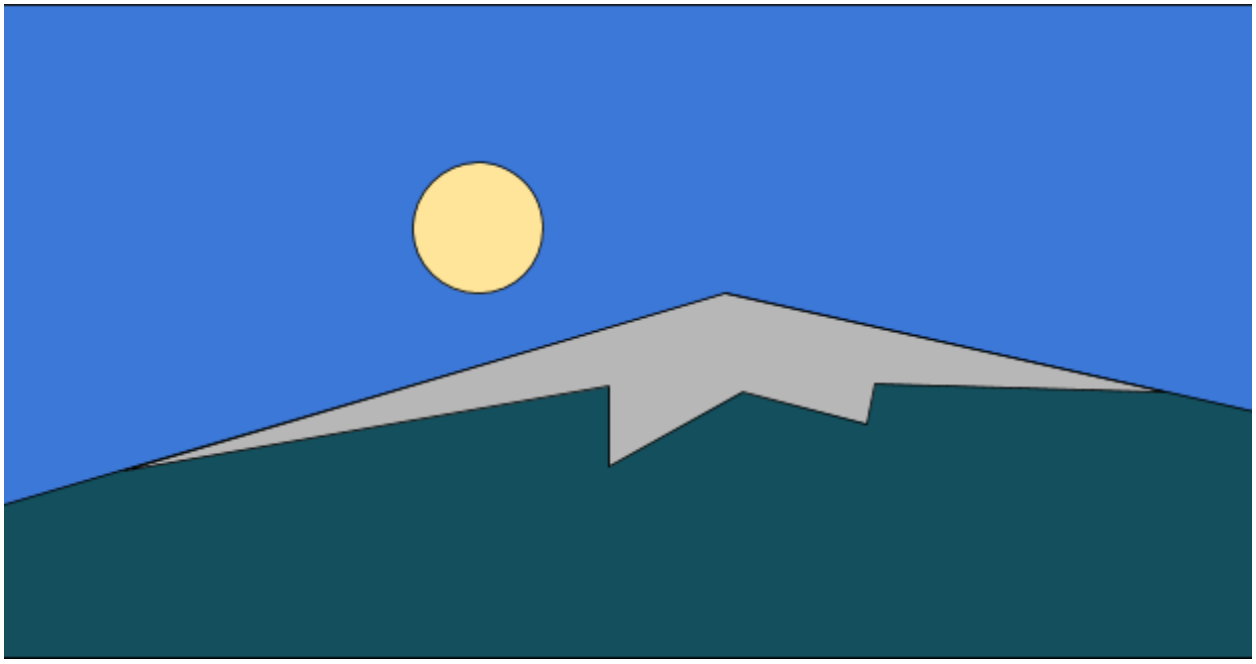
Introdução a Técnicas de Programação - 2019.2

Descrição de projeto

Desenho gráfico



Introdução

Há muitos programas de edição de imagens ou desenho vetorial em que você pode criar sua própria arte digital. Neste projeto, você irá criar seu próprio programa de desenho, a partir do qual os usuários poderão criar figuras como a apresentada acima. Para isso, será necessário aprender como imagens são representadas e alguns algoritmos de desenho de primitivas gráficas, como linhas, polígonos e círculos.

Representações de imagens

Imagens digitais são normalmente representadas de duas formas: vetorial ou *raster*. A primeira usa figuras geométricas, como polígonos, circunferências entre outros, para definir objetos vetoriais a serem usados na composição da imagem. A segunda forma é uma representação em que há uma correspondência de cada pixel da imagem a um conjunto de informações (como cor, transparência etc). Dizemos que esta última forma é uma representação “ponto-a-ponto” da imagem, normalmente realizada através de uma matriz onde cada célula contém a informação de um ponto, também conhecida como “mapa de bits” ou *bitmap*.

O tamanho da matriz define a resolução da imagem, normalmente especificada através da quantidade de colunas e de linhas. Por exemplo, a resolução 640 x 480 indica uma imagem

cuja matriz possui 640 colunas e 480 linhas, ou seja que contém 307.200 pontos de informação.

Apesar do presente projeto ter como objetivo o desenho de figuras geométricas como polígonos, seu programa irá trabalhar com imagens do tipo *raster*. Ele deverá transformar informações de linhas e polígonos em pixels, em um processo que chamamos de *rasterização*. A matriz de pixels da imagem será, portanto, alterada em função das primitivas de desenho que forem aplicadas.

Quando desejamos armazenar, transferir ou imprimir a imagem, faz-se necessário armazená-la em um arquivo. A representação da imagem no arquivo não utiliza necessariamente a mesma representação da imagem em memória. Enquanto esta procura facilitar e tornar as operações de manipulação da imagem eficientes, aquela é normalmente voltada à compactação da imagem (tamanho do arquivo) com ou sem perdas de qualidade.

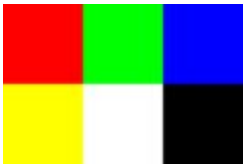
Sua representação na memória do computador é normalmente efetuado através de matrizes de pontos de informação, mas a representação em arquivo pode seguir diferentes formatos. Os formatos mais simples são os definidos pelo pacote NetPBM, encontrados nos ambientes Linux. O pacote NetPBM define alguns formatos, como o PBM (Portable Bitmap), PGM (Portable Graymap) e PPM (Portable Pixmap). Este último será utilizado no presente projeto.

Formato de imagem PPM

O formato de imagem PPM possui duas versões: uma textual e outra binária. A versão textual armazena os valores de cada pixel em um arquivo ASCII. Cada pixel é definido por três componentes, indicando a intensidade das cores vermelho, verde e azul. Ou seja, a cor do pixel é resultante da combinação das cores vermelho, verde e azul. Para entender como funciona essa combinação, você pode acessar http://www.rapidtables.com/web/color/RGB_Color.htm e variar os campos R (Red), G (Green) e B (Blue). A cor preta, por exemplo, é representada por R = 0, G = 0 e B = 0. Já o amarelo é representado por R = 255, G = 255 e B = 0.

O arquivo PPM obedece a uma formação específica composta por um cabeçalho, com informações sobre a imagem em geral, seguido de uma sequência de triplas RGB para cada um dos pixels. A função do cabeçalho é identificar que se trata de uma imagem PPM e de sua versão, as dimensões da imagem e o valor máximo de cada cor.

Para ilustrar melhor, a figura e conteúdo a seguir exemplificam uma imagem (ampliada para ser melhor exibida) com dimensões de 3x2 representada no formato PPM, versão textual.



```
P3
3 2
255
255 0 0
0 255 0
0 0 255
255 255 0
255 255 255
0 0 0
```

A primeira linha contém o identificador `P3`. Esse identificador serve para os programas que leem arquivos de imagem saber que o conteúdo a seguir adotam o formato textual do PPM. Em seguida, os valores 3 e 2, separados por espaço, indicam que a imagem tem 3 colunas e 2 linhas. Na terceira linha está o valor 255. Esse valor é usado para informar o valor máximo de cada componente do pixel e indica, de certa forma, o nível de qualidade da imagem. Valores baixos representam imagens com pouca qualidade e valores maiores representam maior qualidade. Por exemplo, se esse valor máximo for 4, ou seja cada componente pode assumir 5 valores (intervalo de 0 a 4), significa que a imagem pode ter no máximo 125 cores diferentes (5x5x5). Se o valor for 255, ou seja cada componente pode assumir 256 valores, teremos 16.777.216 possíveis cores para representar cada pixel.

As demais linhas do arquivo se referem aos componentes RGB dos pixels da imagem. A ordem em que os pixels se encontram seguem linha a linha. Ou seja, em uma imagem de 3 colunas e 2 linhas, os três primeiros pixels correspondem à primeira linha e os três seguintes à segunda linha.

Neste trabalho, considere os arquivos de imagem no formato PPM versão textual (P3) com o valor de qualidade fixo em 255 (cada componente pode variar de 0 a 255). As dimensões podem, entretanto, variar.

Primitivas Gráficas

Primitivas gráficas são funções ou procedimento que desenham objetos geométricos simples, que não são normalmente decompostos por outras primitivas. Primitivas comuns em programas de desenho são as rotinas para o desenho de pontos, retas, polígonos, círculos, elipses, entre outros. A partir dessas primitivas, podemos elaborar figuras mais complexas. Nas seções a seguir, apresentamos algumas dessas primitivas.

Cor atual

Embora não seja uma primitiva gráfica a cor atual das operações é um parâmetro comum quando lidamos com desenho e primitivas gráficas. A ideia principal é que o processador gráfico usa a última cor que foi configurada para todas as operações que seguem. Assim as primitivas de desenho como desenho de linhas, curvas e outras figuras não precisam receber cores como argumento. Por exemplo, se definirmos a cor atual como “azul”, as próximas primitivas gráficas (desenho de retas ou figuras geométricas) serão desenhadas com a cor azul, até que se mude a cor atual.

Desenho de retas

O problema de desenho de retas é identificar quais pixels devem ser alterados no percurso do ponto inicial da reta ao ponto final. Por exemplo, na Figura 1 a reta a ser desenhada entre dois pontos (do verde ao laranja) passa por vários pixels. Se alterarmos todos os pixels pelos quais a reta passa, o resultado seria uma reta mais espessa do que deveria ser. Além disso, como saber por quais pixels a reta passaria? Uma solução é usar o ângulo de inclinação da reta e através de cálculos com o seno e cosseno desse ângulo saber os pixels ao longo da reta. Porém, o cálculo do seno e cosseno de um ângulo é custoso para um procedimento que será efetuado centenas ou milhares de vezes. É necessário algo mais eficiente.

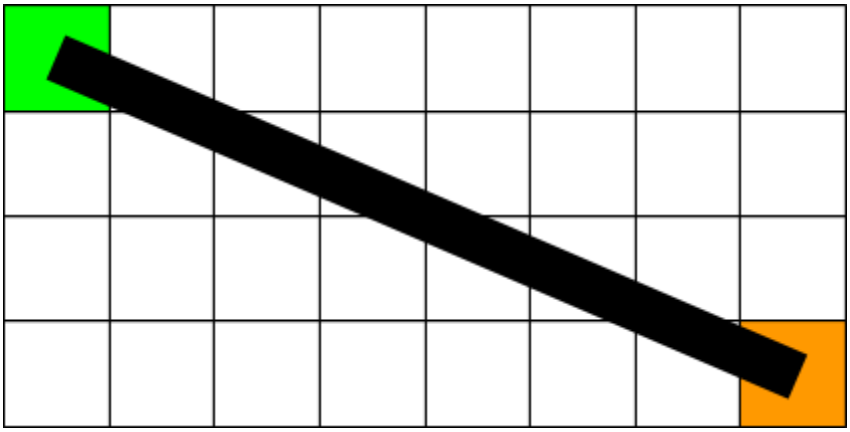


Figura 1 - Reta a ser desenhada do ponto inicial (verde) ao ponto final (laranja).

Em 1962, Jack Bresenham desenvolveu um algoritmo capaz de resolver as limitações descritas anteriormente. Seu algoritmo é tão eficiente que não usa sequer operações de multiplicação, divisão ou arredondamento. No seu laço, utiliza-se apenas adições. Ele parte da ideia que os pixels a serem alterados sempre “andam” uma casa no eixo X, no eixo Y ou em ambos. A questão é saber quando deve-se “andar” em cada um dos eixos.

Para simplificar, ele dividiu a área de desenho em oito regiões similares (octantes), como ilustrado na Figura 2. Basta definir uma solução para um dos octantes que ela pode ser replicada para os demais. Por exemplo, sabendo a solução para traçar uma reta do centro da

Figura 2 ao ponto O1, é possível traçar uma solução até o ponto O8, basta inverter o sinal do deslocamento dos pixels no eixo Y, ao invés de ir somando, vai subtraindo. O mesmo ocorre para traçar uma solução até o ponto O2, basta trocar o eixo X pelo Y. Ou seja, o problema de traçar qualquer reta no plano pode ser simplificado no problema de traçar uma reta em apenas um dos octantes. Considerando que estamos lidando com uma matriz de pixels, cuja coordenada (0,0) encontra-se no canto superior esquerdo, vamos trabalhar com o octante O8, pois o deslocamento do tracejado da reta irá usar apenas operações de soma.

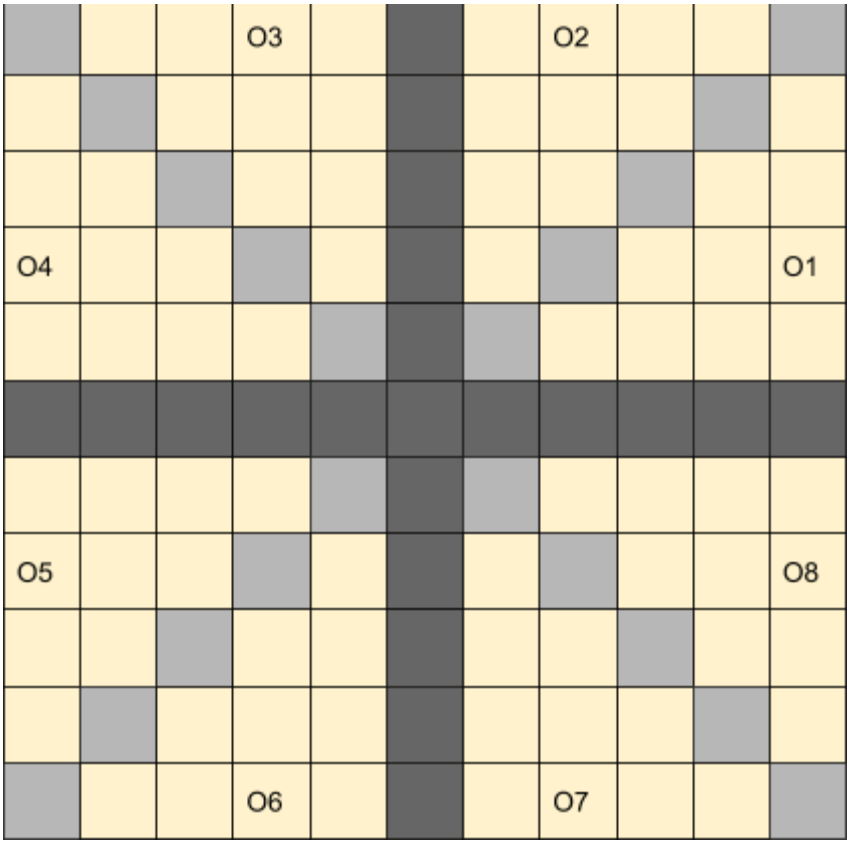


Figura 2 - Octantes do plano representando as oitos regiões similares do algoritmo.

A Figura 3 ilustra um tracejado no octante O8, ligando o ponto verde ao laranja. É possível perceber que o tracejado sempre se desloca uma casa no eixo X. Ou seja, para cada coordenada X, há apenas um pixel alterado. Isso significa que em um laço a coordenada X do pixel a ser alterado irá incrementar a cada iteração. Por outro lado, no eixo Y, é possível que haja mais de um pixel alterado por coordenada. Ou seja, nem sempre a coordenada Y do pixel a ser alterado é incrementada. Resumindo, o deslocamento no eixo Y será menor que no eixo X, e seu incremento vai depender de quão inclinado a reta se encontra.

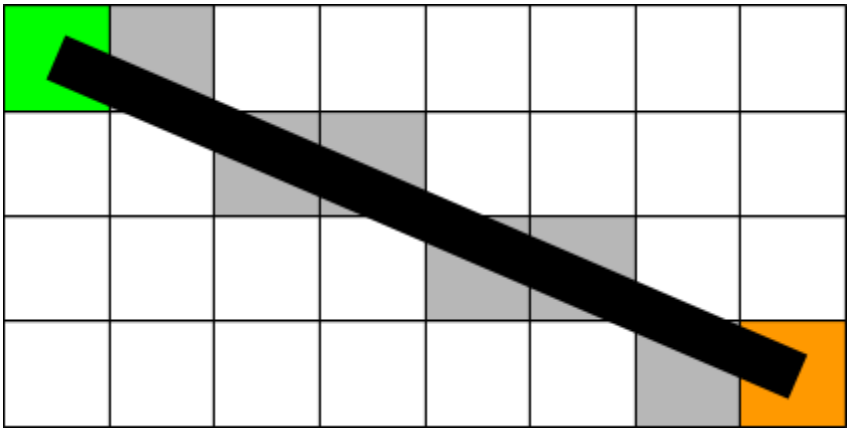


Figura 3 - Reta a ser desenhada do ponto inicial (verde) ao ponto final (laranja).

Digamos que reta a ser tracejada seja do ponto (x1, y1) ao ponto (x2, y2), a inclinação pode ser calculada pela razão da diferença de suas coordenadas, ou seja:

$$\frac{y2 - y1}{x2 - x1}$$

Esse valor de inclinação indica o deslocamento a ser executado no eixo Y a cada iteração do laço. Como no octante O8 a diferença das coordenadas X dos pontos é maior (ou igual) a diferença das coordenadas Y, o valor do deslocamento será sempre menor (ou igual) a 1. A cada iteração, esse valor vai sendo acumulado até o momento que for maior ou igual a 1. Quando isso ocorre, significa que a reta já passou para o próximo pixel no eixo Y e, portanto, a coordenada Y deve ser incrementada. Esse processo iterativo continua até que o pixel a ser alterado seja o pixel destino (x2,y2). Nesse momento, a reta foi toda desenhada.

Algoritmo de desenho de círculos

O algoritmo de desenho de retas de Bresenham foi adaptado para várias situações, inclusive para desenhar círculos. A ideia de dividir em octantes é a mesma. Ou seja, basta definirmos uma solução para desenhar um arco em um dos octantes que teremos uma solução para desenhar todo o círculo. A diferença entre ambos é que no tracejado da reta sabíamos que o eixo X iria sempre se deslocar a cada iteração e no do círculo é possível que haja deslocamento tanto no eixo X quanto no Y, mas não em ambos.

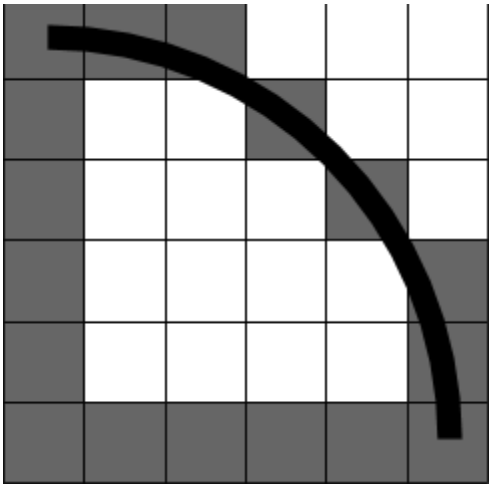


Figura 4 - Arco a ser desenhado pelo algoritmo de círculo de Bresenham

Desenho de polígonos

Um polígono é uma figura **fechada** composta por um conjunto de pontos ligados por um conjunto de segmentos de **retas que não se intersectam**. Todas as figuras geométricas fechadas em 2D compostas por segmentos de reta que não se intersectam são polígonos. Alguns exemplos mais conhecidos são retângulos, triângulos, trapézios e paralelogramos. A Figura 5 ilustra alguns exemplos.

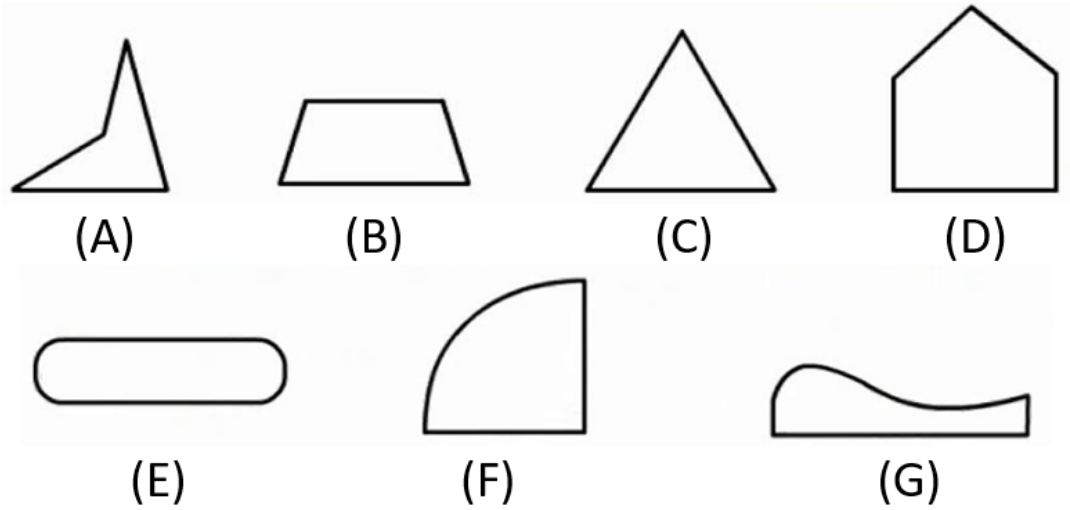


Figura 5 - Figuras de A à D são polígonos, enquanto as demais não são

No entanto, como mostra a Figura 5, polígonos podem ter diversas formas além de um número infinito de lados e pontos. Seja V um vetor com os pontos de um polígono, um algoritmo para desenhá-lo pode ser descrito como visto no fluxograma na Figura 6. Observe que, para que o processo funcione, o primeiro ponto em V precisa ser igual ao último, garantindo assim o fechamento da figura.

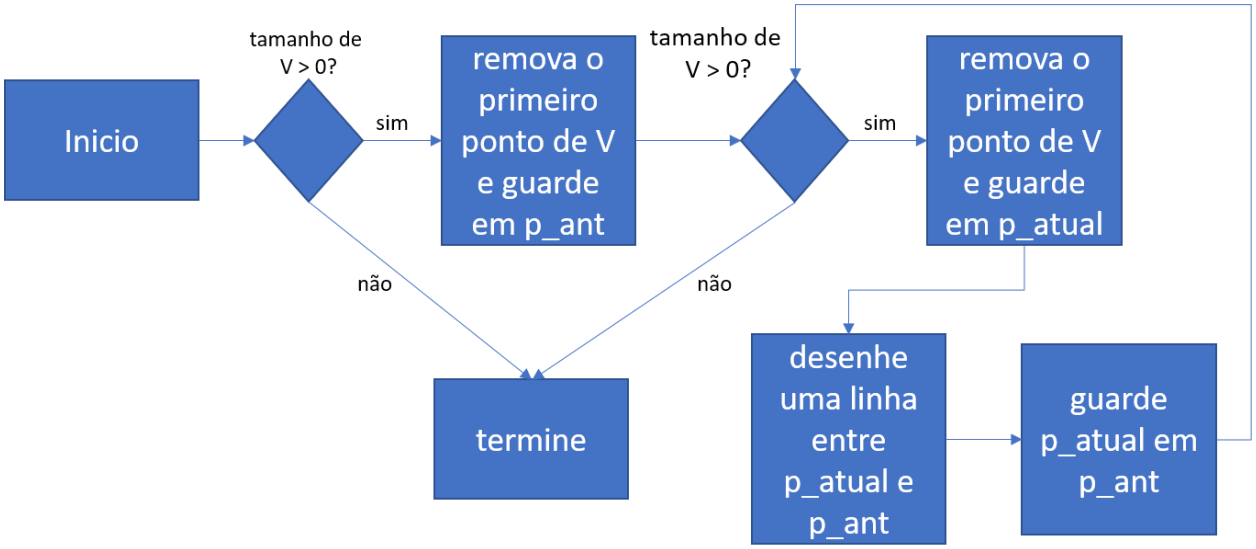


Figura 6 - Fluxograma desenho de polígonos

Observe que o algoritmo da Figura 6 apenas descreve o procedimento de desenho. Não existem checagens que garantam que o conjunto de pontos contidos em V descrevem realmente um polígono.

Preenchimento de figuras fechadas

Assim como a configuração da cor atual, outra operação comum nas bibliotecas de desenho é a capacidade de preencher as formas com uma dada cor. O comportamento é simples, dada uma figura fechada, o programa deve preencher toda a figura com uma dada cor. Esse comportamento pode ser visto no “balde de tinta” do programa Paint do windows.

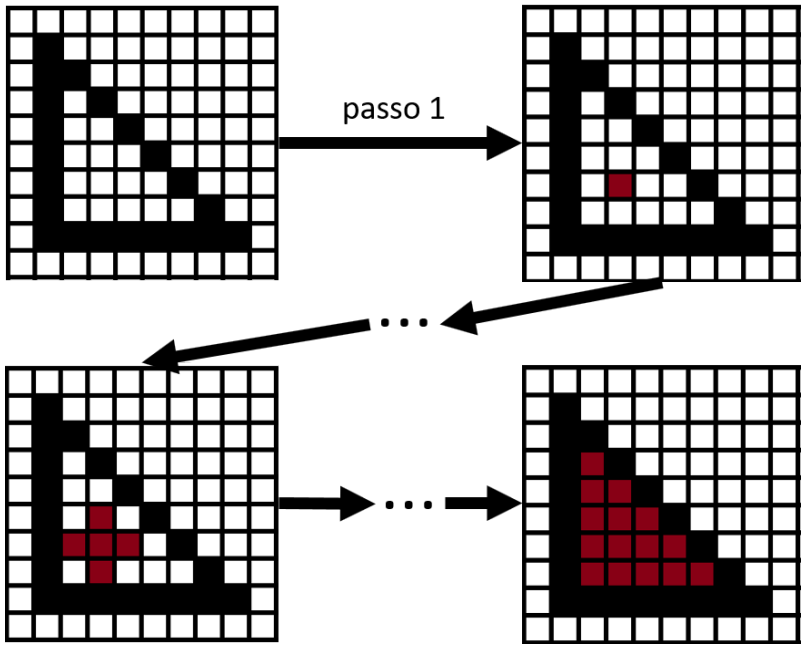


Figura 7 - Comportamento do algoritmo de preenchimento

A Figura 7 mostra como o algoritmo deve se comportar em uma figura fechada quando o usuário decidir pintar a parte interior iniciando de um ponto interno à figura. De forma análoga, o algoritmo de preenchimento preencheria toda a parte externa à figura se o ponto inicial fosse algum pixel na parte externa.

Descrição do projeto

O projeto de desenho gráfico deve ser desenvolvido em linguagem C e ser executado, em sua versão mais simples, através de linha de comando (entrada e saída em um console/terminal). Seu programa deve ler um arquivo texto contendo informações de uma imagem a ser gerada a partir de primitivas gráficas. A lista de funções gráficas que devem ser implementadas, no mínimo, estão listadas na Tabela 1.

Tablea 1, primitivas básicas mínimas que devem ser implementadas no projeto

Primitiva	Descrição
image	Cria uma nova “imagem”, com a largura e altura especificadas na primitiva
color	Muda a cor atual para uma cor especificada

clear	Limpa a imagem, setando todos os pixels para a cor especificada
rect	Desenha um retângulo nas posições x, y, e com tamanho especificados
circle	Desenha um círculo nas posições x, y e tamanho especificados
polygon	Desenha um polígono delimitado por uma lista de pontos
fill	Pinta todo o polígono a partir do ponto especificado. Caso o ponto não esteja em algum polígono, o comando deve preencher toda a tela até encontrar bordas de algum polígono ou as bordas da imagem (balde de tinta).
save	Salva a imagem atual em um arquivo usando o formato ppm
open	Carrega uma imagem ppm no programa para futuras operações de desenho

A título de ilustração, o texto a seguir exemplifica o conteúdo de um possível arquivo do projeto. Porém, fica a critério dos autores o formato das instruções que podem existir no arquivo de cada projeto.

```
image 600 400
clear 0 0 0
color 100 170 200
line 0 400 600 200
polygon 3 0 400 300 200 600 400
circle 200 100 50
color 180 30 50
fill 300 300
color 255 0 0
fill 0 0
save test.ppm
```

Neste exemplo, cada linha do arquivo utiliza um comando de desenho, exceto as duas primeiras linhas, que indicam respectivamente o tamanho da imagem a ser gerada (largura e altura). O comando clear limpa toda a imagem com a cor dada, neste caso RGB(0,0,0). O comando `color` altera a cor de desenho atual para um valor RGB definido pelos componentes R (100), G (170) e B (200). O comando `line` especifica os dois pontos das extremidades da linha a ser desenhada, cada um com suas coordenadas (x, y). O comando `polygon` no exemplo tem um valor inicial, 3, que indica que o polígono tem 3 pontos, seguido

então de 3 coordenadas (x,y), a saber: (0, 400), (300, 200) e (600, 400). O comando `circle` indica um ponto central e seu raio e, por fim, o comando `fill` indica o preenchimento usando a cor atual a partir de um ponto, neste caso o ponto 300,300 está dentro do triângulo desenhado pelo comando polygon, por isso espera-se que o triângulo seja pintado com a cor atual RGB(180,30,50). A segunda chamada do comando fill pinta o restante da imagem com a cor RGB(255,0,0).

O projeto deve atender também os seguintes critérios de programação:

1. Uso de arranjos / matrizes;
2. Uso de registros (`struct`);
3. Definição de novos tipos de dados através de `typedef`;
4. Uso de alocação dinâmica;
5. Uso de recursão;
6. Leitura e escrita de arquivos;
7. Modularização do programa em diferentes arquivos (uso de diferentes arquivos .c e .h, cada um com sua funcionalidade);
8. Definição de um padrão de indentação do código fonte e de nomenclatura das sub-rotinas e variáveis, e a adequação a este padrão;
9. Documentação adequada do código-fonte.

Observações:

- O projeto deve ser desenvolvido individualmente ou em grupos (grupos de três alunos). Não serão permitidos grupos com quatro ou mais alunos.
- Para facilitar o acompanhamento do projeto, cada grupo deve estar associada a uma única turma de ITP. Ou seja, não serão permitidos grupos formadas por alunos matriculados em turmas de ITP diferentes.
- Cada grupo deve desenvolver sua solução de forma independente das demais. Soluções idênticas serão consideradas plágios e, portanto, sanções serão devidamente aplicadas em todos os grupos com soluções similares.
- Códigos e algoritmos podem ser utilizados da web desde que devidamente referenciados. Caso sejam encontrados trechos de código na web equivalentes aos apresentados pelo grupo sem a devida citação, o código será igualmente considerado plágio e sanções serão aplicadas ao grupo. Vale salientar que a avaliação será realizada unicamente sobre o código produzido pelo grupo. Códigos retirados da web, apesar de permitidos com a devida citação, serão desconsiderados dos critérios de pontuação.

Avaliação

Esta avaliação compreende a execução e desenvolvimento do projeto em apresentações. A cada dia, o grupo deve apresentar uma etapa do projeto desenvolvido, seguindo o calendário abaixo:

ETAPA 1 - 25/out/2019

1. Documento de estilo de escrita de código, versão 1 (consultar o arquivo "Regras de estilo para codigo fonte" para mais detalhes).
2. Modularização do programa (quais os arquivos .c e .h), versão 1.

ETAPA 2 - 11/nov/2019

3. Tipos de dados necessários (`typedef`, `structs` e `enums`).
4. Documento de estilo de escrita de código, versão 2 (final)
5. Modularização do programa (quais os arquivos .c e .h), versão 2 (final).
6. Leitura do arquivo de especificação.
7. Geração de uma imagem PPM.

ETAPA 3 - 18/nov/2019

8. Desenho de retas.
9. Desenho de polígonos.

ETAPA 4 - 25/nov/2019

10. Desenho de círculos.
11. Preenchimento de cores.

ETAPA 5 - 02/dez/2019

12. Implementação de elementos extras, definidos pelo próprio grupo.

Atrasos - 06/dez/2019

Entrega de qualquer parte que não tenha sido apresentada. Pontos serão descontados nessa entrega.

Importante: O trabalho é em grupo, mas as avaliações nos chekpoints serão individuais. Desta forma, alunos de um mesmo grupo poderão ficar com notas diferentes.

Sobre os grupos

Os alunos têm ATÉ o dia 25 de outubro para comunicar aos professores se farão o trabalho em grupo (e a composição do mesmo) ou se farão individualmente.

Critérios de pontuação

O desenvolvimento do projeto aqui descrito vale **100%** da nota da terceira unidade.

A pontuação da avaliação seguirá os critérios e distribuição abaixo:

- **Atendimento dos requisitos funcionais: 50%**
  a imagem está representada corretamente? os filtros estão implementados corretamente? os filtros podem ser aplicados a qualquer imagem? etc.
- **Uso dos recursos da linguagem C: 20%**
  O grupo demonstrou saber usar de forma adequada os recursos da linguagem C (arranjos, structs, typedefs, recursividade, etc)?
- **Organização do código e documentação: 20%**
  o código está documentado? a identação e uso de { } seguem um padrão **(identação)**? o programa está devidamente modularizado em diferentes arquivos?
- **Funcionalidades complementares: 10%**
  as funcionalidades extras desenvolvidas pelo grupo foram suficientemente complexas?

A pontuação a ser dada pelas funcionalidades extras não é definida *a priori*. Cada caso será avaliado em função da complexidade envolvida. Itens extras de baixa complexidade serão desconsiderados na pontuação.

Entrega do projeto

O projeto deve ser submetido pelo SIGAA até a data 29 de novembro de 2019 em um **arquivo comprimido (.zip)** contendo os arquivos fontes do projeto (.c e .h) e um arquivo README.TXT. Este arquivo deve ter as seguintes informações:

- O que foi feito (o básico e as funcionalidades extras implementadas);
- O que não foi feito (caso não tenha sido possível implementar alguma funcionalidade);
- O que seria feito diferentemente (quando aprendemos à medida que vamos implementando, por vezes vemos que podíamos ter implementado algo de forma diferente. Assim, esse tópico é para vocês refletirem sobre o que vocês aprenderam durante o processo que, se fossem fazer o mesmo projeto novamente, fariam diferente);
- Como compilar o projeto, deixando explícito se foi utilizada alguma biblioteca externa que precise ser instalada, que versão e quais parâmetros extras são necessários para o compilador.
- Em caso de grupos:
  - Identificação dos autores;
  - Contribuição de cada integrante no desenvolvimento do projeto (quem fez o quê).

Recomendações diversas:

1. Solução de back-up: não será tolerada a desculpa de que um disco rígido falhou nas avaliações. Assim, é importante manter várias cópias do código-fonte desenvolvido ou usar um sistema de backup automático como o Dropbox, Google Drive, Box ou

similares. Uma solução melhor ainda é fazer uso de um sistema de controle de versões como git, e um repositório externo como Github ou Bitbucket.

2. Especificar precisamente a interface e o comportamento de cada sub-rotina. Usar esta informação para guiar o desenvolvimento e documentar o código desenvolvido.

Extras

Segue exemplos de funcionalidades extras que o programa pode ter.

- Interface gráfica.
- Trabalhar com desenhos figuras mais complexas como curvas de Bézier.
- Desenho de fontes. A ideia é usar curvas de Bezier para desenhar fontes como TTF, por exemplo. Pode-se utilizar também padrões matriciais para especificar letras. Um exemplo de chamada seria plot_string(x, y, "string");
- Compressão de imagens: usar um algoritmo como o RLE para criar imagens compactadas simples. (https://en.wikipedia.org/wiki/Run-length_encoding)
- Desenho de gráficos de funções. Ler uma função e plotar a mesma.

ATENÇÃO: o professor reserva-se o direito de modificar sem aviso prévio qualquer exigência ou descrição deste projeto, incluindo datas de entrega.