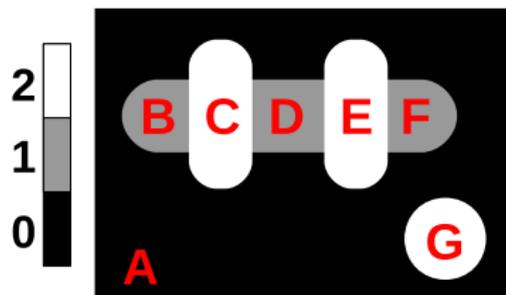


Árvore de componentes

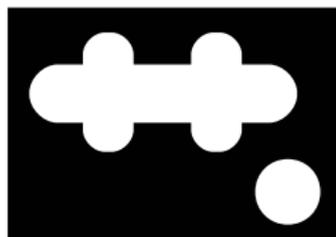
Prof. Dr. Paulo A. V. de Miranda
Instituto de Matemática e Estatística (IME),
Universidade de São Paulo (USP)
pmiranda@vision.ime.usp.br

Decomposição por limiarização

Uma imagem \hat{I} decomposta por limiar forma um conjunto $\mathcal{T}_{\hat{I}}$ de imagens binárias $\hat{B}_l = (\mathcal{D}_I, B_l)$, $l = 0, 1, \dots, I_{max}$, onde $B_l(p) = 1$ se $I(p) \geq l$ e $B_l(p) = 0$ caso contrário ($I_{max} = \max_{\forall p \in \mathcal{D}_I} \{I(p)\}$).



\hat{B}_0



\hat{B}_1



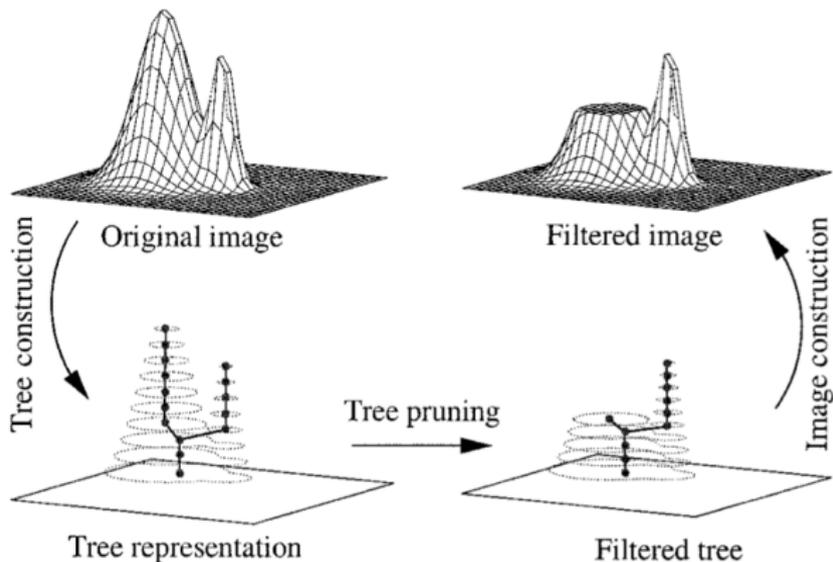
\hat{B}_2

Uma árvore de componentes é uma representação da imagem que descreve relações topológicas entre os componentes conexos de sua **decomposição por limiarização**.

- ▶ Analisando níveis consecutivos desta decomposição, podemos perceber que componentes do nível $l + 1$ estão contidos em componentes do nível l .
- ▶ Uma árvore de componentes é, portanto, um grafo que armazena esta hierarquia entre componentes.
- ▶ Nesta árvore as folhas são sempre os máximos regionais da imagem.

Árvore de componentes - Aplicações

Através de regras para a poda de ramos da árvore é possível gerar filtros conexos (ex: *area opening*, *volume opening*) e segmentações.



Árvore de componentes - *Max-tree*

- ▶ *Max-tree* é uma representação compacta da árvore de componentes, onde cada pixel é armazenado em um único nó da árvore, correspondendo ao nível mais alto em que ele aparece na árvore de componentes.
- ▶ Esta árvore é conhecida como *max-tree*, pois as folhas são sempre os máximos regionais da imagem.

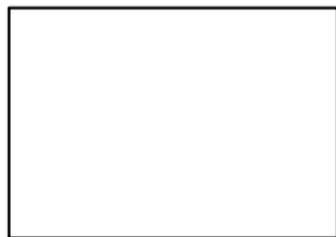
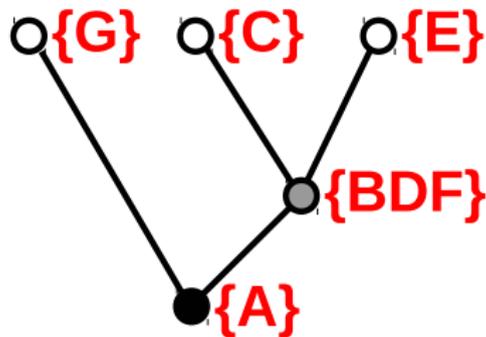
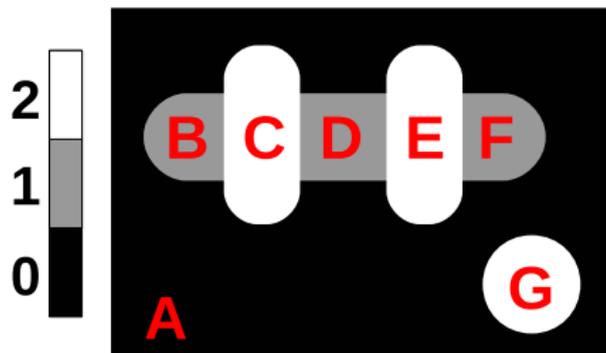
Os nós da *max-tree* são conjuntos de pixels de mesmo brilho (mas não necessariamente conexos).

- ▶ $I(p) = I(q)$ (mesmo brilho) é uma condição necessária (mas não suficiente) para dois pixels p e q pertencerem ao mesmo nó da *max-tree*.
- ▶ Pertencer a um mesmo platô (*flat zone*)¹ é uma condição suficiente (mas não necessária) para dois pixels pertencerem ao mesmo nó da *max-tree*.

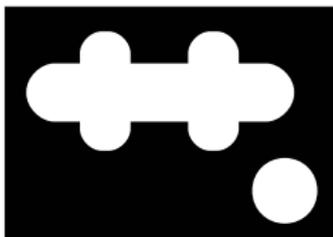
¹Flat zone é um componente conexo maximal, onde todos os pixels possuem o mesmo valor (mesma altitude).

Árvore de componentes - *Max-tree*

Exemplo:



\hat{B}_0



\hat{B}_1



\hat{B}_2

- ▶ O algoritmo processa os pixels em ordem decrescente de brilho (das folhas da árvore em direção a raiz),
- ▶ Um único representante, dado pelo mapa \hat{R} , é obtido em cada *flat zone*, considerando uma política LIFO entre vizinhos de mesmo brilho.
- ▶ Porém, um nó da *max-tree* pode conter várias *flat zones* desconexas de mesmo brilho l , contanto que elas pertençam ao mesmo componente na limiarização em \hat{B}_l . Logo, essas *flat zones* são tratadas como conjuntos disjuntos que devem ser unidos em \hat{R} .

Max-tree - Algoritmo

- ▶ Relações de parentesco entre nós são identificadas apenas quando pixels sendo processados no nível atual l encontram vizinhos em níveis superiores l^* já processados ($l^* > l$).
- ▶ Quando um pixel de um nó x no nível l encontra um vizinho de um nó y de nível maior l^* , existem três situações:
 - ▶ O nó y ainda não tem pai, logo x adota y como filho, ou
 - ▶ y já tem pai, e o brilho do seu ancestral de menor nível é igual ao brilho de x , portanto x e o ancestral de y pertencem ao mesmo nó, ou
 - ▶ y já tem pai, e o brilho do seu ancestral de menor nível é maior que o brilho de x , portanto x é pai deste ancestral de y .

Árvore de componentes - Algoritmo parte I

Algoritmo para construção de uma max-tree

Entrada: Imagem $\hat{I} = (\mathcal{D}_I, I)$ e adjacência \mathcal{A} .

Saída: Imagens $\hat{P} = (\mathcal{D}_I, P)$ de parentesco entre nós, e $\hat{R} = (\mathcal{D}_I, R)$ de representantes dos nós.

Auxiliares: Fila de prioridade Q com política de desempate LIFO, e imagem $\hat{N} = (\mathcal{D}_I, N)$ do número de elementos por nó.

Max-tree - Algoritmo parte II

01 Para Cada $t \in \mathcal{D}_I$, Faça

02 $P(t) \leftarrow nil, R(t) \leftarrow t, N(t) \leftarrow 1$, e insira t em Q .

03 Enquanto $Q \neq \emptyset$, Faça

04 Remova um pixel s de Q tal que $I(s)$ seja máximo.

05 Faça $r_s \leftarrow Representante(\hat{R}, s)$.

06 Para Cada $t \in \mathcal{A}(s)$, Faça

07 Se $I(s) = I(t)$, Então

08 Se $t \in Q$, Então

09 Se $R(t) = t$, Então $N(r_s) \leftarrow N(r_s) + 1$.

10 Remova t de Q , $R(t) \leftarrow r_s$, e insira t em Q .

11 Senão, Se $I(s) < I(t)$, Então

12 $r_t \leftarrow Representante(\hat{R}, t)$.

13 $r \leftarrow RaizAtual(\hat{P}, \hat{R}, r_t)$.

Max-tree - Algoritmo parte III

14 **Se** $r = nil$, **Então** $P(r_t) \leftarrow r_s$.

15 **Senão**, **Então**

16 **Se** $I(r) = I(r_s)$, **Então**

17 **Se** $r \neq r_s$, **Então** $Junte(\hat{R}, \hat{N}, r, r_s)$.

18 **Senão**, $P(r) \leftarrow r_s$.

19 **Para Cada** $t \in \mathcal{D}_I$, **Faça** $R(t) \leftarrow Representante(\hat{R}, t)$.

Max-tree - Algoritmo parte IV

Algoritmo para encontrar o representante com compressão:

Representante(\hat{R}, s)

01 Se $R(s) = s$, Então

02 Retorne s .

03 Senão , Então

04 Retorne $R(s) \leftarrow \text{Representante}(\hat{R}, R(s))$.

Algoritmo para unir componentes de mesmo nível:

Junte(\hat{R}, \hat{N}, r, r_s)

01 Se $N(r_s) \leq N(r)$, Então

02 $R(r_s) \leftarrow r$, $N(r) \leftarrow N(r) + N(r_s)$, e $r_s \leftarrow r$.

03 Senão , Então

04 $R(r) \leftarrow r_s$ e $N(r_s) \leftarrow N(r_s) + N(r)$.

P.S.: A variável r_s deve ser passada por referência.

Max-tree - Algoritmo parte V

Algoritmo para encontrar a raiz da subárvore atual (ancestral) de um nó:

RaizAtual(\hat{P}, \hat{R}, r_t)

01 $r_1 \leftarrow r_2 \leftarrow P(r_t)$

02 **Enquanto** $r_2 \neq nil$, **Faça**

03 $r_1 \leftarrow r_2 \leftarrow Representante(\hat{R}, r_2)$.

04 $r_2 \leftarrow P(r_2)$.

05 **Retorne** r_1 .

Max-tree - Algoritmo

- ▶ No final, o número de nós da *max-tree* é dado pelo número de representantes em \hat{R} (i.e., pixels p tais que $R(p) = p$).
- ▶ O mapa \hat{P} pode então ser usado para encontrar a raiz da árvore, o pai, e os filhos de cada nó.
- ▶ Após criar a árvore, o mapa de representantes deve ser convertido em um mapa de componentes, onde cada pixel p aponta para o nó correspondente na *max-tree*.
- ▶ Várias informações adicionais (ex: nível de cinza, número de pixels do nó, total de pixels da subárvore) podem ser armazenadas nos nós da árvore, de modo a favorecer critérios de poda para fins de filtragem.

- ▶ *Prof. Alexandre Xavier Falcão,*
Anotações de aula
(MO815 - Processamento de Imagens usando Grafos)
<http://www.ic.unicamp.br/~afalcao/mo815-grafos/index.html>
- ▶ *L. Najman, and M. Couprie,*
Building the component tree in quasi-linear time,
IEEE TIP, vol. 15, no. 11, pp. 3531-3539, 2006.
<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1709995>
- ▶ *P. Salembier, A. Oliveras, L. Garrido,*
Antiextensive Connected Operators for Image and Sequence Processing,
IEEE Transactions on Image Processing, vol. 7, no. 4, 1998.