Introdução à Análise de Algoritmos
2º semestre de 2020 - Turmas 04 e 94

Primeiro Exercício Programa

Recursividade aplicada à computação gráfica

1 Introdução

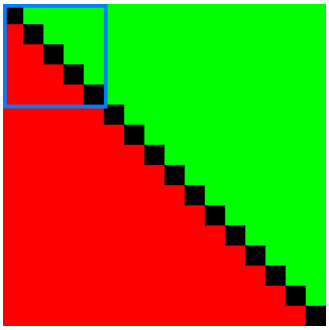
A proposta deste exercício programa é exercitar a programação de algoritmos recursivos aplicados à síntese de imagens digitais. Imagens digitais podem ser representadas como uma matriz de valores numéricos na qual cada elemento corresponde a um pixel da imagem. O valor de cada elemento está associado à luminosidade (para imagens em tons de cinza) ou à cor (para imagens coloridas) do pixel correspondente.

Para imagens em tons de cinza, é comum a representação em que cada pixel está associado a um valor inteiro contido no intervalo [0..255]. Neste caso, o valor 0 representa ausência total de luz (cor preta), o valor 255 representa intensidade luminosa máxima (cor branca) e os valores intermediários representam tons de cinza. Observe que, para esta representação, o valor de cada pixel pode ser armazenado em 1 byte.

Já para imagens coloridas, um pixel é representado por uma tripla de valores (r, g, b), onde cada valor da tripla também está contido no intervalo [0..255]. O valores r, g e b representam, respectivamente, a intensidade da cores vermelha, verde e azul presente no pixel. A combinação destes valores de cores primárias permite a geração de praticamente qualquer cor. Note que, para esta representação, são necessários 3 bytes para definir o valor de um pixel mas, se a imagem for criada como uma matriz de valores inteiros (de 32 bits), é possível empacotar as três componentes r, g e b do pixel em apenas um elemento da matriz (e ainda "sobra" um byte livre que pode ser usado, por exemplo, para definir um valor de transparência - *alpha channel*).

Um exemplo de imagem digital colorida de tamanho 16×16, representada como uma matriz, pode ser visto na Figura 1. À direita podemos ver o conteúdo da matriz referente a uma região de 5×5 pixels que está destacada na imagem da esquerda. É importante observar que, embora as representações descritas (tanto para imagens em tons de cinza quanto para imagens coloridas) sejam frequentemente utilizadas, elas não são as únicas, e existem diversas outras representações.

A tarefa de vocês neste exercício programa consiste em implementar duas funcionalidades para desenho/edição de imagens digitais: (i) desenho da *curva de Koch*; e (ii) preenchimento de região. Como auxílio para esta tarefa, é disponibilizada a classe **Image** que representa uma imagem digital e "esconde" a representação de baixo nível da imagem, além de fornecer funcionalidades básicas de desenho. Detalhes referentes à classe **Image**, assim como detalhes sobre as novas funcionalidades a serem implementadas, são apresentados nas seções a seguir.



(a)

(000,000,000)	(000,255,000)	(000,255,000)	(000,255,000)	(000,255,000)
(255,000,000)	(000,000,000)	(000,255,000)	(000,255,000)	(000,255,000)
(255,000,000)	(255,000,000)	(000,000,000)	(000,255,000)	(000,255,000)
(255,000,000)	(255,000,000)	(255,000,000)	(000,000,000)	(000,255,000)
(255,000,000)	(255,000,000)	(255,000,000)	(255,000,000)	(000,000,000)

(b)

Figura 1: (a) imagem de 16x16 pixels, com uma região de 5×5 pixels em destaque; (b) porção da matriz que representa a imagem, referente à região de 5×5 pixels destacada em (a). Cada pixel é representada por uma tripla de valores inteiros, cada um variando entre 0 e 255. O fundo de cada elemento da matriz foi colorido com a cor correspondente representada pela tripla.

2 Classe Image

Para que não seja preciso lidar diretamente com a representação de baixo nível usada para representar a imagem, a classe **Image** está disponível para a realização do EP. Para criar uma nova imagem basta instanciar um objeto desta classe (o construtor aceita como parâmetros as dimensões da imagem, bem como a cor de fundo a ser usada na mesma), e realizar chamadas aos métodos que implementam funcionalidades elementares de desenho: definição da cor de primeiro plano, desenho de ponto e desenho de reta (estas duas últimas funcionalidades usam a cor de primeiro plano - definida previamente - para realizar os desenhos). Há também um método que permite salvar a imagem gerada em arquivo, no formato **.png**.

Um exemplo de como usar a classe **Image** para gerar uma imagem é apresentado no código da classe **ImageTest**. Este, quando compilado e executado, produz como saída a imagem exibida na Figura 2.

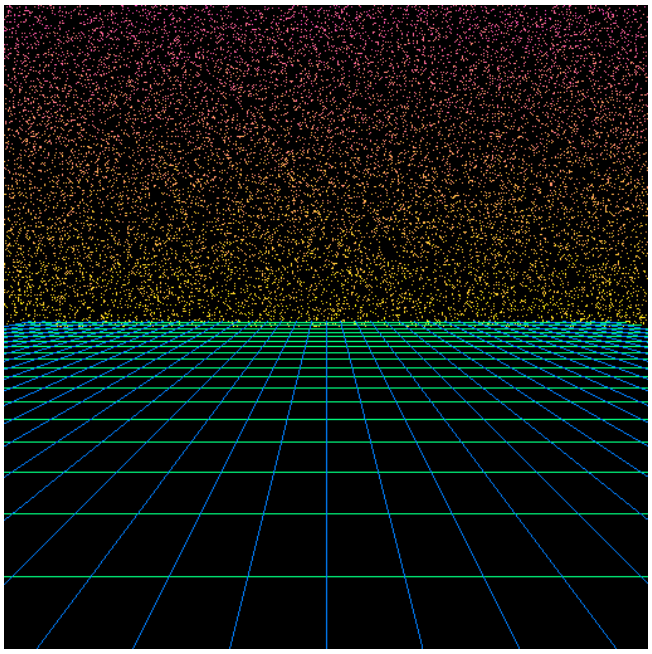


Figura 2: imagem no melhor estilo *computação gráfica retrô* produzida ao executar a classe **ImageTest**. Qualquer semelhança com as imagens encontradas no *Google Images* ao se buscar por *80's cgi grid* **não é** mera coincidência!

Para implementar as novas funcionalidades pedidas neste EP, deve-se criar uma classe derivada da classe **Image** que acrescente à classe base os novos recursos de desenho. Lembre-se que através do mecanismo de herança, os métodos que implementam as funcionalidades básicas (como desenho de ponto ou reta, cujo uso é ilustrado na classe **ImageTest**) estarão disponíveis para serem utilizados na implementação das novas funcionalidades dentro da classe derivada.

3 Curva de Koch

A *curva de Koch*, apresentada pela primeira vez em 1904 pelo matemático sueco Helge von Koch, foi uma das primeiras curvas fractais a ser descrita. Sem entrar em detalhes matemáticos formais, uma característica típica presente em curvas fractais é a autossimilaridade, observada pela repetição de uma mesma estrutura em diferentes escalas. Ou seja, existem partes da curva que possuem aspecto idêntico (ou bastante similar) à curva inteira. A Figura 3 ilustra o desenho da curva de Koch. Observe como os trechos da curva delimitados pelos retângulos possuem exatamente a mesma estrutura da curva toda, embora em escalas diferentes.

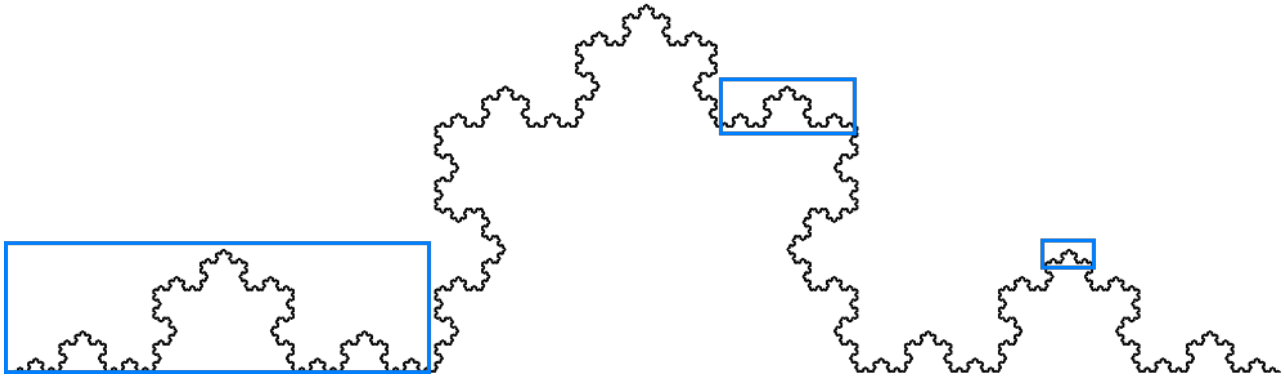


Figura 3: curva de Koch (*http://commons.wikimedia.org/wiki/File:Koch_curve.svg*).

A partir de um segmento de reta $\overline{PQ}$ inicial, já desenhado, o desenho da curva de Koch pode ser descrito através do seguinte algoritmo (veja a Figura 4):

1. calcule os pontos A, B, C (de modo que o comprimento de cada um dos segmentos $\overline{PA}$, $\overline{AB}$, $\overline{BC}$ e $\overline{CQ}$ seja 1/3 do comprimento de $\overline{PQ}$).
2. desenhe os segmentos $\overline{AB}$ e $\overline{BC}$.
3. apague o segmento $\overline{AC}$.
4. repita o algoritmo para cada um dos novos segmentos resultantes: $\overline{PA}$, $\overline{AB}$, $\overline{BC}$ e $\overline{CQ}$.

Obviamente é necessário interromper o processo descrito pelo algoritmo em algum momento. Isso pode ser feito quando o comprimento do segmento a ser processado for menor do que o valor determinado por um limiar. Observe também como o quarto passo tem "cara" de chamada recursiva: depois de dividir um problema (segmento de reta inicial) em subproblemas menores (segmentos com 1/3 do tamanho inicial) todo o procedimento é repetido para cada um dos subproblemas menores. Embora o algoritmo descrito funcione, perceba que alguns trechos da curva são desenhados sem necessidade, pois são apagados em um momento posterior. Assim, é possível modificar o algoritmo para que não haja a necessidade de apagar o segmento de reta no passo 3. Dados c o comprimento do segmento $\overline{PQ}$, e l o valor do limiar (que determina o comprimento mínimo que $\overline{PQ}$ deve ter para ser subdividido), podemos definir o seguinte novo algoritmo (bem mais elegante):

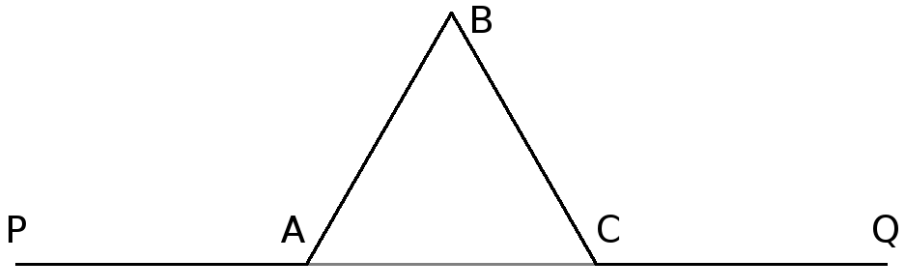


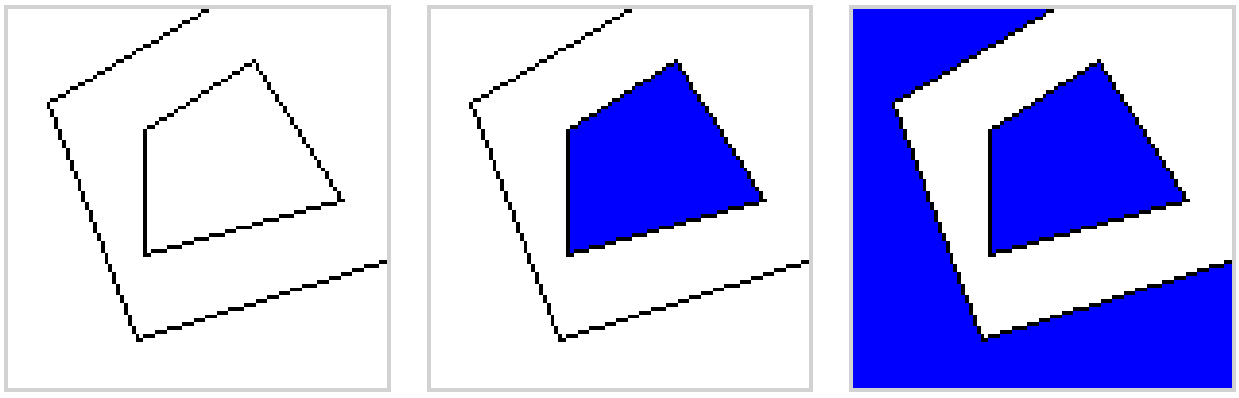
Figura 4: pontos utilizados para a construção da curva de Koch.

1. se $c < l$ então desenhe o segmento reta $\overline{PQ}$.
2. caso contrário:
   (a) calcule os pontos A, B, C.
   (b) chame o algoritmo recursivamente para $\overline{PA}$, $\overline{AB}$, $\overline{BC}$ e $\overline{CQ}$.

A primeira tarefa de vocês neste EP é implementar este algoritmo para desenho da curva de Koch. Sua implementação deve receber como parâmetros as coordenadas x e y dos pontos P e Q e o valor l do limiar. A curva deve ser desenhada com a cor previamente especificada através de chamada ao método **setColor** da classe **Image**.

4 Preenchimento de região

A segunda tarefa do EP consiste em implementar a funcionalidade de preenchimento de região. O preenchimento de região deve funcionar como ilustrado na Figura 5. Especifica-se um ponto (pixel) inicial, e toda a região contínua que existente em torno deste pixel (formada pelos pixels que possuem o mesmo valor de cor do pixel inicial) deve ser colorida com a cor previamente especificada através de chamada ao método **setColor** da classe **Image**.



(a) (b) (c)

Figura 5: funcionamento da funcionalidade de preenchimento de região: (a) antes do preenchimento; (b) depois de uma operação de preenchimento, tomando o ponto central da imagem como ponto inicial; (c) depois de mais uma operação de preenchimento, tomando como ponto inicial o canto superior esquerdo que possui coordenadas $(0, 0)$. Observe que a moldura cinza não faz parte da imagem, e foi desenhada apenas para delimitar a região ocupada pela mesma.

Uma forma simples de implementar essa ferramenta para preenchimento de regiões é através de um algoritmo recursivo que recebe como parâmetro as coordenadas de um pixel e executa os seguintes

passos: caso o valor de cor do pixel (recebido como parâmetro) seja o mesmo da cor original do pixel inicial (aonde se inciou o processo de preenchimento), então tal pixel é colorido e, em seguida, chama-se o algoritmo recursivamente para os pixels vizinhos (à esquerda, à direita, acima e abaixo); caso contrário, nada é feito. Para obter-se a cor de um determinado pixel pode-se usar o método **getPixel** e para colorir um pixel (com a cor previamente definida pelo método **setColor**) pode-se usar o método **setPixel** (funções estas pertencentes à classe **Image**).

5 Convenções adotadas no EP

Durante a realização deste EP, lembre-se que qualquer ponto (ou pixel) é definido através de suas coordenadas x e y, onde x e y são, respectivamente, as coordenadas horizontais e verticais do pixel (definindo, portanto, coluna e linha na matriz que representa a imagem). Lembre-se também que o canto superior esquerdo da imagem possui coordenada $(0, 0)$, que o valor da coordenada x cresce para a direita, e que o valor da coordenada y cresce para baixo.

6 O que deve ser entregue

Neste EP, como já mencionado, deve-se criar uma classe derivada da classe **Image**, que implementa as duas funcionalidades novas de desenho descritas neste enunciado. Cada uma delas deve ser implementada como um método da classe derivada. Vocês tem a liberdade para definir a assinatura de cada método, assim como criar métodos auxiliares que julgarem necessários. Importante: não faça qualquer modificação na classe **Image**.

Além desta classe vocês devem implementar um **programa principal**, que recebe dois parâmetros pela linha de comando, e deve ser executado da seguinte forma:

```
java ProgPrincipal entrada.txt saida.png
```

O arquivo de entrada é um arquivo em formato texto que contem as definições para criação de uma imagem (dimensões e cor de fundo), bem como comandos de desenho a serem executados. A imagem resultante da execução dos comandos deve ser salva com o nome de arquivo especificado no segundo parâmetro. O arquivo de entrada possui o seguinte formato:

```
LARGURA ALTURA FUNDO_R FUNDO_G FUNDO_B
COMANDO PARAM_1 PARAM_2 ...
COMANDO PARAM_1 PARAM_2 ...
...
COMANDO PARAM_1 PARAM_2 ...
```

A primeira linha contem, obrigatoriamente, 5 valores inteiros que definem as dimensões da imagem (largura e altura) e as componentes r, g e b da cor de fundo da mesma. Cada uma das demais linhas especifica um comando de desenho e os parâmetros exigidos por cada comando. Os seguintes comandos de desenho são válidos: **SET_COLOR**, **SET_PIXEL**, **DRAW_LINE**, **KOCH_CURVE** e **REGION_FILL**. Os três primeiros correspondem a métodos já fornecidos na classe **Image**: **setColor**, **setPixel** e **drawLine**. Os demais referem-se às novas funcionalidades que devem ser implementadas neste EP.

Para o comando **SET_COLOR** devem ser especificados 3 parâmetros inteiros, correspondentes às componentes r, g e b da cor de primeiro plano (cor esta que será usada pelos comandos de desenho

subsequentes). Para o comando **SET_PIXEL** devem ser especificados 2 parâmetro inteiros, que são as coordenadas x e y do pixel a ser colorido. Já para o comando **DRAW_LINE** devem ser especificados 4 valores inteiros: x_1 e y_1, (coordenadas de um dos pontos da reta), e x_2 e y_2 (coordenadas do outro ponto da reta).

Para o comando **KOCH_CURVE** devem ser especificados 5 valores inteiros: x_1 e y_1 (coordenada do ponto P), x_2 e y_2 (coordenada do ponto Q), e o valor l do limiar. Finalmente, para o comando **REGION_FILL**, devem ser especificados 2 valores inteiros referentes às coordenadas x e y do ponto a partir do qual o preenchimento será feito.

7 Entrega

Este EP deve ser feito individualmente, e sua entrega deve ser feita até o dia 22/11 às 23:59 pelo eDisciplinas. Para gerar o arquivo a ser entregue crie um diretório com seu número USP, e coloque dentro dele os arquivos fontes criados, juntamente com um arquivo **README** que explique como compilar seu EP (além outros comentários/informações que julgar relevante). Em seguida compacte o diretório, gerando um arquivo no formato **.zip**, que deve então ser enviado através do sistema. Não se esqueça de colocar seu nome e número USP em todos os arquivos fontes, e não se esqueça também de comentar seu código!

Boa diversão! :)