计算机视觉与模式识别

苏远岐，新型计算机研究所

Jianbo Shi， University of Pennsylvania

第七章 Morphing and Carving

Morphing是物体的平均，包括几何和外观的平均；

Carving是依据图像内容的物体缩放；

一、Morphing

第七章 Morphing and Carving

Morphing是物体的平均，包括几何和外观的平均；

Carving是依据图像内容的物体缩放；

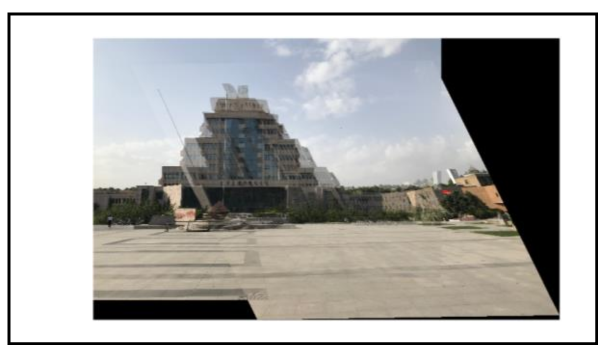
一、Morphing

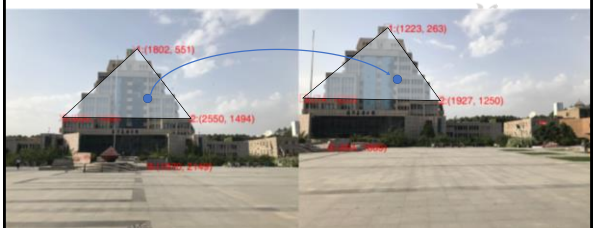
二、Carving

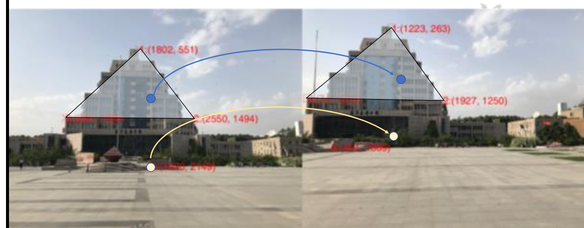
一、Morphing

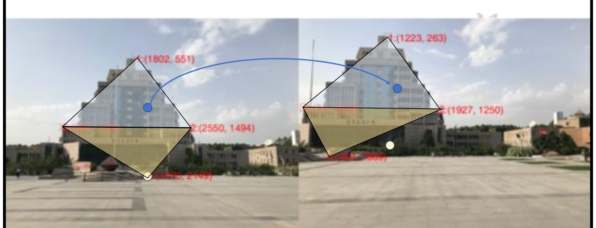

1:(1802, 551)
2:(2550, 1494)
1:(1223, 263)
2:(1927, 1250)


1:(1802, 551)
2:(2550, 1494)
1:(1223, 263)
2:(1927, 1250)





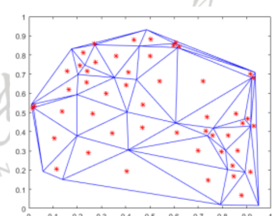




DT = delaunayTriangulation(P)

```
% Generate 30 2D points
P = gallery('uniformdata',[30 2],0);
DT = delaunayTriangulation(P);

% Display the triangulation results
IC = incenter(DT);
triplot(DT)
hold on
plot(IC(:,1),IC(:,2),'*r')
```



Morphing = 物体的平均

两个物体之间的平均

不是两个物体图像的直接平均…

…而是物体均值的图像!

Morphing = 物体的平均

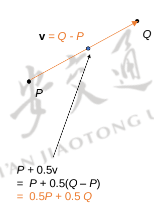
对于我们而言，我们如何得知两个物体的平均是什么?

- 从物理世界的角度我们不知道!
- 但是我们可以通过一定的方式让人感觉两个物体之间的平均。

Morphing = Warping + Cross Dissolving

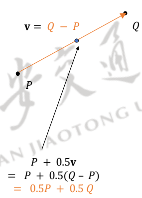
点的平均

点P和Q的平均是什么?



点的平均

点P和Q的平均是什么?



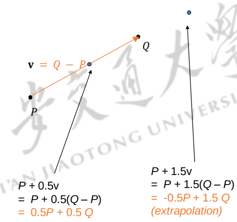
线性插值(仿射组合):

新的点坐标：$aP + bQ$

需要满足：$a + b = 1$

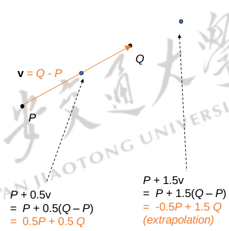
因此： $aP + bQ = aP + (1 - a)Q$

点的平均

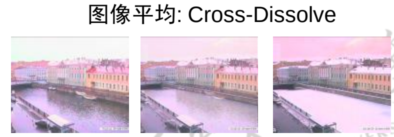


点的平均

- P和Q可以是任何东西:
  - 二维(2D)或者三维(3D)空间的点
  - RGB或者HSV(3D)空间的点
  - 整幅图像(m-by-n D)… etc.



图像平均: Cross-Dissolve



基于全图像的插值:

$$\text{Image}_{\text{halfway}} = (1\text{-}t)*\text{Image}_1 + t*\text{image}_2$$

这种技术在电影工业中称为：**cross-dissolve**

图像平均: Cross-Dissolve

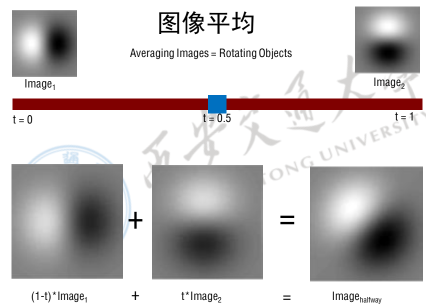
基于全图像的插值:

$$Image_{halfway} = (1-t)*Image_1 + t*image_2$$

这种技术在电影工业中称为：**cross-dissolve**

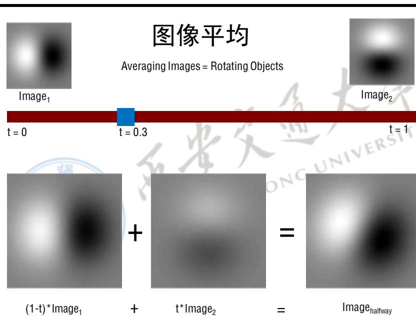
图像平均

Averaging Images = Rotating Objects



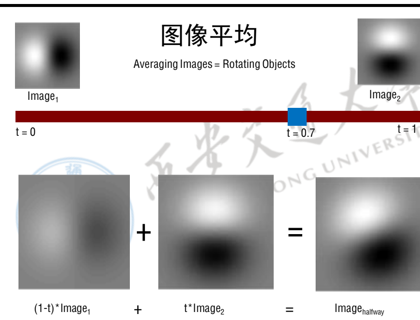
图像平均

Averaging Images = Rotating Objects

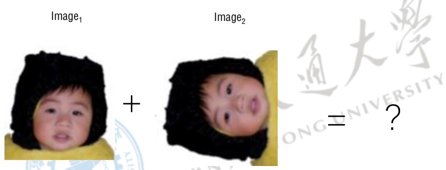


图像平均

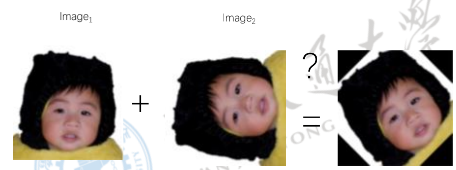
Averaging Images = Rotating Objects



图像平均



图像平均



图像平均

Averaging Images != Rotating Complex Objects

猫-孩子的平均

基于特征匹配的物体平均!

鼻子对鼻子，眼睛对眼睛，嘴巴对最大 等等.

非参数化的**warp**

猫-孩子的平均

基于特征匹配的物体平均！

鼻子对鼻子，眼睛对眼睛，嘴巴对最大 等等.

非参数化的**warp**

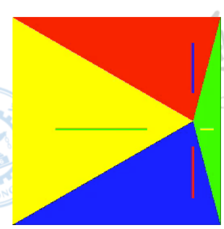
Warping, then cross-dissolve

Morphing的过程:

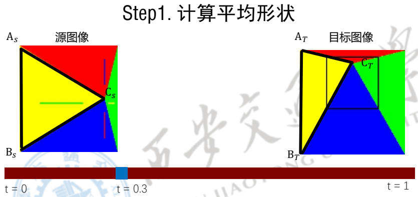
1. 获取平均形状
2. 非参数化的warping
3. 获取平均图像
   - Cross-dissolve the warped images

7.1.2、一个示例

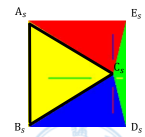
一个示例：



Step1. 计算平均形状

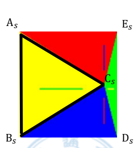


形状如何表示呢？



二维点集：$(A_S, B_S, C_S, D_S, E_S)$

形状如何表示呢？

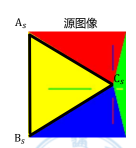


二维点集：$(A_S, B_S, C_S, D_S, E_S)$

三角面片：

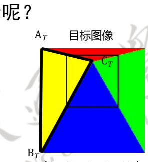
A_S	B_S	C_S
A_S	C_S	E_S
B_S	D_S	C_S
C_S	D_S	E_S

形状如何表示呢？



$(A_S, B_S, C_S, D_S, E_S)$

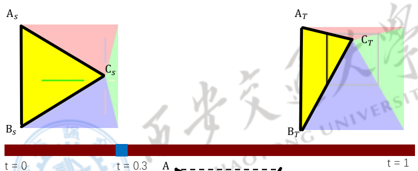
A_S	B_S	C_S
A_S	C_S	E_S
B_S	D_S	C_S
C_S	D_S	E_S



$(A_T, B_T, C_T, D_T, E_T)$

A_T	B_T	C_T
A_T	C_T	E_T
B_T	D_T	C_T
C_T	D_T	E_T

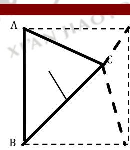
Step 1: 三角形插值



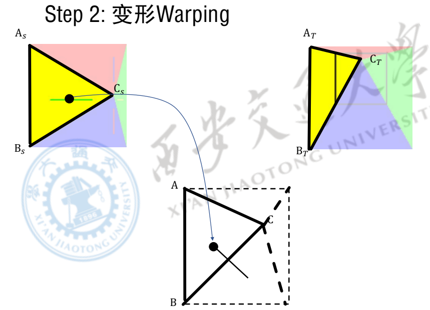
$A_t = (1-t)A_S + tA_T$

$B_t = (1-t)B_S + tB_T$

$C_t = (1-t)C_S + tC_T$



Step 2: 变形Warping



Step 2: 变形（Warping）

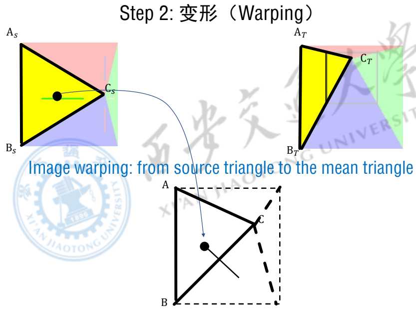


Image warping: from source triangle to the mean triangle

给定输入的数字图像：

$I(x,y)$

其中$x = 1,2,\cdots,X$；$y = 1,2,\cdots,Y$；

后向变换：

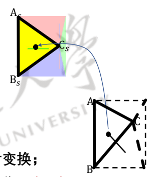
1. 确定输出图像O的区域$[1 \cdots X'] \times [1 \cdots Y']$
2. 遍历输出图像O的像素(x',y');
3. 根据逆变换，寻找输入图像中的相应像素位置(x,y);
4. 填充输出图像O的相应位置的像素值:

$$O(x',y') = I(x,y)$$

给定输入的数字图像：

$I(x,y)$

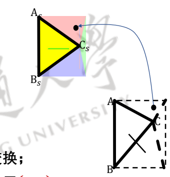
其中$x = 1,2,\cdots,X$；$y = 1,2,\cdots,Y$；



后向变换：

1. 确定输出图像O的区域$[1\cdots X'] \times [1\cdots Y']$
2. 遍历输出图像O的像素(x', y');
3. **确定像素所属的三角形，获得相应的仿射变换；**
4. 根据逆变换，寻找输入图像中的相应像素位置(x,y);
5. 填充输出图像O的相应位置的像素值:

$O(x', y') = I(x,y)$

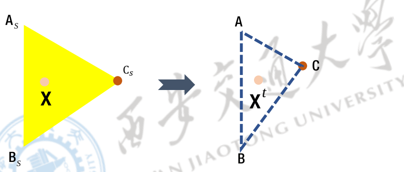


后向变换：

1. 确定输出图像O的区域$[1\cdots X'] \times [1\cdots Y']$
2. 遍历输出图像O的像素(x', y');
3. **确定像素所属的三角形，获得相应的仿射变换；**
4. 根据逆变换，寻找输入图像中的相应像素位置(x,y);
5. 填充输出图像O的相应位置的像素值:

$O(x', y') = I(x,y)$

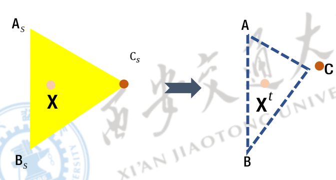
三角形的变形 ＝ 仿射变换



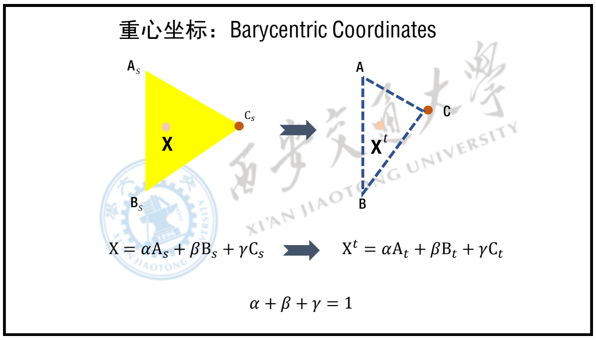
仿射变换是点到点的变换 $\mathbf{X} \longrightarrow \mathbf{X}^t$

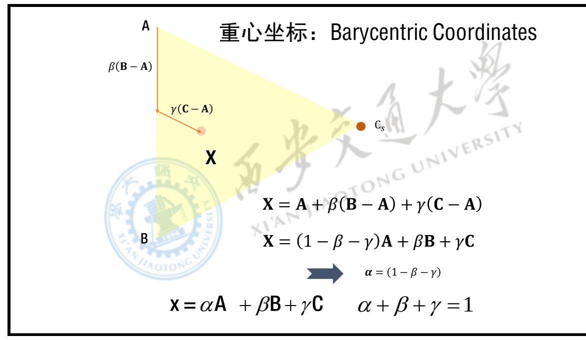
It is controlled by the movement of the three vertices of the triangle

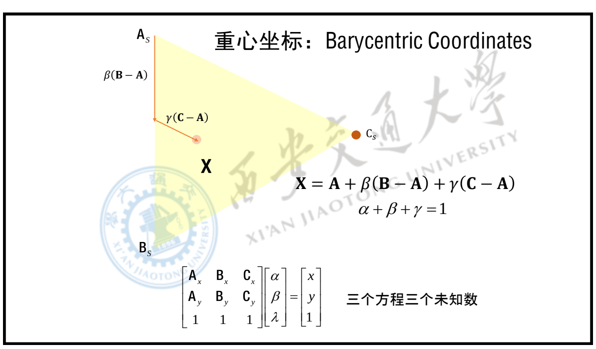
重心坐标：Barycentric Coordinates

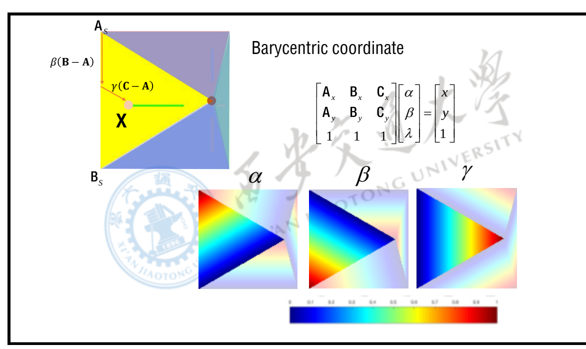


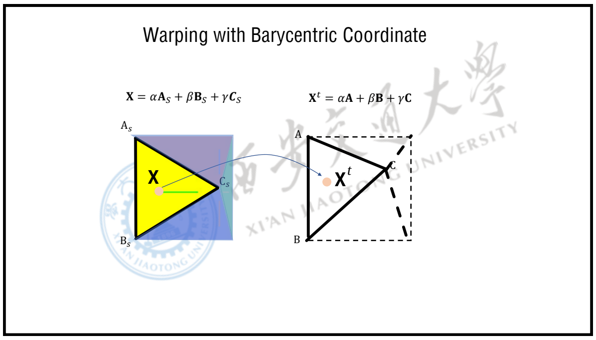
三角形中的每个点X相对于三个顶点有一个仿射不变的表示

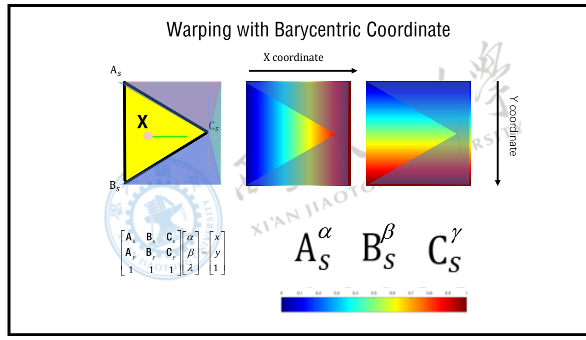

重心坐标：Barycentric Coordinates
A_s
C_s
X
B_s
A
C
X^t
B
$X = \alpha A_s + \beta B_s + \gamma C_s$
$X^t = \alpha A_t + \beta B_t + \gamma C_t$
$\alpha + \beta + \gamma = 1$

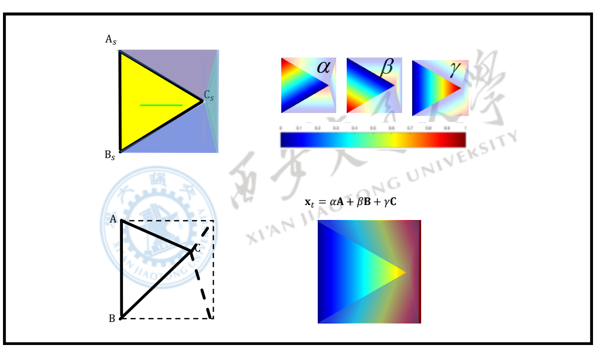

A
重心坐标：Barycentric Coordinates
$\beta(\mathbf{B}-\mathbf{A})$
$\gamma(\mathbf{C}-\mathbf{A})$
C_s
X
B
$\mathbf{X} = \mathbf{A} + \beta(\mathbf{B}-\mathbf{A}) + \gamma(\mathbf{C}-\mathbf{A})$
$\mathbf{X} = (1-\beta-\gamma)\mathbf{A} + \beta\mathbf{B} + \gamma\mathbf{C}$
$\alpha = (1-\beta-\gamma)$
$\mathbf{X} = \alpha\mathbf{A} + \beta\mathbf{B} + \gamma\mathbf{C} \quad \alpha+\beta+\gamma=1$

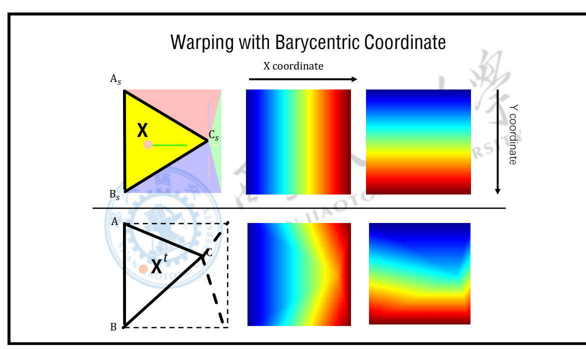

A_s
重心坐标：Barycentric Coordinates
$\beta(\mathbf{B}-\mathbf{A})$
$\gamma(\mathbf{C}-\mathbf{A})$
C_s
X
$\mathbf{X} = \mathbf{A} + \beta(\mathbf{B}-\mathbf{A}) + \gamma(\mathbf{C}-\mathbf{A})$
$\alpha+\beta+\gamma=1$
B_s
$\begin{bmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \lambda \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
三个方程三个未知数


A_s
$\beta(\mathbf{B}-\mathbf{A})$
$\gamma(\mathbf{C}-\mathbf{A})$
X
B_s
Barycentric coordinate
$\begin{bmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \lambda \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
α
β
γ

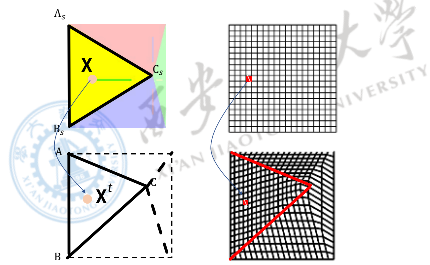

Warping with Barycentric Coordinate
$\mathbf{X} = \alpha\mathbf{A}_S + \beta\mathbf{B}_S + \gamma C_S$
$\mathbf{X}^t = \alpha\mathbf{A} + \beta\mathbf{B} + \gamma\mathbf{C}$
A_s
C_s
B_s
X
A
X^t
C
B


Warping with Barycentric Coordinate
X coordinate
Y coordinate
A_s
X
C_s
B_s
$A_S^\alpha \quad B_S^\beta \quad C_S^\gamma$

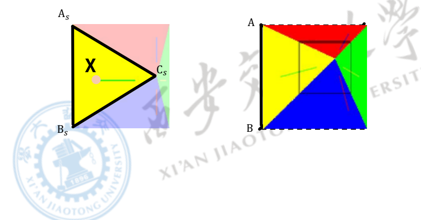

A_s
C_s
B_s
α
β
γ
$\mathbf{x}_t = \alpha\mathbf{A} + \beta\mathbf{B} + \gamma\mathbf{C}$
A
C
B


Warping with Barycentric Coordinate
X coordinate
Y coordinate
A_s
X
C_s
B_s
A
X^t
C
B

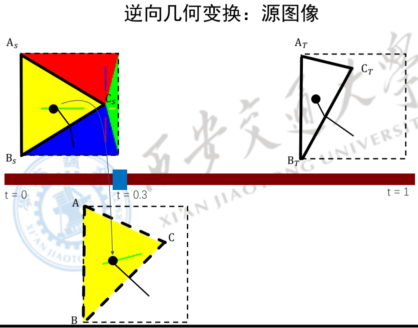
Grids before and after warping



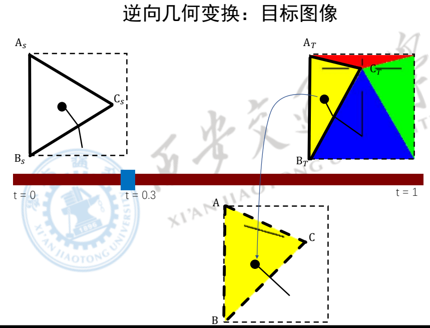
Step 3: Average warped image



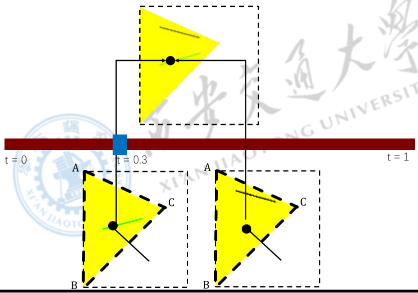
逆向几何变换：源图像



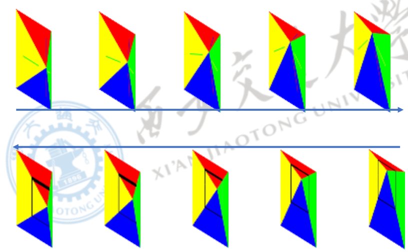
逆向几何变换：目标图像



Step 3: 加权平均变形后的图像



Warping and Cross Dissolve

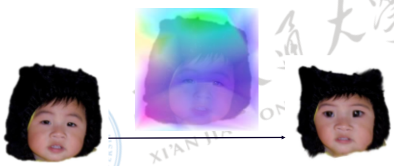


7.1.3、真实图像上的演示

Morphing = Warping, then cross-dissolve

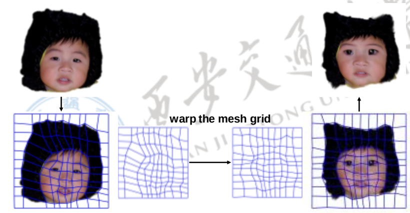
Morphing procedure:
for every t,
1. Find the average shape
2. Non-parametric warping
3. Find the average color
   - Cross-dissolve the warped images

Image warping idea 1: dense flow



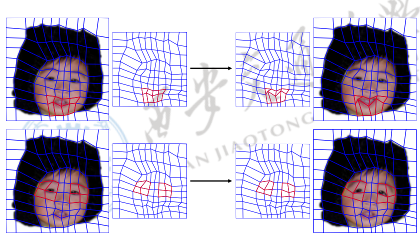
Displacement vector (u,v) for each pixel.
Great details··· but too much work, let's simply it to mesh grid

Image warping idea 2 : dense grid



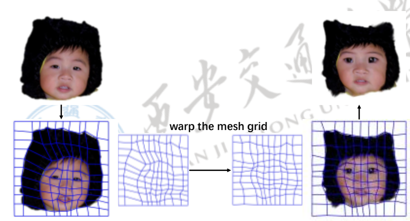
Define and manipulate the mesh grid

Image warping idea 2 : dense grid



Grid deformation generates expression change

Image warping idea 2 : dense grid

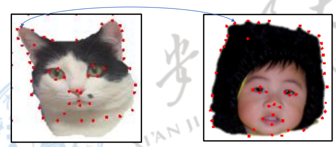


Still too much work…
simplify it to sparse control points and triangles

Step0：准备点对与三角剖分

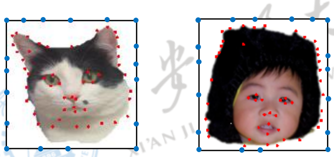
找到一组一一对应的点对

Step0：准备点对与三角剖分



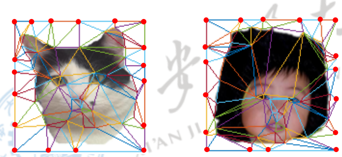
找到一组一一对应的点对

Step0：准备点对与三角剖分



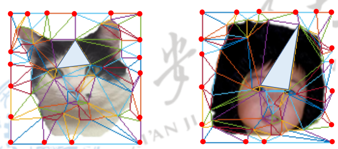
在边框上加上适量的点，覆盖背景区域

Step0：准备点对与三角剖分



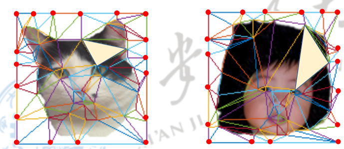
- 通过选择的特征点定义三角面片
- 三角面片和三角面片之间存在一一对应的关系
- 整体图像的变形通过每个三角面片进行变形实现

Step0：准备点对与三角剖分



- 通过选择的特征点定义三角面片
- 三角面片和三角面片之间存在一一对应的关系
- 整体图像的变形通过每个三角面片进行变形实现

Step0：准备点对与三角剖分



- 通过选择的特征点定义三角面片
- 三角面片和三角面片之间存在一一对应的关系
- 整体图像的变形通过每个三角面片进行变形实现

Step0.准备点对与三角剖分

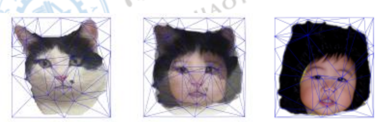
1. 给定两幅图像之间的一一对应的特征点
2. 同时我们在特征点上定义了三角剖分
   - 三角剖分和三角剖分之间一一对应

Step1.获取平均形状

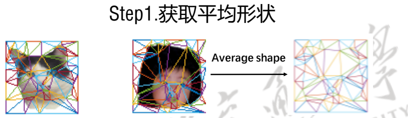
- 给定t，我们如何获得平均形状呢?
  - 假设 $t = [0,1]$，我们在每一个特征点对之间插值
  - 给定源图像和目标图像的点对(p_1, p_2)

$$p_t = (1-t) \times p_1 + t \times p_2$$

  - 保持三角面片不变



Step1.获取平均形状

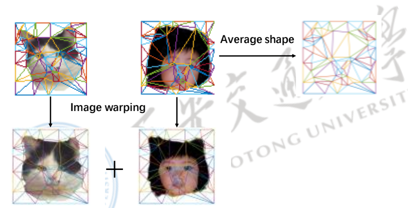


- 给定t，我们如何获得平均形状呢?
  - 假设 $t = [0,1]$，我们在每一个特征点对之间插值
  - 给定源图像和目标图像的点对(p_1, p_2)

$$p_t = (1-t) \times p_1 + t \times p_2$$

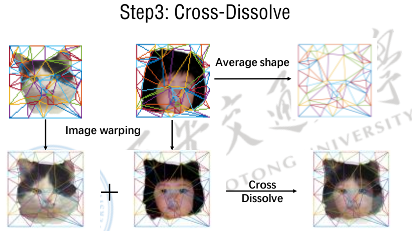
  - 保持三角面片不变

Step2:变形 Warping



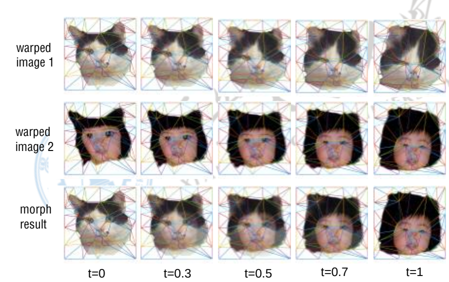
- 将源图像和目标图像都变形到平均形状

Step3: Cross-Dissolve



- 将变形后的图像进行平均（Cross Dissolve）

Morphing产生的序列

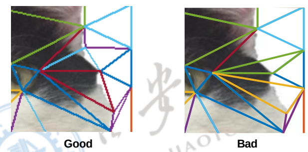


Delaunay三角剖分

- Delaunay三角剖分的时间复杂度可以为：$O(n\log n)$.
- 可以利用Matlab的函数来完成

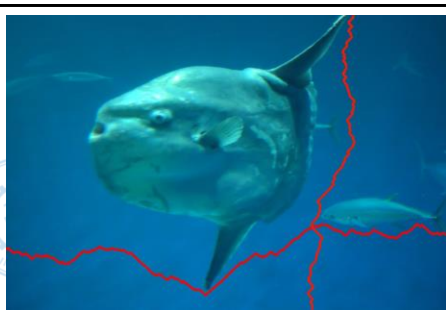
DT = delaunayTriangulation(P);

什么特征点是好的特征点呢?



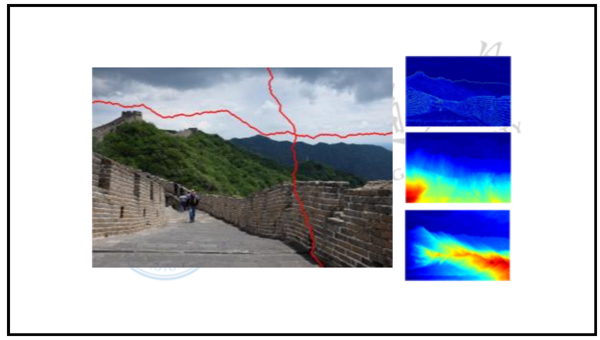
- 三角剖分应该与物体的边界保持一致
  - 物体的纹理不会和背景的颜色产生混淆
- 同时需要保持物体各个部件之间的相对关系不变

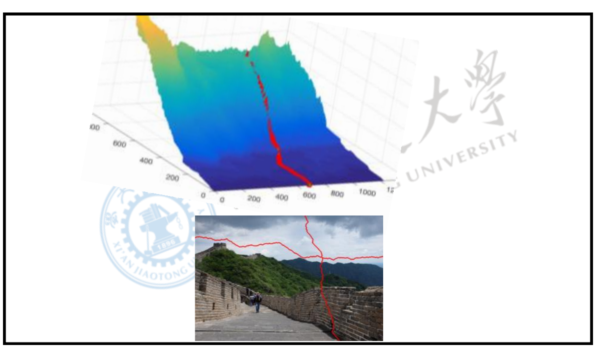
二、Carving

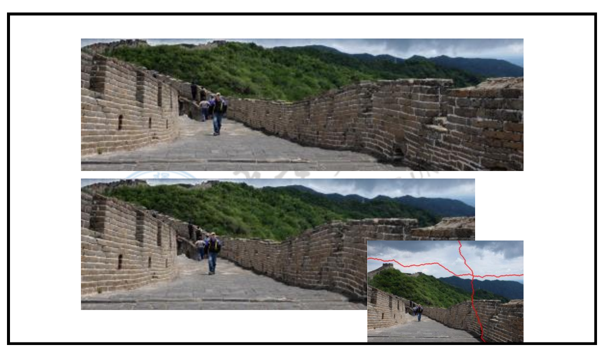


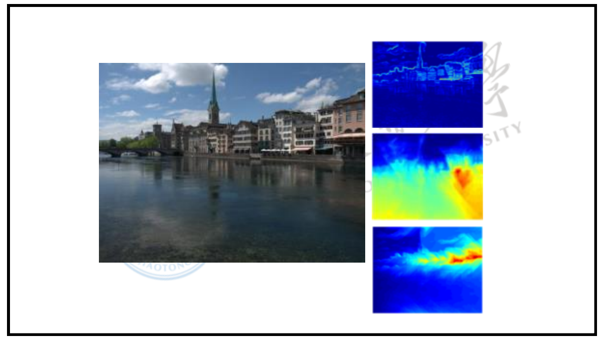
$820 \times 546 \times 3$

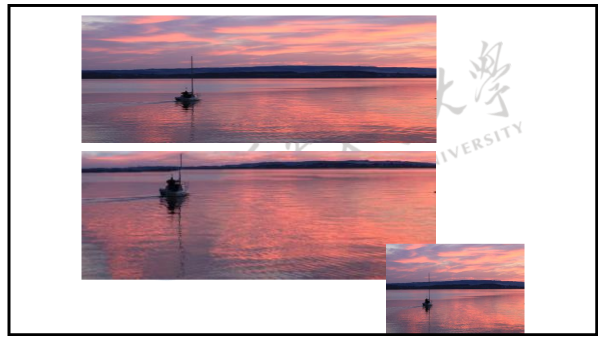

420 × 546 × 3
(a)
(b)
(c)
猜一猜
我们采用了“缩放”、“carving”和“扣取”来对源图像
进行尺寸的处理
猜猜哪一个是‘Carving’
820 × 546 × 3

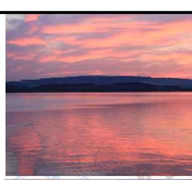




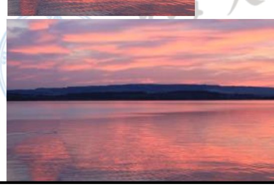


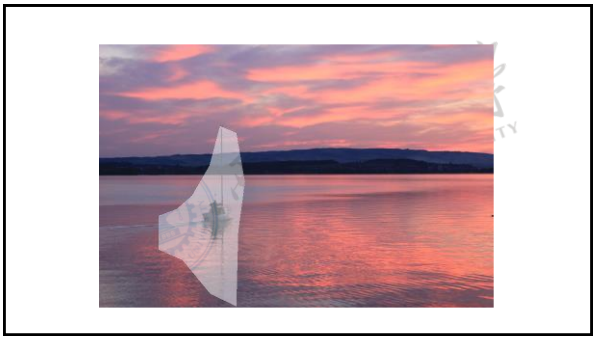


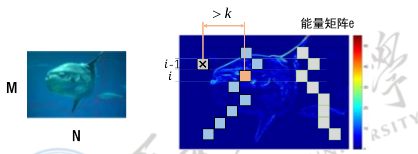




Expanded

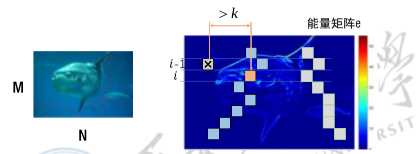






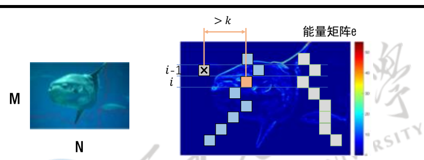
Seam:

$S^y: \{(i, y(i)) | i = 1, \cdots, M\}$ **s.t.** $|y(i) - y(i-1) \leq k|$

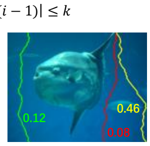


$S^y: \{(i, y(i)) | i = 1, \cdots, M\}$ **s.t.** $|y(i) - y(i-1) \leq k|$

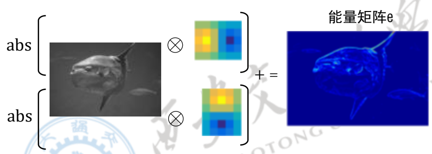
Seam Cost: $E(S^y) = \sum_{i=1}^{M} e(S^y(i))$



- Seam: $S^y: \{(i, y(i)) | i = 1, \cdots, M\}$ s.t. $|y(i) - y(i-1)| \leq k$
- Seam Cost: $E(S^y) = \sum_{i=1}^{M} e(S^y(i))$
- Goal: $S^* = \min_{S} E(S^y)$



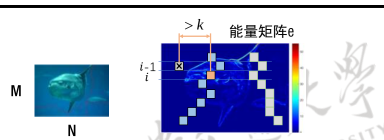
能量矩阵e从哪儿来呢?



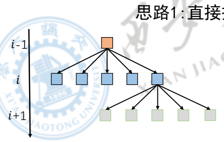
能量矩阵

For example: L1 norm of the edge gradients for the energy function

有了能量矩阵之后，如何求取最佳的Seam呢？

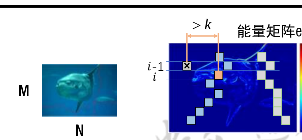


思路1：直接搜索

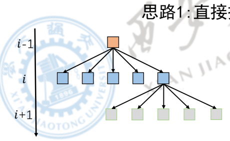


- 1st row: N

一共有多少可能的S^y?

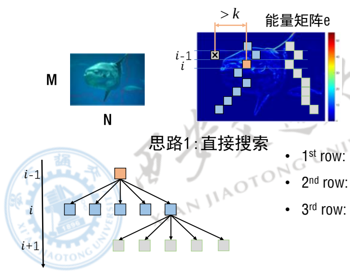


思路1：直接搜索



- 1st row: N
- 2nd row: (2K + 1)N

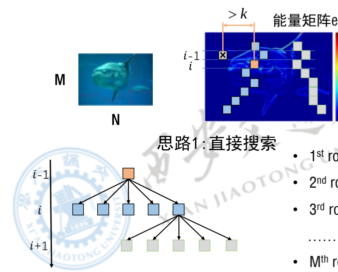
一共有多少可能的S^y?



思路1：直接搜索

- 1^{st} row: N
- 2^{nd} row: $(2K+1)N$
- 3^{rd} row: $(2K+1)^2N$

一共有多少可能的S^y?



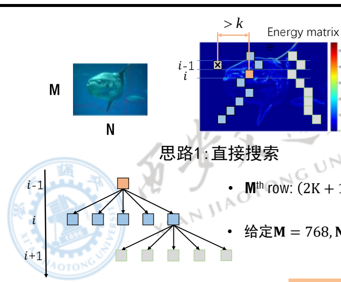
思路1：直接搜索

- 1^{st} row: N
- 2^{nd} row: $(2K+1)N$
- 3^{rd} row: $(2K+1)^2N$

……

- M^{th} row: $(2K+1)^{M-1}N$

一共有多少可能的S^y?



思路1：直接搜索

- $\mathbf{M}^{th}$ row: $(2K+1)^{M-1}N$
- 给定$\mathbf{M}=768, \mathbf{N}=1024, k=1$

一共有多少可能的S^y? 1024×3^{767}

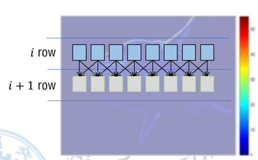
太多的可能性了！

1024×3^{767}

思路2：找到一条从第一行到最后一行的最短路径！

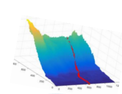
思路2：找到一条从第一行到最后一行的最短路径！

Dijkstra算法



构建一个有向图

有向图的边由每一个像素（节点）和它在下一行的 $(2k+1)$近邻构成



- 构建一个内部集合**S**，将这个内部集合初始化为第一行
- 构建一个值函数矩阵$\mathbf{V}(u)$，记录有向图中任意一个节点到内部集合**S**的最短距离
- 逐步增长内部矩阵集合**S**，直至这个集合包含有最后一行的像素（节点）

- 将内部集合S初始化为为第一行，并令值函数$\mathbf{V}(u)$

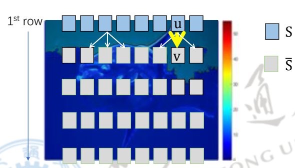
$$\mathbf{V}(u) = \mathbf{e}(u), u \in \mathbf{S}$$

对第一行元素而言，最短路径仅仅包含它们自身

- 将内部集合S初始化为为第一行，并令值函数$\mathbf{V}(u)$

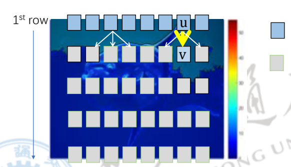
$$\mathbf{V}(u) = \mathbf{e}(u), u \in \mathbf{S}$$

对第一行元素而言，最短路径仅仅包含它们自身



迭代：(1) 找到最小的代价扩展S: $v = \arg \min_{u\in S, v\in \bar{S}} [\mathbf{V}(u) + \mathbf{e}(v)]$

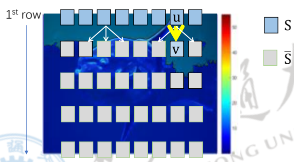
$(u \to v)$是有向图的一条边



迭代：(1) 找到最小的代价扩展S: $v = \arg \min_{u\in S, v\in \bar{S}} [\mathbf{V}(u) + \mathbf{e}(v)]$

$(u \to v)$是有向图的一条边

(2) 将节点v添加到内部集合S: $S \leftarrow S \cup \{v\}$

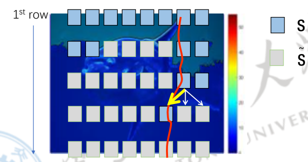


迭代：（1）找到最小的代价扩展S: $v = \arg\min_{u\in S, v\in \bar{S}} [\mathbf{V}(u) + \mathbf{e}(v)]$

(u → v)是有向图的一条边

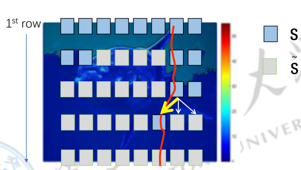
（2）将节点v添加到内部集合S: $S \leftarrow S\cup\{v\}$

（3）记录每个新加入的节点的前一个节点：$P(v) = u$



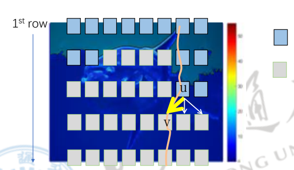
值函数：$\mathbf{V}(v) = \min_{u\in S, q\in \tilde{S}} [\mathbf{V}(u) + \mathbf{e}(q)]$

?



值函数：$\mathbf{V}(v) = \min_{u\in S, q\in \tilde{S}} [\mathbf{V}(u) + \mathbf{e}(q)]$

节点v到第一行的最短路径的能量值

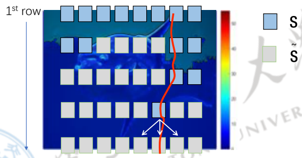


迭代：（1）找到最小的代价扩展S: $v = \arg\min_{u\in S, q\in \tilde{S}} [\mathbf{V}(u) + \mathbf{e}(q)]$

(u → q)是有向图的一条边

（2）将节点v添加到内部集合S: $S \leftarrow S\cup\{v\}$

（3）记录每个新加入的节点的前一个节点：$P(v) = u$

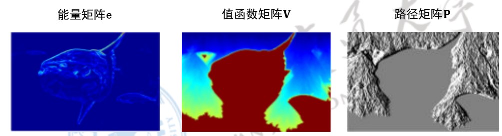


迭代：（1）找到最小的代价扩展S：$v = \arg\min_{u \in S, q \in \tilde{S}} [\mathbf{V}(u) + \mathbf{e}(q)]$

$(u \rightarrow q)$是有向图的一条边

（2）将节点v添加到内部集合S：$S \leftarrow S \cup \{v\}$

（3）记录每个新加入的节点的前一个节点：$P(v) = u$



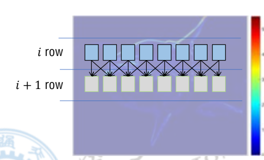
这种方式有什么缺点呢？

这种方式有什么缺点呢？

- 每次只能增加一个节点
- 每次只能找到一个最佳路径

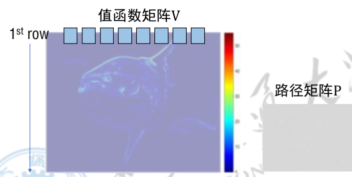
以行的方式进行扩张

思路3：动态规划



仍然采用相同的有向图结构

但是采用行的方式来扩张内部集合S，并且更新值函数矩阵V

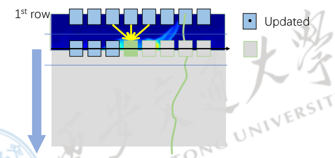


- 仍然从第一行开始，将值函数矩阵初始化为能量函数的第一行

$$\mathbf{V}(1,j) = \mathbf{e}(1,j) \qquad j = 1,\cdots,N$$

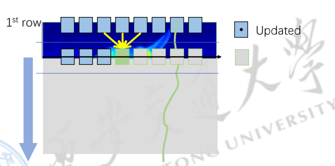
- 设置路径函数

$$\mathbf{P}(1,j) = 0$$



- 扩张到第二行，并更新值函数矩阵

$$V(2,j) = e(2+j) + \min\big(V(1,j-1), V(1,j), V(1,j+1)\big)$$

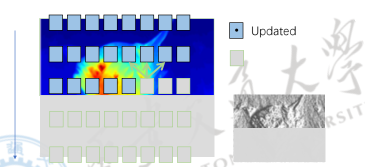


- 扩张到第二行，并更新值函数矩阵

$$V(2,j) = e(2+j) + \min\big(V(1,j-1), V(1,j), V(1,j+1)\big)$$

- 设置路径函数

$$P(2,j) = \mathrm{argmin}\big(V(1,j-1), V(1,j), V(1,j+1)\big)$$

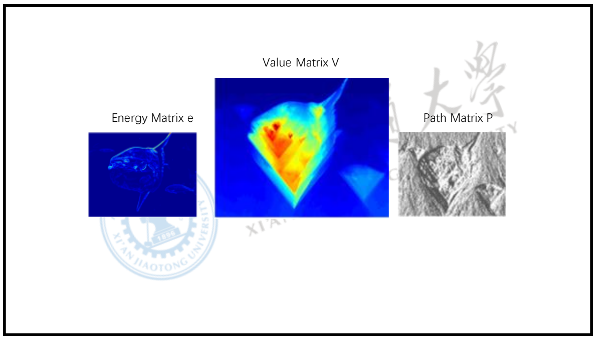


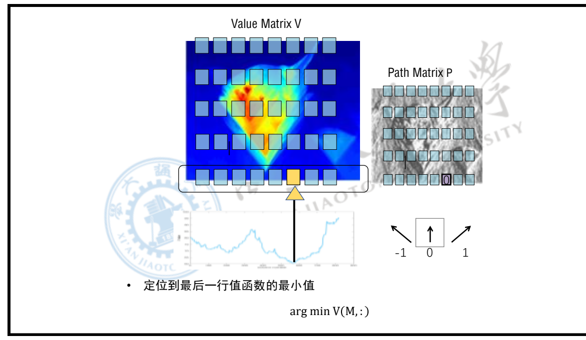
- 逐行迭代，并更新值函数矩阵

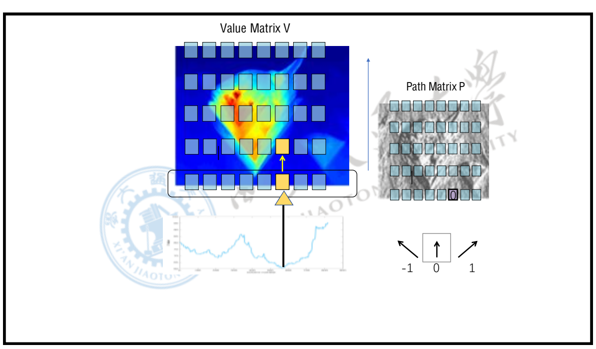
$$V(2,j) = e(2+j) + \min\big(V(1,j-1), V(1,j), V(1,j+1)\big)$$

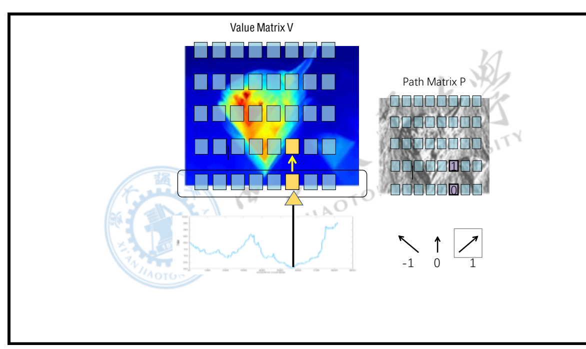
- 设置路径函数

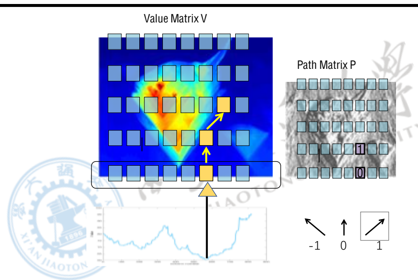
$$P(2,j) = \mathrm{argmin}\big(V(1,j-1), V(1,j), V(1,j+1)\big)$$

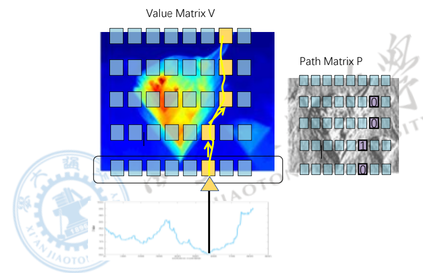

Energy Matrix e
Value Matrix V
Path Matrix P

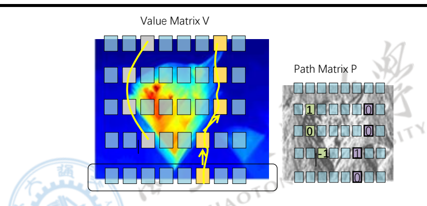

Value Matrix V
Path Matrix P
-1
0
1
• 定位到最后一行值函数的最小值
arg min V(M, :)


Value Matrix V
Path Matrix P
-1
0
1


Value Matrix V
Path Matrix P
-1
0
1

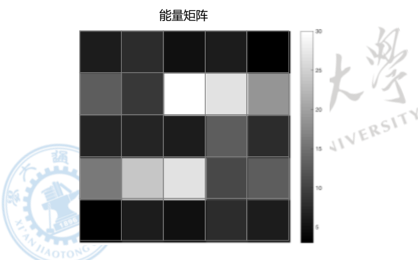


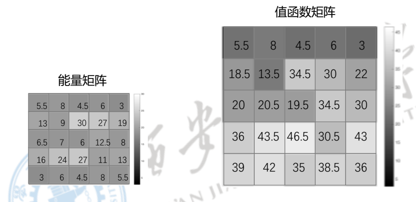




我们获得的V是一个和原始图像一样大小的矩阵，记录着初始行到当前元素的最短路径的能量值，因此我们可以通过它回复任意的最短路径。

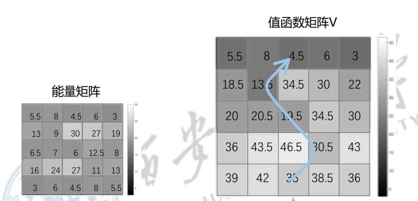
一个合成的例子





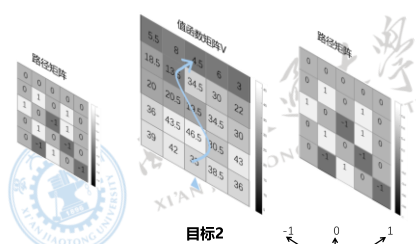
目标:

- 根据**能量矩阵**来计算**值函数和路径矩阵**
- 值矩阵V是一个和原始图像一样大小的矩阵，记录着**初始行到当前元素**的最短路径的能量值



目标1:

- 根据**能量矩阵**来计算**值函数矩阵**
- 值矩阵V是一个和原始图像一样大小的矩阵，记录着**初始行到当前元素**的最短路径的能量值
- 性质：每个位置都记录着从起始行开始的最短路径的能量值



目标2

- 构建**路径矩阵**
- 路径矩阵记录着最短路径上，当前节点的上一个节点

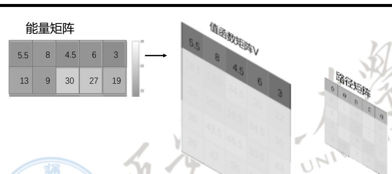
e: 能量矩阵

5.5	8	4.5	6	3
13	9	30	27	19
6.5	7	6	12.5	8
16	24	27	11	13
3	6	4.5	8	5.5

能量矩阵

- 记录每一个像素点的能量值
- 通常使用图像梯度的幅值

有了能量矩阵之后，如何生成我们的两个矩阵呢？

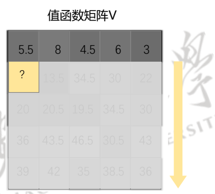


Step1：初始化两个矩阵

- 将值函数和路径矩阵设置成和能量矩阵的***同样大小***
- 将值矩阵的第一行初始化为***能量矩阵的第一行***
- 将路径矩阵的第一行均***初始化为0***

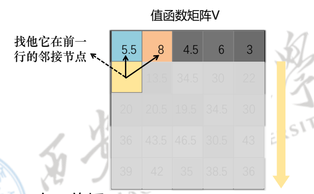
Step2：向下传播

- 从***第二行开始***，一行一行的向下传播



Step2：向下传播

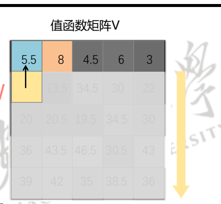
- 从***第二行开始***，一行一行的向下传播



Step2：向下传播

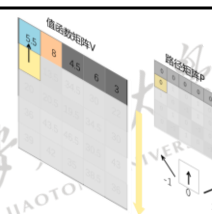
- 从***第二行开始***，一行一行的向下传播
- 找到它在前一行的***邻接节点***

$$V(2,1) = \min\begin{pmatrix} V(1,1) & 5.5 \ \checkmark \\ V(1,2) & 8 \end{pmatrix} + e(2,1)$$



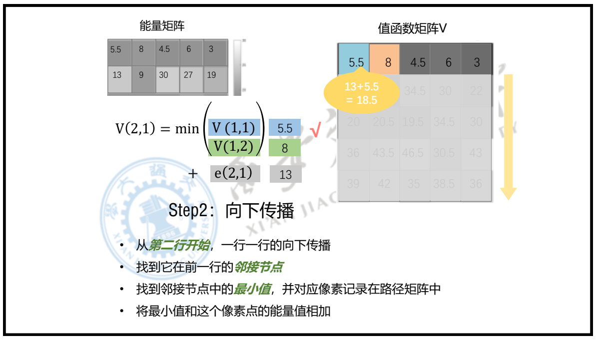
Step2：向下传播

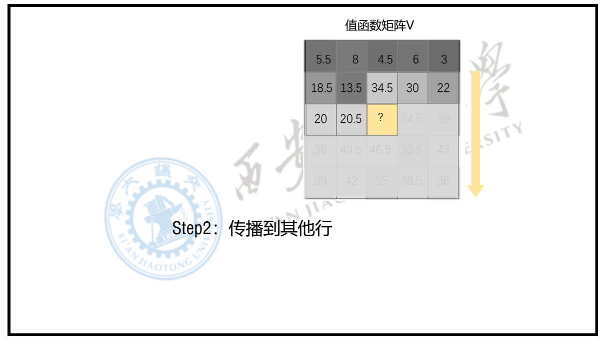
- 从***第二行开始***，一行一行的向下传播
- 找到它在前一行的***邻接节点***
- 找到邻接节点中的***最小值***

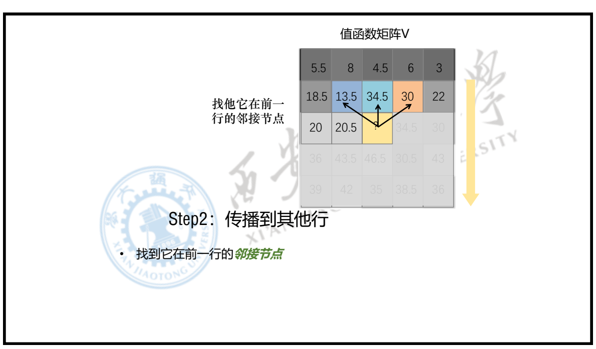


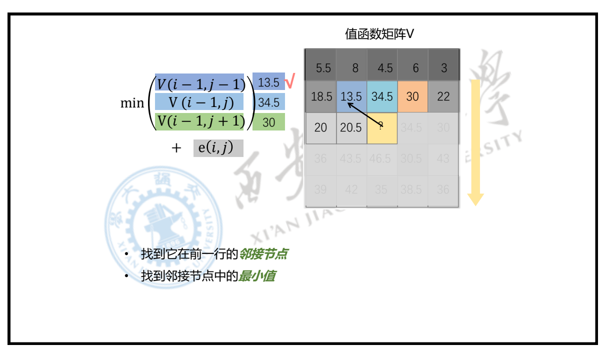
Step2：向下传播

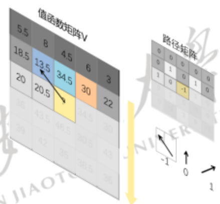
- 从***第二行开始***，一行一行的向下传播
- 找到它在前一行的***邻接节点***
- 找到邻接节点中的***最小值***，并对应像素记录在路径矩阵中


能量矩阵
5.5 8 4.5 6 3
13 9 30 27 19
值函数矩阵V
5.5 8 4.5 6 3
13+5.5 = 18.5
V(2,1) = min V(1,1) 5.5 √
V(1,2) 8
+ e(2,1) 13
Step2：向下传播
从第二行开始，一行一行的向下传播
找到它在前一行的邻接节点
找到邻接节点中的最小值，并对应像素记录在路径矩阵中
将最小值和这个像素点的能量值相加

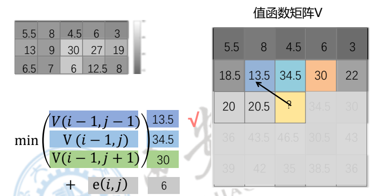

值函数矩阵V
5.5 8 4.5 6 3
18.5 13.5 34.5 30 22
20 20.5 ?
Step2：传播到其他行


值函数矩阵V
5.5 8 4.5 6 3
18.5 13.5 34.5 30 22
20 20.5
找他它在前一行的邻接节点
Step2：传播到其他行
找到它在前一行的邻接节点

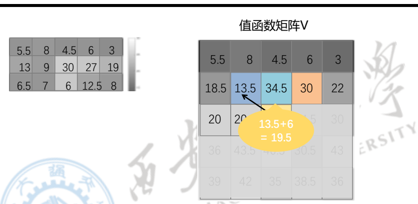

值函数矩阵V
5.5 8 4.5 6 3
18.5 13.5 34.5 30 22
20 20.5 ?
min V(i − 1, j − 1) 13.5 √
V(i − 1, j) 34.5
V(i − 1, j + 1) 30
+ e(i, j)
找到它在前一行的邻接节点
找到邻接节点中的最小值



- 找到它在前一行的***邻接节点***
- 找到邻接节点中的***最小值***，并对应像素记录在路径矩阵中

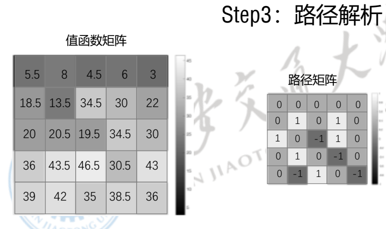


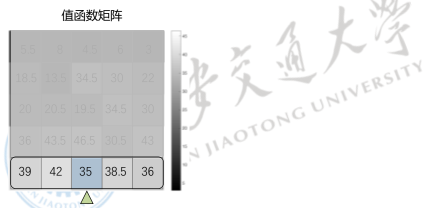
- 找到它在前一行的***邻接节点***
- 找到邻接节点中的***最小值***，并对应像素记录在路径矩阵中
- 将最小值和这个像素点的能量值相加



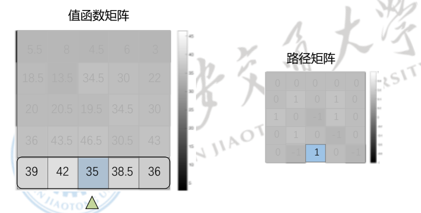
- 找到它在前一行的***邻接节点***
- 找到邻接节点中的***最小值***，并对应像素记录在路径矩阵中
- 将最小值和这个像素点的能量值相加

Step3：路径解析

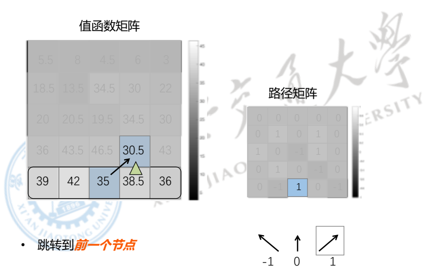




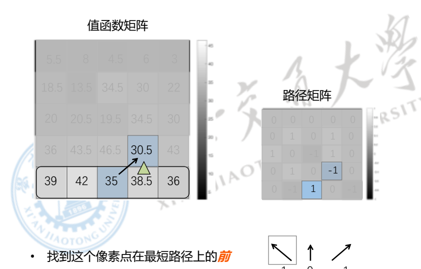
- 找到最后一行的***最小元素***
- 找到这个像素点在最短路径上的***前一个节点***



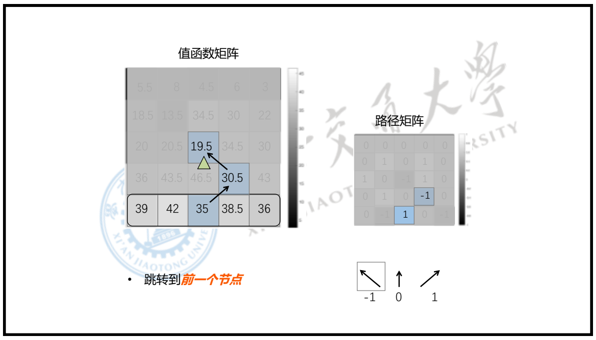
- 找到最后一行的***最小元素***
- 找到这个像素点在最短路径上的***前一个节点***

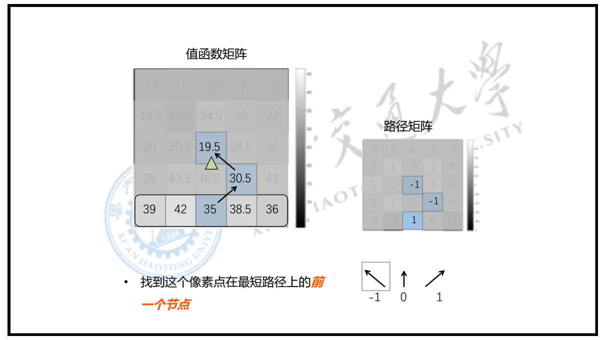


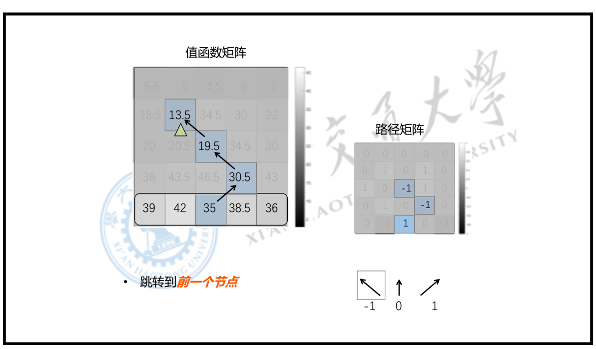
- 跳转到***前一个节点***

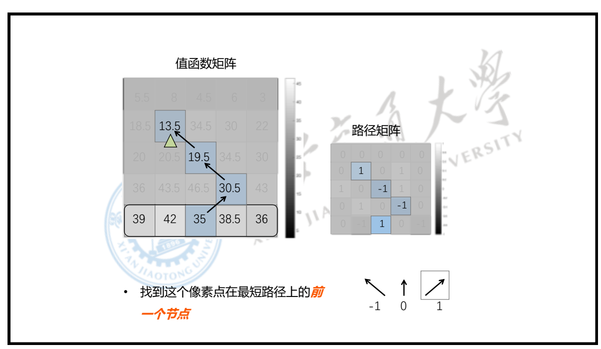


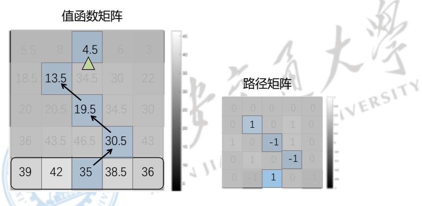
- 找到这个像素点在最短路径上的***前一个节点***

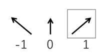

值函数矩阵
路径矩阵
19.5
30.5
39 42 35 38.5 36
• 跳转到前一个节点
-1 0 1


值函数矩阵
路径矩阵
19.5
30.5
39 42 35 38.5 36
• 找到这个像素点在最短路径上的前一个节点
-1 0 1

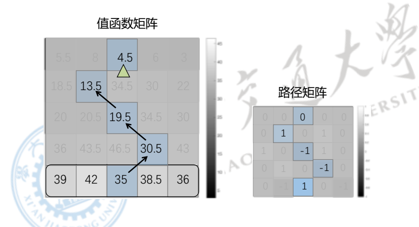

值函数矩阵
路径矩阵
13.5
19.5
30.5
39 42 35 38.5 36
• 跳转到前一个节点
-1 0 1


值函数矩阵
路径矩阵
13.5
19.5
30.5
39 42 35 38.5 36
• 找到这个像素点在最短路径上的前一个节点
-1 0 1

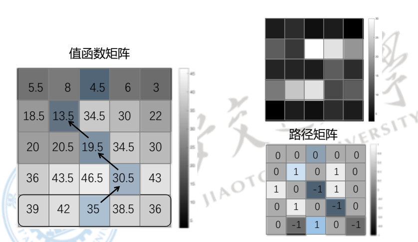




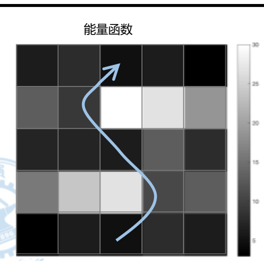
- 直至到达第一行



- Step4：Seam carving是将这个最短路径删除掉



- Step4：Seam carving是将这个最短路径删除掉



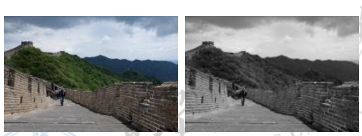
- Step4：Seam carving是将这个最短路径删除掉

Carved能量函数矩阵

- Step4： Seam carving是将这个最短路径删除掉

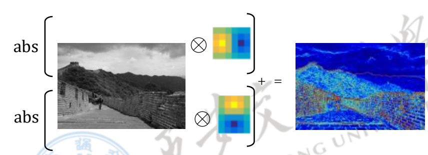
看看一个真实的示例

X方向的Seam



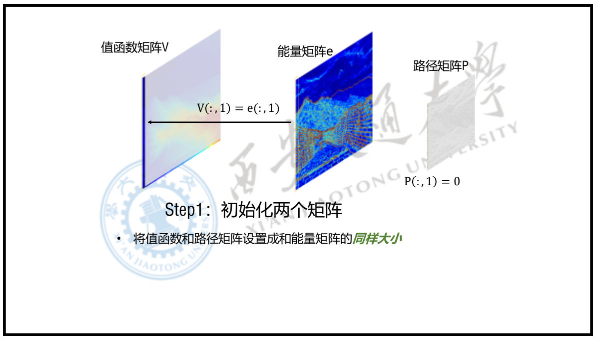
Step0：准备能量函数

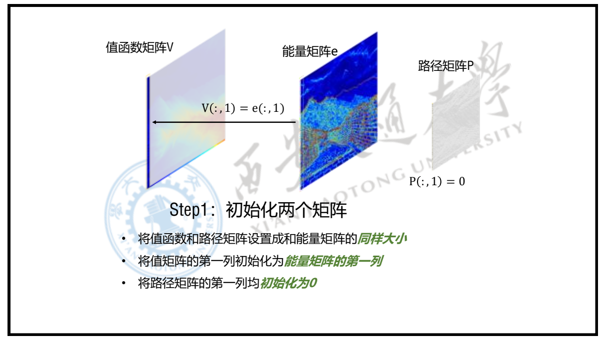
- 将彩色图像转化为灰度图像，Im = rgb2gray(Im)
- 计算图像的梯度值
- 采用梯度的L1或者L2范数来计算能量矩阵

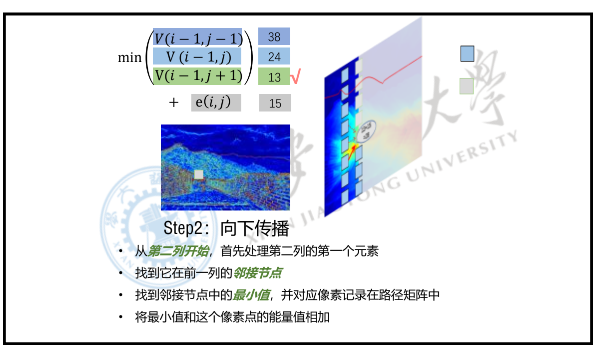


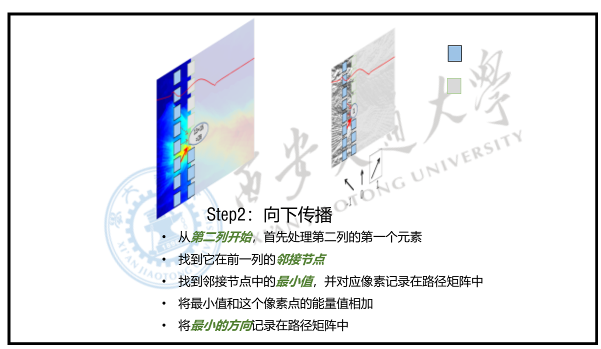
Step0：准备能量函数

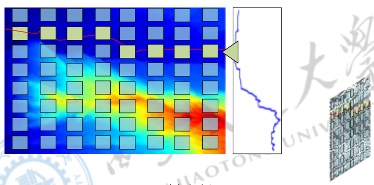
- 将彩色图像转化为灰度图像，Im = rgb2gray(Im)
- 计算图像的梯度值
- 采用梯度的L1或者L2范数来计算能量矩阵


值函数矩阵V
能量矩阵e
路径矩阵P
V(:, 1) = e(:, 1)
P(:, 1) = 0
Step1：初始化两个矩阵
• 将值函数和路径矩阵设置成和能量矩阵的同样大小


值函数矩阵V
能量矩阵e
路径矩阵P
V(:, 1) = e(:, 1)
P(:, 1) = 0
Step1：初始化两个矩阵
• 将值函数和路径矩阵设置成和能量矩阵的同样大小
• 将值矩阵的第一列初始化为能量矩阵的第一列
• 将路径矩阵的第一列均初始化为0

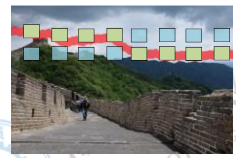

min(V(i − 1, j − 1), V(i − 1, j), V(i − 1, j + 1))
38
24
13 √
+ e(i, j)
15
Step2：向下传播
• 从第二列开始，首先处理第二列的第一个元素
• 找到它在前一列的邻接节点
• 找到邻接节点中的最小值，并对应像素记录在路径矩阵中
• 将最小值和这个像素点的能量值相加


Step2：向下传播
• 从第二列开始，首先处理第二列的第一个元素
• 找到它在前一列的邻接节点
• 找到邻接节点中的最小值，并对应像素记录在路径矩阵中
• 将最小值和这个像素点的能量值相加
• 将最小的方向记录在路径矩阵中



Step3：路径解析

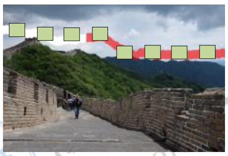
- 找到最后一列的*最小元素*
- 找到这个像素点在最短路径上的*前一个节点*



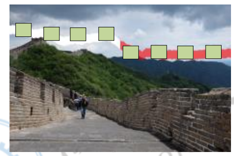
Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

Step4: 删除Seam对应的行



Step4: 删除Seam对应的行



Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

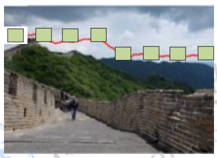
Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

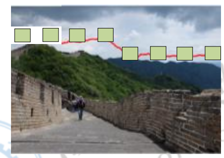
Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

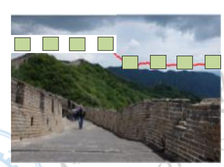


Step4: 删除Seam对应的行

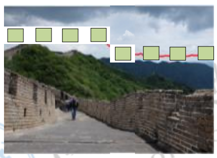


Step4: 删除Seam对应的行

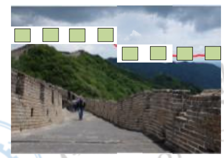
Step4: 删除Seam对应的行



Step4: 删除Seam对应的行



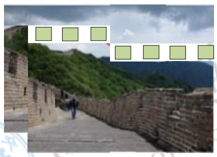
Step4: 删除Seam对应的行



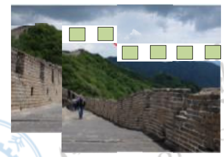
Step4: 删除Seam对应的行

Step4: 删除Seam对应的行

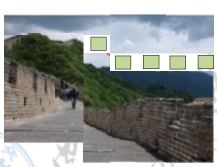
Step4: 删除Seam对应的行



Step4: 删除Seam对应的行

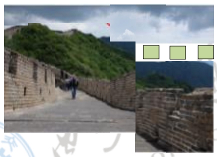


Step4: 删除Seam对应的行

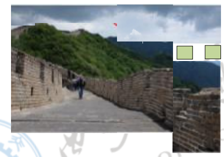


Step4: 删除Seam对应的行

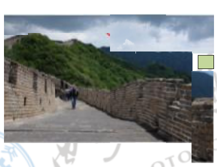
Step4: 删除Seam对应的行



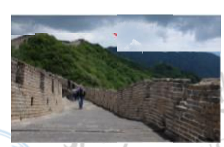
Step4: 删除Seam对应的行



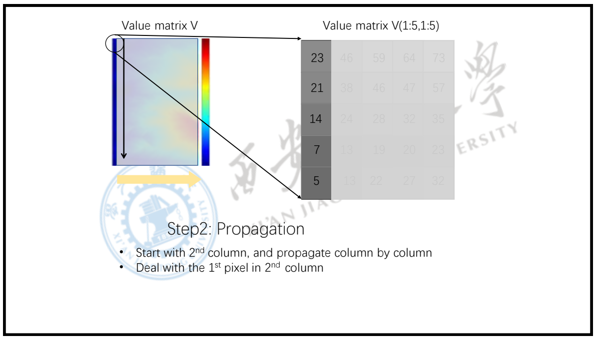
Step4: 删除Seam对应的行



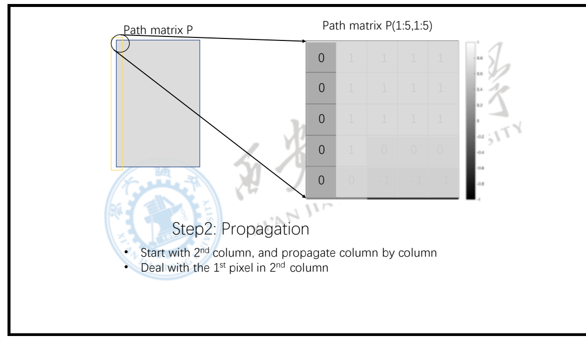
Step4: 删除Seam对应的行



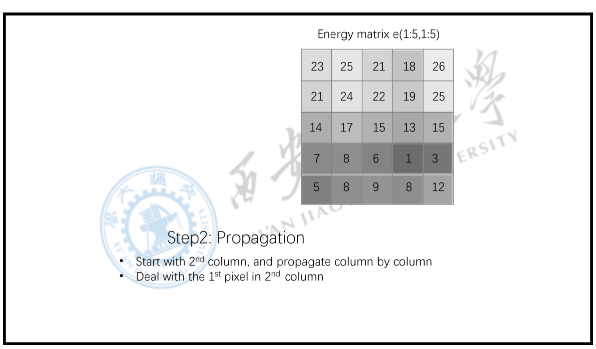
Step4: 删除Seam对应的行



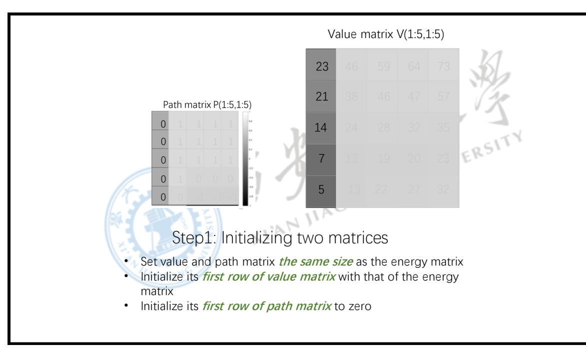
Value matrix V
Value matrix V(1:5,1:5)
23
21
14
7
5
Step2: Propagation
• Start with 2nd column, and propagate column by column
• Deal with the 1st pixel in 2nd column



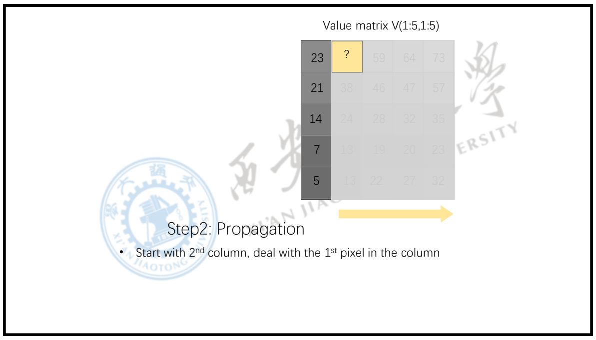
Path matrix P
Path matrix P(1:5,1:5)
0
0
0
0
0
Step2: Propagation
• Start with 2nd column, and propagate column by column
• Deal with the 1st pixel in 2nd column

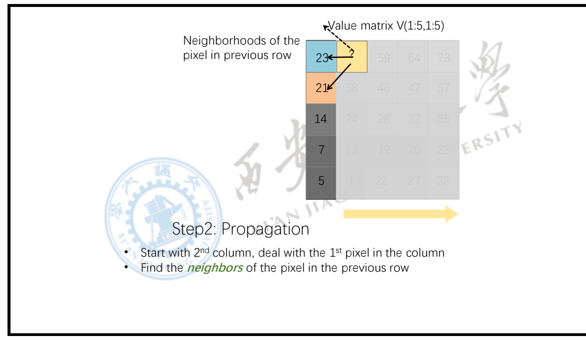


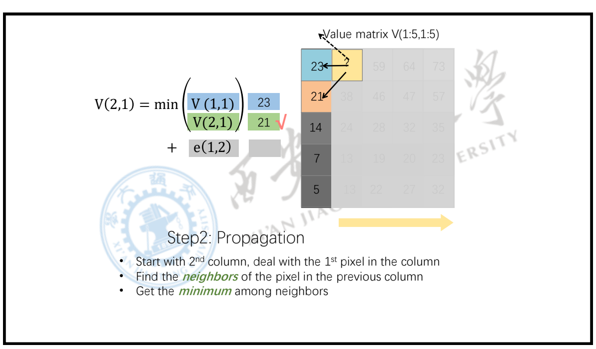
Energy matrix e(1:5,1:5)
23 25 21 18 26
21 24 22 19 25
14 17 15 13 15
7 8 6 1 3
5 8 9 8 12
Step2: Propagation
• Start with 2nd column, and propagate column by column
• Deal with the 1st pixel in 2nd column

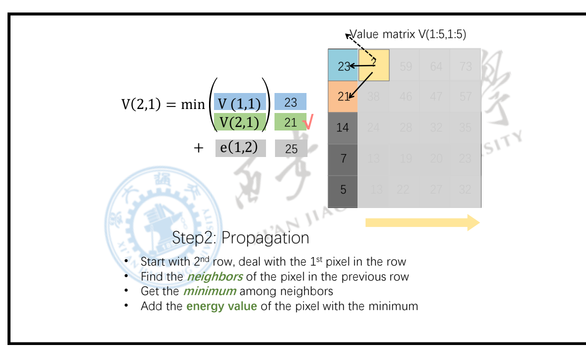


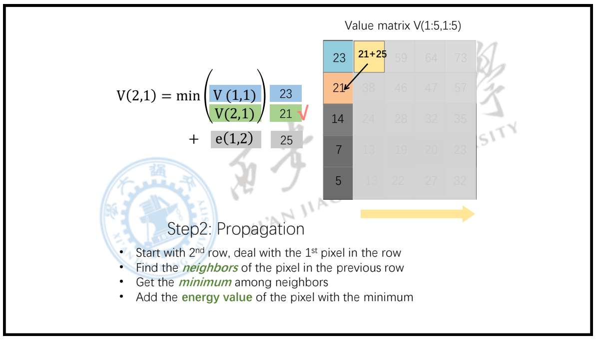
Value matrix V(1:5,1:5)
Path matrix P(1:5,1:5)
23
21
14
7
5
Step1: Initializing two matrices
• Set value and path matrix the same size as the energy matrix
• Initialize its first row of value matrix with that of the energy matrix
• Initialize its first row of path matrix to zero

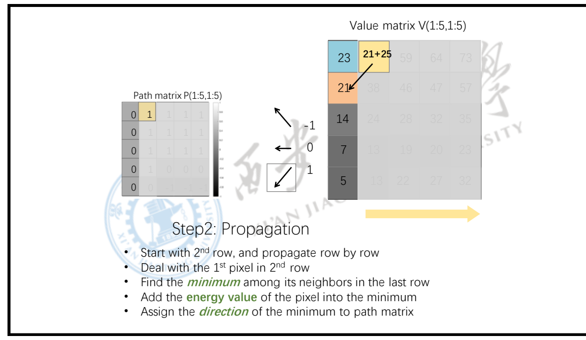

Value matrix V(1:5,1:5)
23
?
21
14
7
5
Step2: Propagation
• Start with 2nd column, deal with the 1st pixel in the column

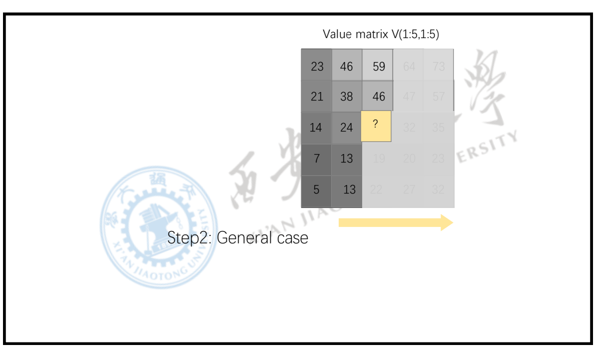

Value matrix V(1:5,1:5)
Neighborhoods of the pixel in previous row
23
21
14
7
5
Step2: Propagation
• Start with 2nd column, deal with the 1st pixel in the column
• Find the neighbors of the pixel in the previous row

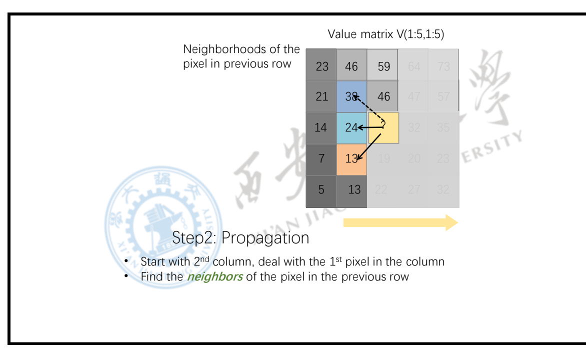

Value matrix V(1:5,1:5)
V(2,1) = min V(1,1) V(2,1)
23
21 √
+ e(1,2)
23
21
14
7
5
Step2: Propagation
• Start with 2nd column, deal with the 1st pixel in the column
• Find the neighbors of the pixel in the previous column
• Get the minimum among neighbors

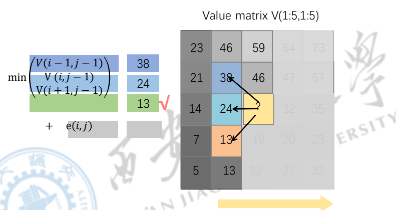

Value matrix V(1:5,1:5)
V(2,1) = min V(1,1) V(2,1)
23
21 √
+ e(1,2)
25
23
21
14
7
5
Step2: Propagation
• Start with 2nd row, deal with the 1st pixel in the row
• Find the neighbors of the pixel in the previous row
• Get the minimum among neighbors
• Add the energy value of the pixel with the minimum


Value matrix V(1:5,1:5)
V(2,1) = min V (1,1) 23 V(2,1) 21 √ + e(1,2) 25
21+25
Step2: Propagation
• Start with 2nd row, deal with the 1st pixel in the row
• Find the neighbors of the pixel in the previous row
• Get the minimum among neighbors
• Add the energy value of the pixel with the minimum


Value matrix V(1:5,1:5)
Path matrix P(1:5,1:5)
-1
0
1
21+25
Step2: Propagation
• Start with 2nd row, and propagate row by row
• Deal with the 1st pixel in 2nd row
• Find the minimum among its neighbors in the last row
• Add the energy value of the pixel into the minimum
• Assign the direction of the minimum to path matrix

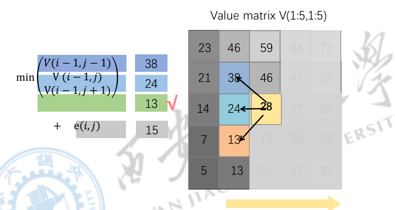

Value matrix V(1:5,1:5)
Step2: General case


Value matrix V(1:5,1:5)
Neighborhoods of the pixel in previous row
Step2: Propagation
• Start with 2nd column, deal with the 1st pixel in the column
• Find the neighbors of the pixel in the previous row



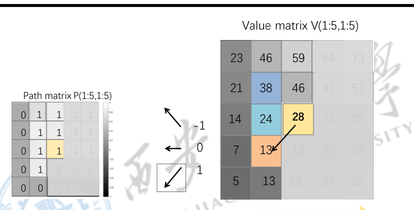
Step2: Propagation

- Start with 2nd column, deal with the 1st pixel in the column
- Find the *neighbors* of the pixel in the previous column
- Get the *minimum* among neighbors



Step2: Propagation

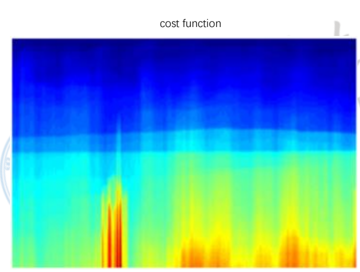
- Start with 2nd row, deal with the 1st pixel in the row
- Find the *neighbors* of the pixel in the previous row
- Get the *minimum* among neighbors
- Add the **energy value** of the pixel with the minimum



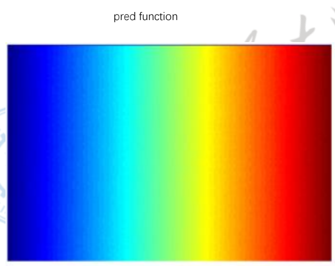
Step2: Propagation

- Start with 2nd row, and propagate row by row
- Deal with the 1st pixel in 2nd row
- Find the *minimum* among its neighbors in the last row
- Add the **energy value** of the pixel into the minimum
- Assign the *direction* of the minimum to path matrix

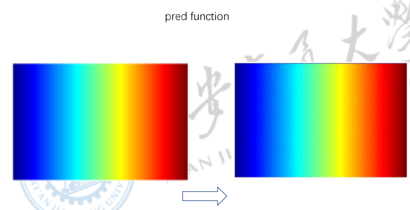
Updating until the end of the image



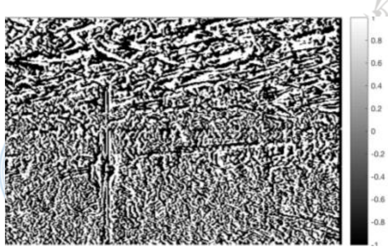
Updating until the end of the image

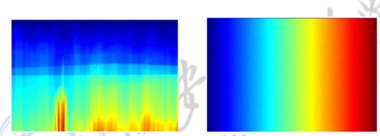


Updating until the end of the image

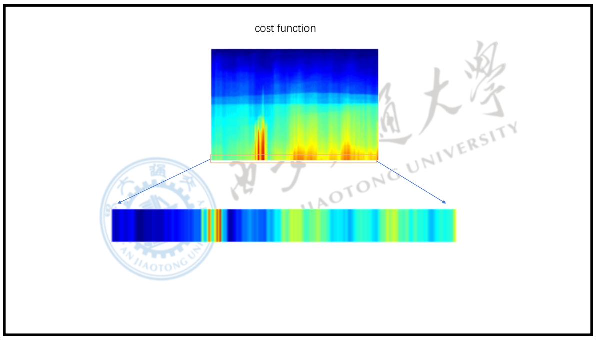


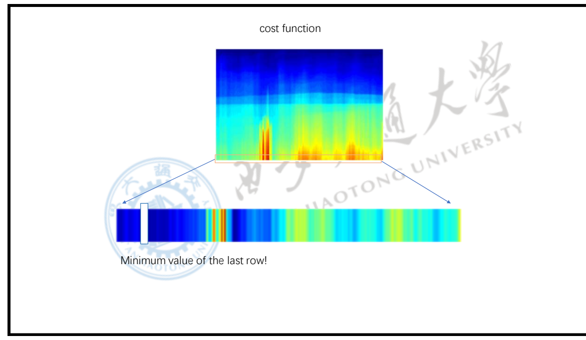
Updating until the end of the image

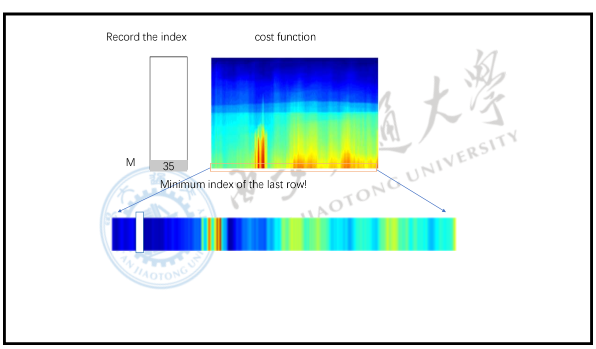


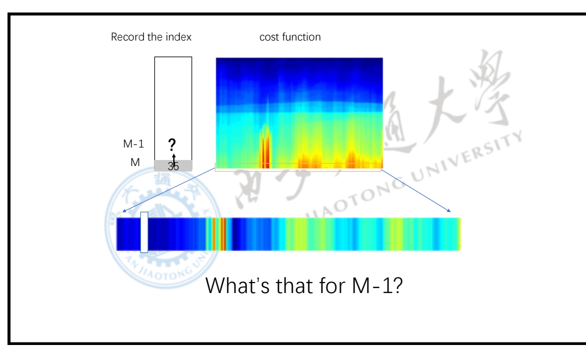


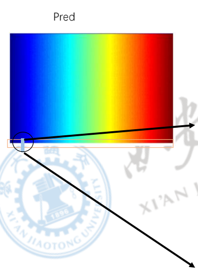
How to find the seam?


cost function

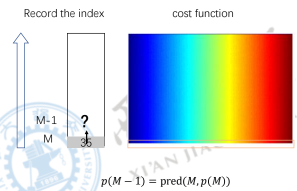

cost function
Minimum value of the last row!


Record the index
cost function
M
35
Minimum index of the last row!


Record the index
cost function
M-1
?
M
35
What's that for M-1?



Resolving the seam



Returning to the original problem

Image Energy Functions

- The entropy energy
  - Computes the entropy over a 9x9 window and adds it to e1.
- The segmentation method
  - first segments the image [Christoudias et al. 2002]
  - and then applies the e1 error norm on the results,
  - effectively leaving only the edges between segments.
- eHoG is defined as follows:

$$e_{HoG}(\mathbf{I}) = \frac{|\frac{\partial}{\partial x}\mathbf{I}| + |\frac{\partial}{\partial y}\mathbf{I}|}{\max\left(HoG(\mathbf{I}(x,y))\right)},$$

- where HoG(I(x,y))
  - is taken to be a **histogram of oriented gradients** at every pixel [Dalal and Triggs 2005].
  - use an 8-bin histogram computed over a 11x11 window around the pixel.
- found either e1 or eHoG to work quite well.

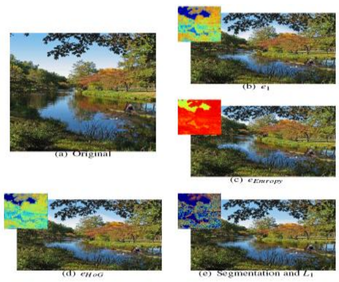


Figure 4: Comparing different energy functions for content aware resizing.

Discrete Image Resizing

- Aspect Ratio Change
  - given image I from nxm to nxm'
  - where m−m' = c
  - be achieved simply by
    - successively removing c vertical seams from I. (Figure 5).
  - can also be achieved by
    - increasing the number of column (Figure 6).
    - The added value of such an approach is that
      - it does not remove any information from the image.

Figure 5: Comparing aspect ratio change. From left to right in the bottom: the image resized using seam removals, scaling and cropping.

Figure 6: Aspect ratio change of pictures of the Janapese master Utagawa Hiroshige, by seam insertion.

Retargeting with Optimal Seams-Order

- Image retargeting
  - generalizes aspect ratio
    - change from one dimension to two dimensions
  - such that an image I of size nxm
    - will be retargeted to size n' xm' and,
    - assume that m' < m and n' < n
- what is the correct order of seam carving?
  - remove vertical seams first?
  - horizontal seams first?
  - or alternate between the two?

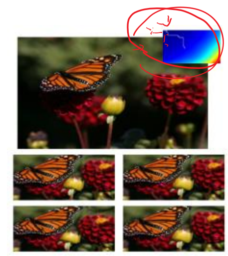


Figure 7: Optimal order retargeting: On the top is the original image and its transport map **T**. Given a target size, we follow the optimal path (white path on **T**) to obtain the retargeted image (top row, right). For comparison we show retargeting results by alternating between horizontal and vertical seam removal (top row, left), removing vertical seams first (bottom row, left), and removing hor-

Image Enlarging

- denote $I^{(t)}$ as
  - the smaller image created after t seam have been removed from I.
- denote $I^{(-1)}$ as
  - the larger image created after 1 seam have been enlarged from I
  - compute the optimal vertical (horizontal) seam s on I
  - and duplicate the pixels of s by averaging them with their left and right neighbors (top and bottom in the horizontal case).
- denote $I^{(-k)}$ as
  - enlarge an image by k,
  - find the first k seams for removal,
  - and duplicate them in order to arrive at $I^{(-k)}$

Image Enlarging

- To continue in content-aware fashion for excessive image enlarging (for instance, greater than 50%),
  - break the process into several steps.
  - Each step does not enlarge the size of the image in more than a fraction of its size from the previous step,
  - essentially guarding the important content from being stretched.
  - Nevertheless, extreme enlarging of an image would most probably produce noticeable artifacts (Figure 8 (f)).

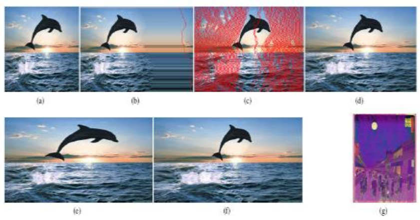


Figure 8: Seam insertion: finding and inserting the optimum seam on an enlarged image will most likely insert the same seam again and again as in (b). Inserting the seams in order of removal (c) achieves the desired 50% enlargement (d). Using two steps of seam insertions of 50% in (f) achieves better results than scaling (e). In (g), the seams inserted to expand figure 6 are shown.

Content Amplification

- **Content Amplification**
  - amplify the content of the image while preserving its size
    - be achieved by combining seam carving and scaling.
  - first
    - use standard scaling to enlarge the image and
    - only then apply seam carving on the larger image to carve the image back to its original size (see Figure 9).
  - Note that the pixels removed are in effect sub-pixels of the original image.

Content Amplification

Figure 9: Content amplification. On the right: a combination of seam carving and scaling amplifies the content of the original image (left).

Object Removal

- use a simple user interface for object removal.
  - The user marks the target object to be removed
  - and then seams are removed from the image
  - until all marked pixels are gone.
- The system can automatically
  - calculate the smaller of the vertical or horizontal diameters (in pixels) of the target removal region
  - and perform vertical or horizontal removals accordingly (Figure 11).
- to regain the original size of the image,
  - seam insertion could be employed on the resulting (smaller) image (see Figure 12).

Figure 11: Simple object removal: the user marks a region for removal (green), and possibly a region to protect (red), on the original image (see inset in left image). On the right image, consecutive vertical seam were removed until no 'green' pixels were left.

Limitations

- this method
  - does not work automatically on all images.
  - can be corrected by adding higher level cues, either manual or automatic. Figure 14, Figure 15
- Other times,
  - not even high level information can solve the problem.
- two major factors that limit this seam carving approach.
  - The first
    - is the amount of content in an image.
    - If the image is too condensed,
    - it does not contain 'less important' areas,
    - then any type of content-aware resizing strategy will not succeed.
  - The second type of limitation
    - is the layout of the image content.
    - In certain types of images, albeit not being condensed,the content is laid out in a manner that prevents the seams to bypass important parts (Figure 16).