图像补全

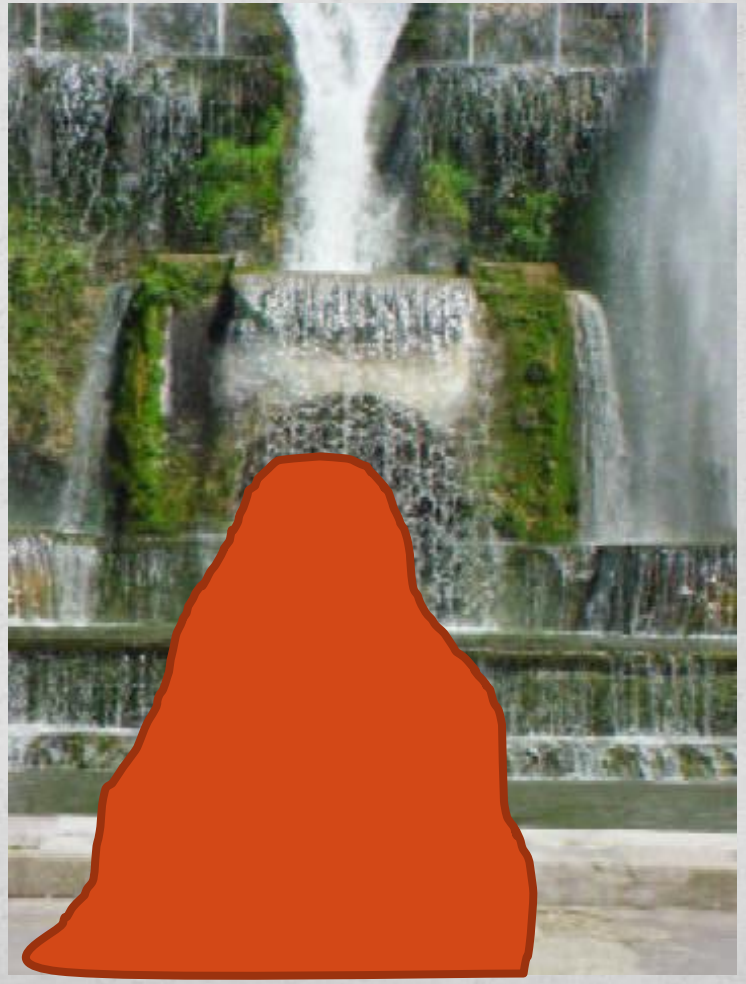
计算机系　　徐昆

提纲

- 图像补全的研究背景
- 基于像素的补全
- 基于块的补全
- 带交互的结构补全

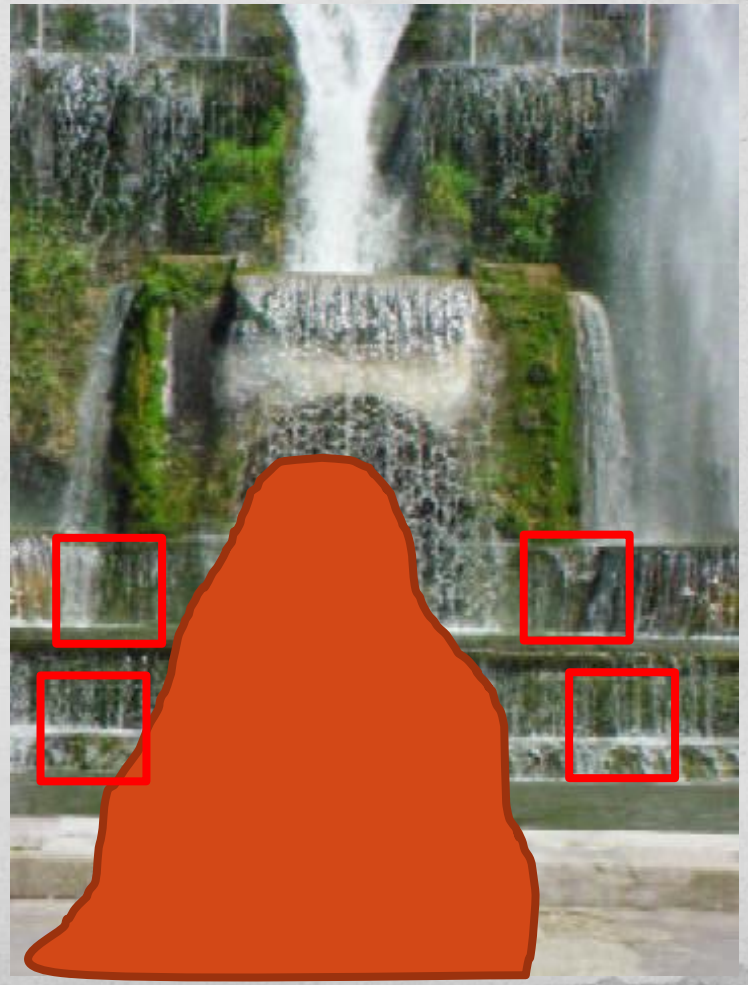
研究背景

- 什么是图像补全?
- 用途
  - 图像修补
  - 图像合成
  - 图像库构建



算法原理

- 图像存在一定的重复性、冗余性
- 利用重复性填补空洞区域
- 关键在于如何合理利用重复性

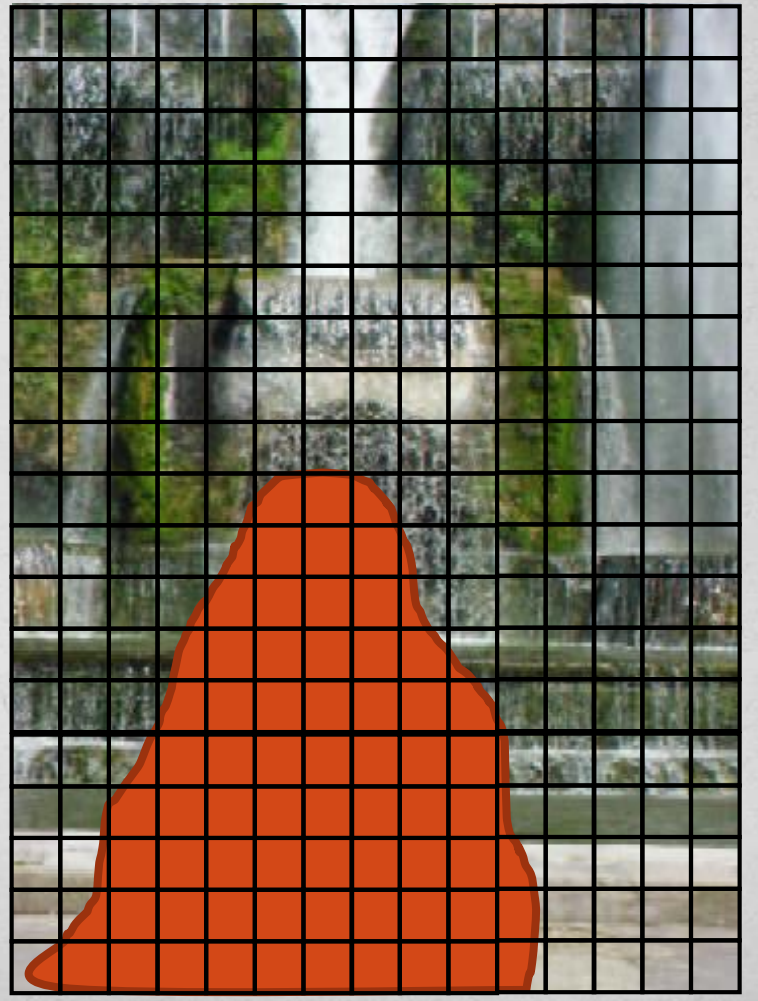


基于像素的图像补全

- Region Filling and Object Removal by Exemplar-Based Image Inpainting. A Criminisi et. al. IEEE Transactions on Image Processing 2004.

算法步骤

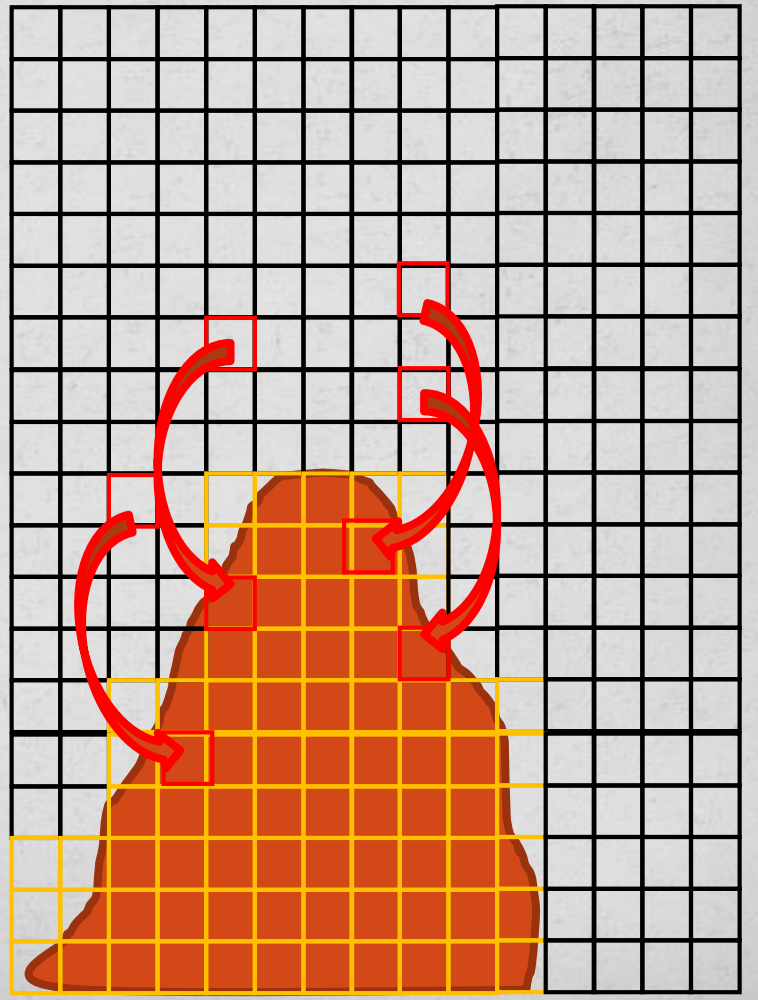
- 标记空洞区域



算法步骤

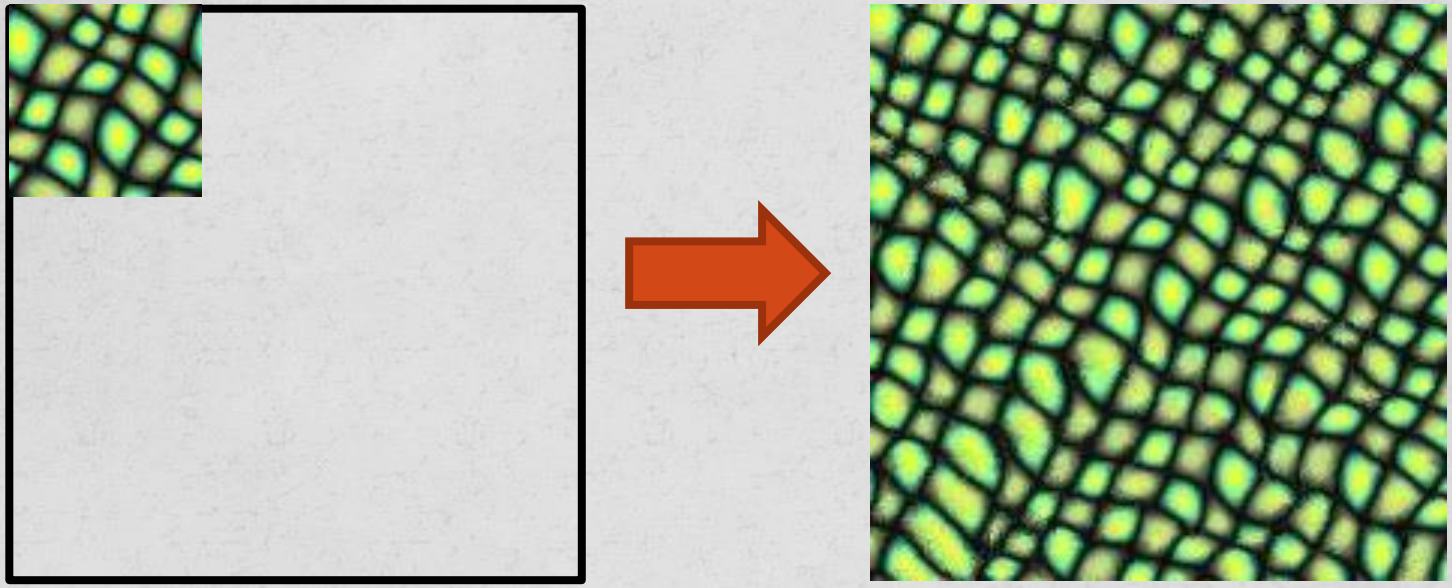
- 标记空洞区域
- 逐个像素填补

- 两个疑问？
  - 如何选择源像素
  - 填补顺序



如何选择源像素

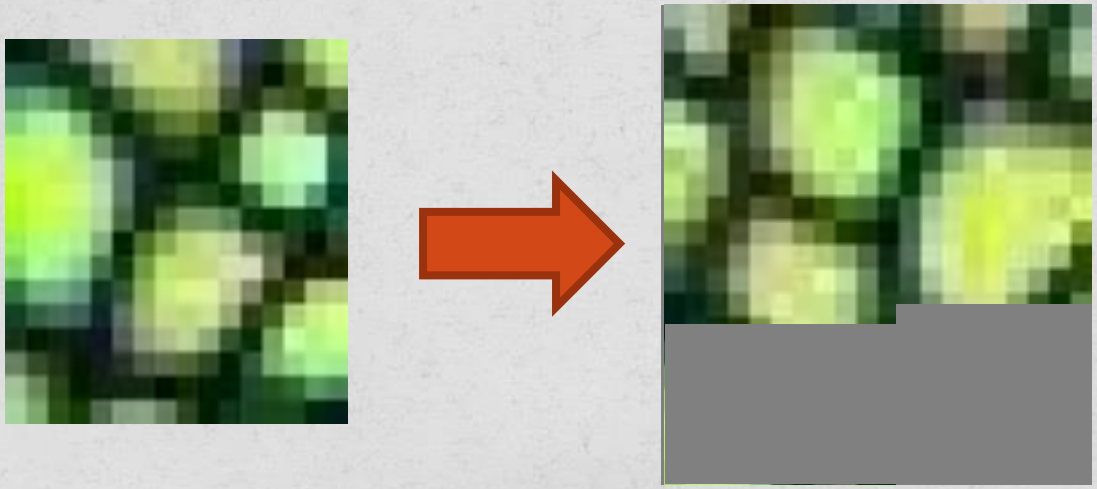
- 考虑一个更加简单的问题 – 纹理补全



- Texture Synthesis by Non-parametric Sampling. Alexei A. Efros and Thomas K. Leung. ICCV 1999.

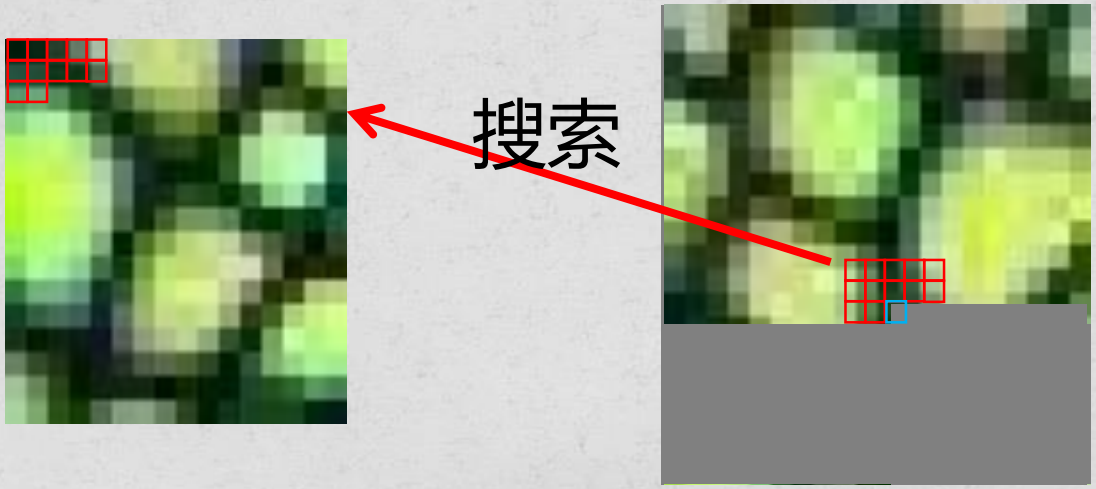
如何选择源像素

- 无需考虑填补顺序
  - 先上后下，先左后右的顺序填补每一个像素



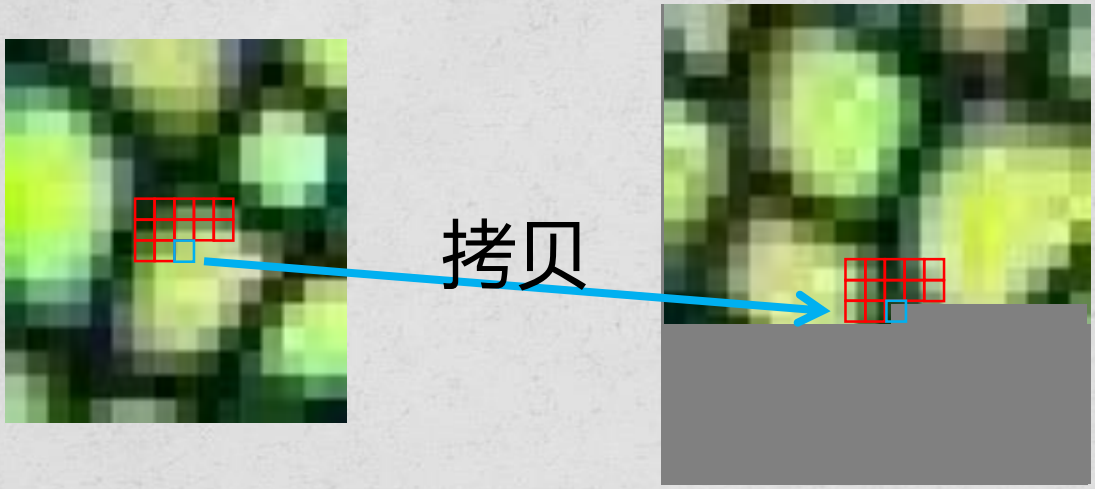
如何选择源像素

- 利用邻域关系
  - 如果邻域相似，那么该像素也会相似
  - 搜索**最匹配的邻域**



如何选择源像素

- 利用邻域关系



  - 拷贝颜色值填充当前的像素

如何选择源像素

- 利用邻域关系

  - 依次填补每一像素点

如何选择源像素

- 利用邻域关系

  - 依次填补每一像素点

如何选择源像素

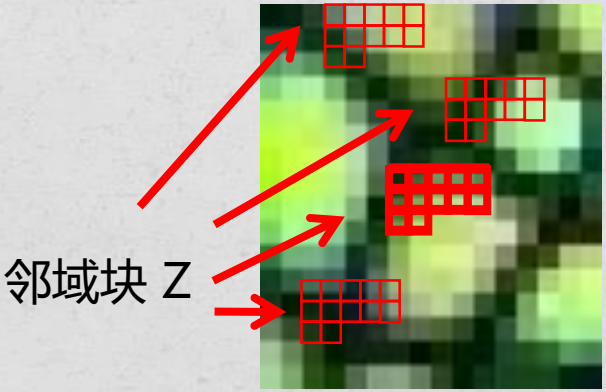
- 利用邻域关系

- 得到最终结果

寻找最匹配的邻域

- 能量函数定义：$E(Z)=\sum_{p}(X_p-Z_p)^2$
- 寻找Z，最小化能量函数：$\min E(Z)$

$E_1=100$
$E_2=130$
$E_3=35$
$E_4=56$

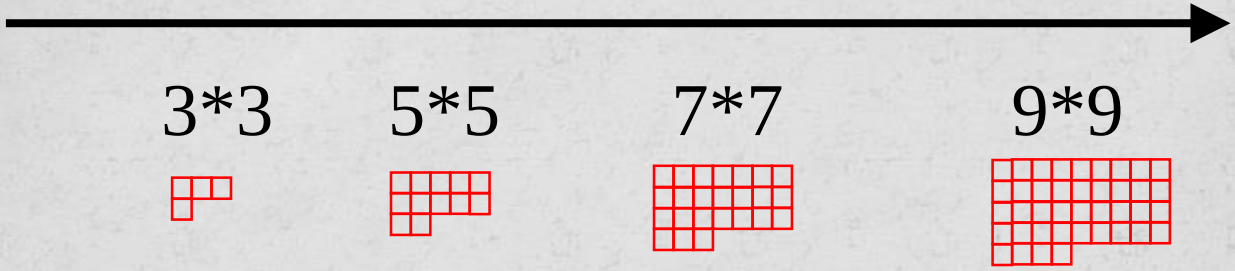


源图像

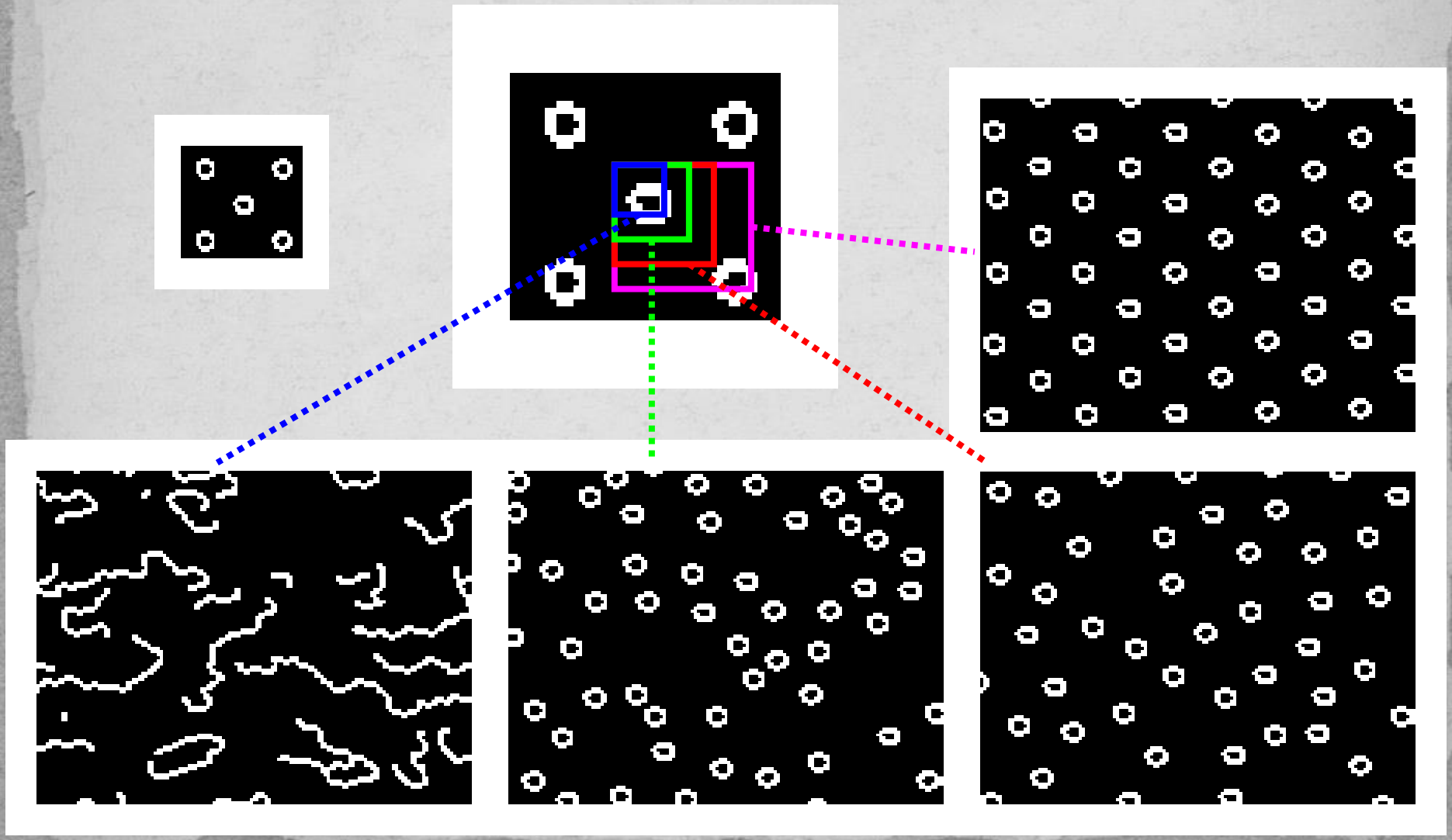
目标图像

邻域窗口大小的影响

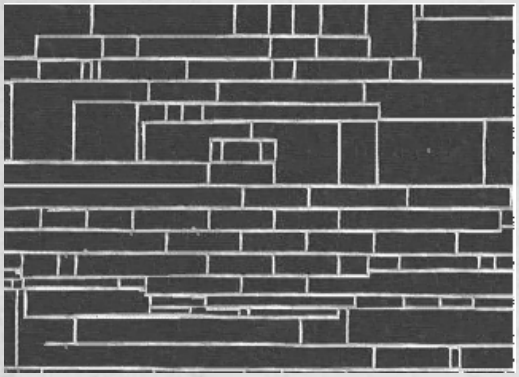
- 邻域窗口大小

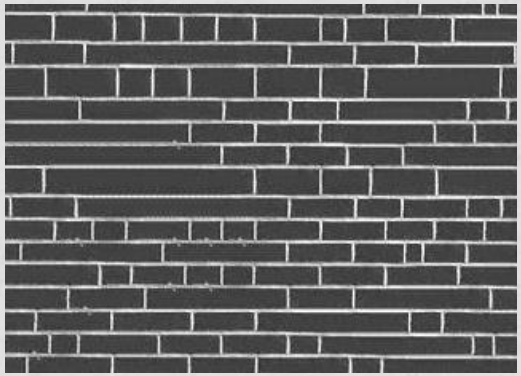


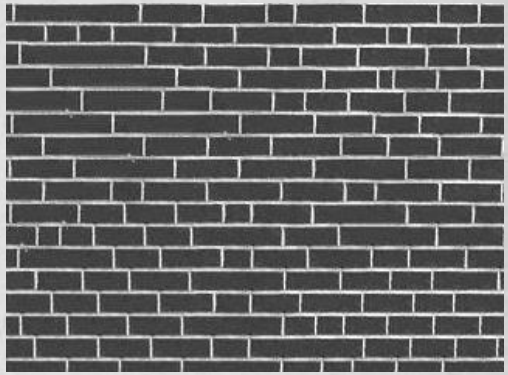
邻域窗口大小的影响



邻域窗口大小的影响

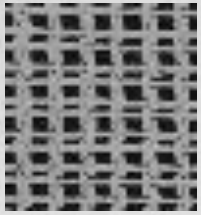


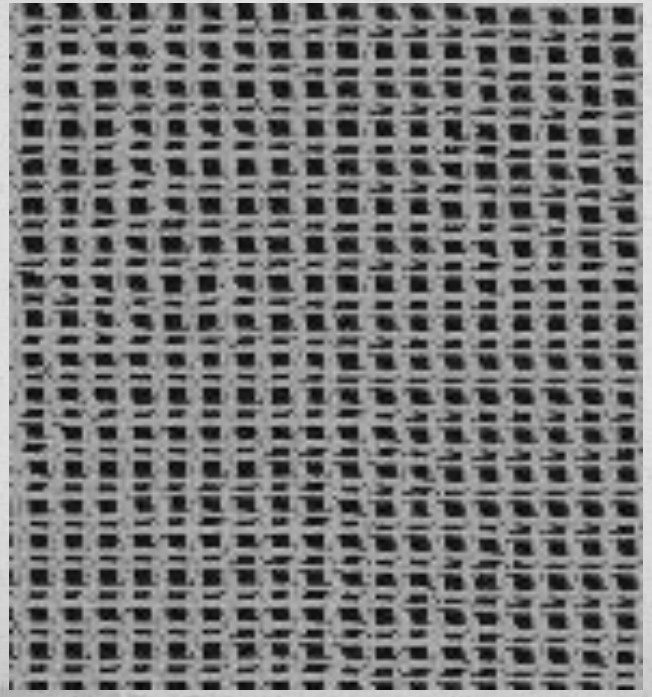


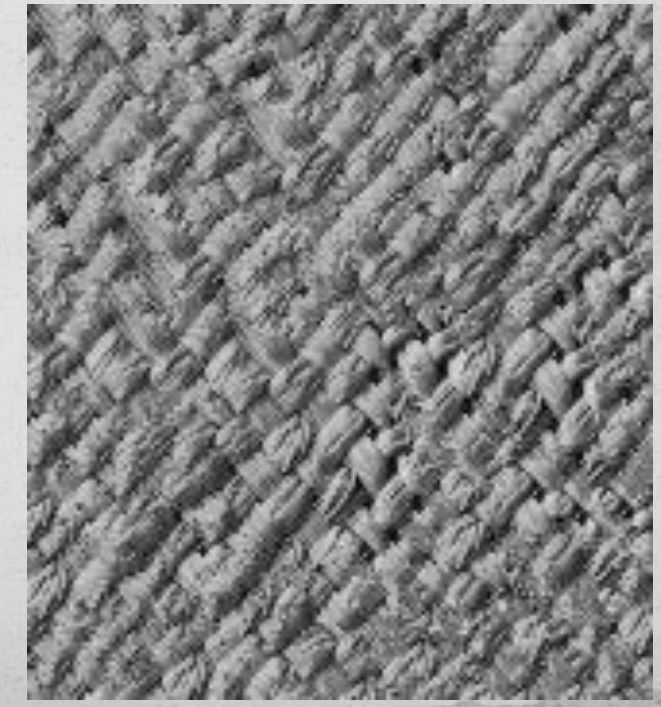


- 结论：窗口太小或太大，都不合适
  - 太小时：无法保持原有结构，结果太随机
  - 太大时：结构保持完整，但重复性太强
  - 合适的大小是**纹理单元大小**

更多结果

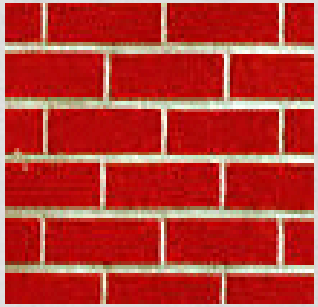


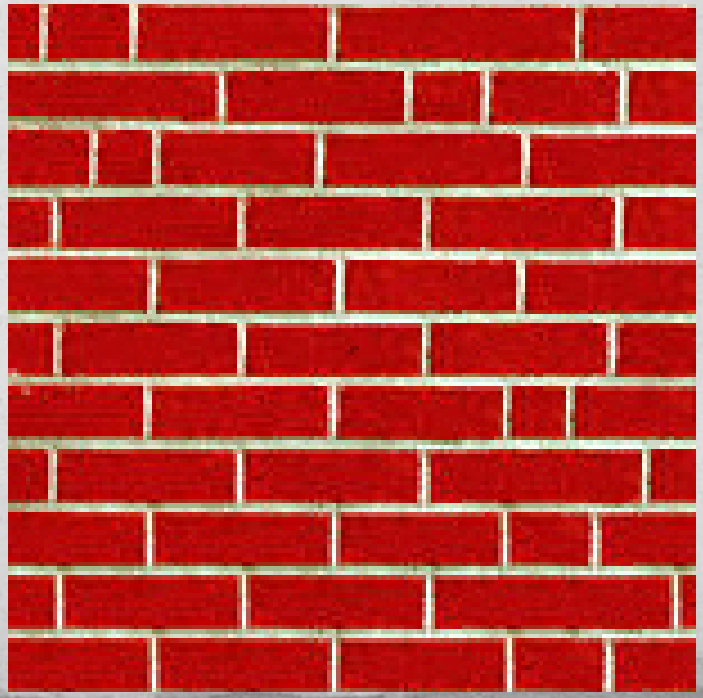




更多结果

更多结果

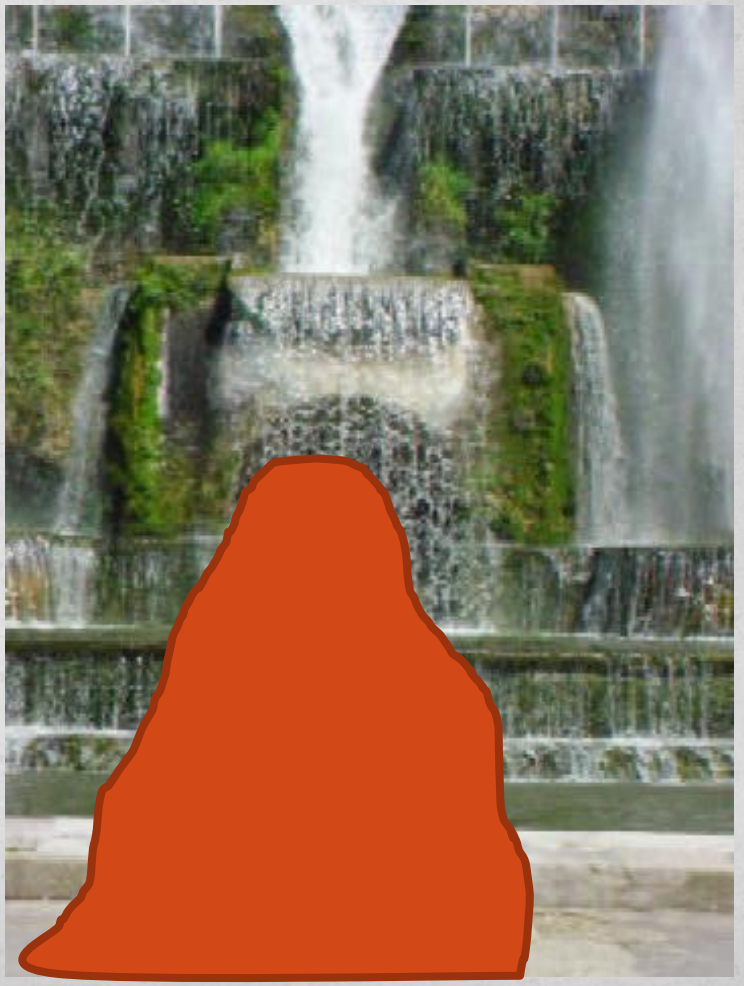




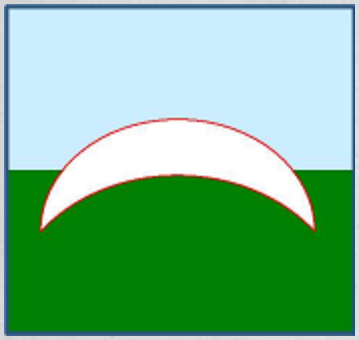
填补顺序

- 回到开始的问题

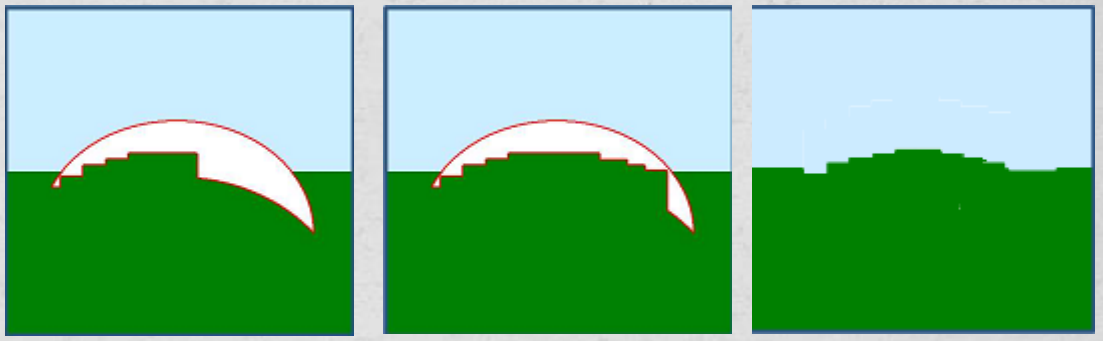
- 填补的顺序有关系吗？



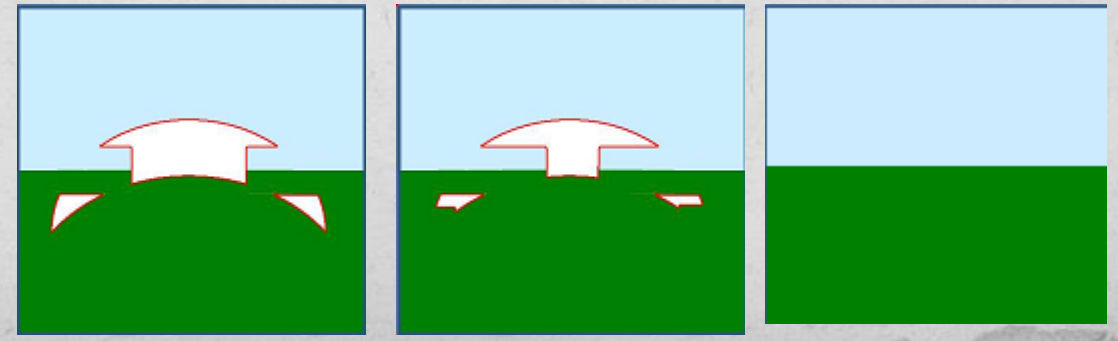
填补顺序–例子



- 按照从下往上填补的顺序

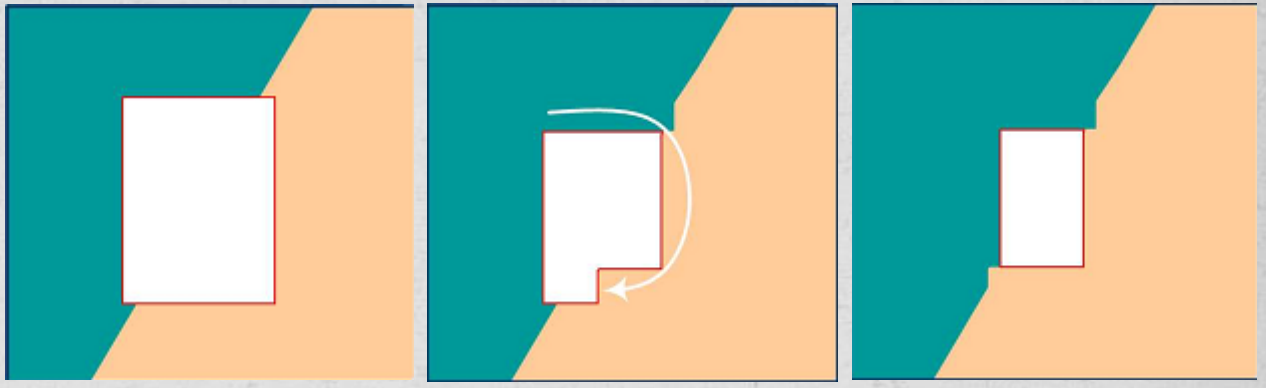


- 按照边缘优先填补的顺序

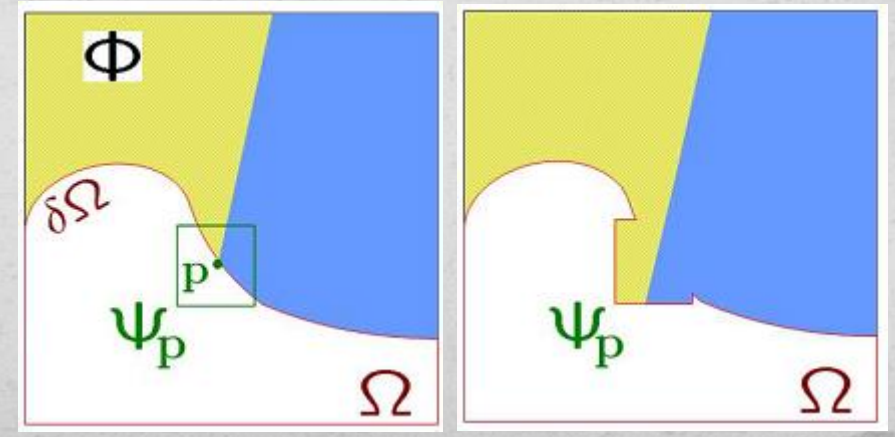


填补顺序-例子2

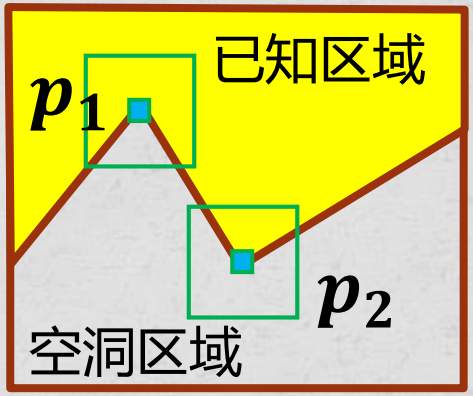
- 顺时针填补顺序



- 结论1
  - 要优先填补边界所在区域(梯度值大的地方)：与图像数据本身相关
  - 这样可以保持住边界区域的结构性



填补顺序-例子3



- 先填补像素p_1还是像素p_2?
  - 先填补p_1。因为它周围的可用邻域更大，可用信息更多。
- 结论2
  - 优先填补周围可用邻域大的地方。

填补顺序 - 算法

- 邻域大小影响

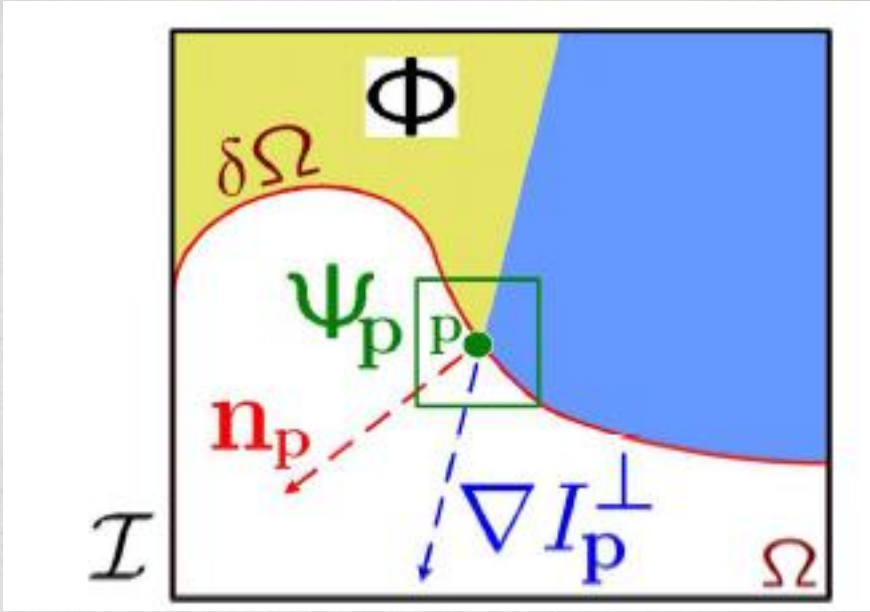
$$C(\mathbf{p}) = \frac{\sum_{\mathbf{q} \in \Psi_\mathbf{p} \cap \bar{\Omega}} C(\mathbf{q})}{|\Psi_\mathbf{p}|}$$

- 数据影响

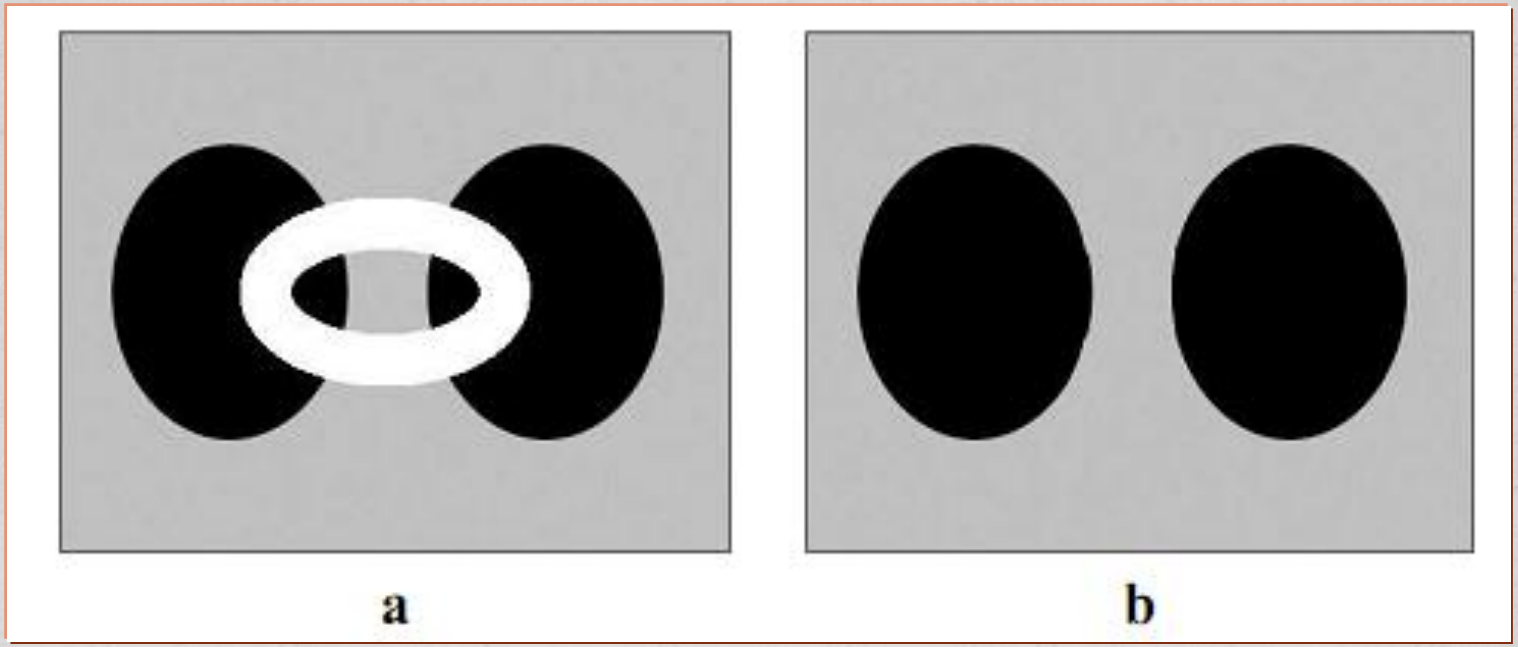
$$D(\mathbf{p}) = \frac{|\nabla I_\mathbf{p}^\perp \cdot \mathbf{n}_\mathbf{p}|}{\alpha}$$

- 填补优先度

$$P(\mathbf{p}) = C(\mathbf{p}) D(\mathbf{p})$$



图像补全结果



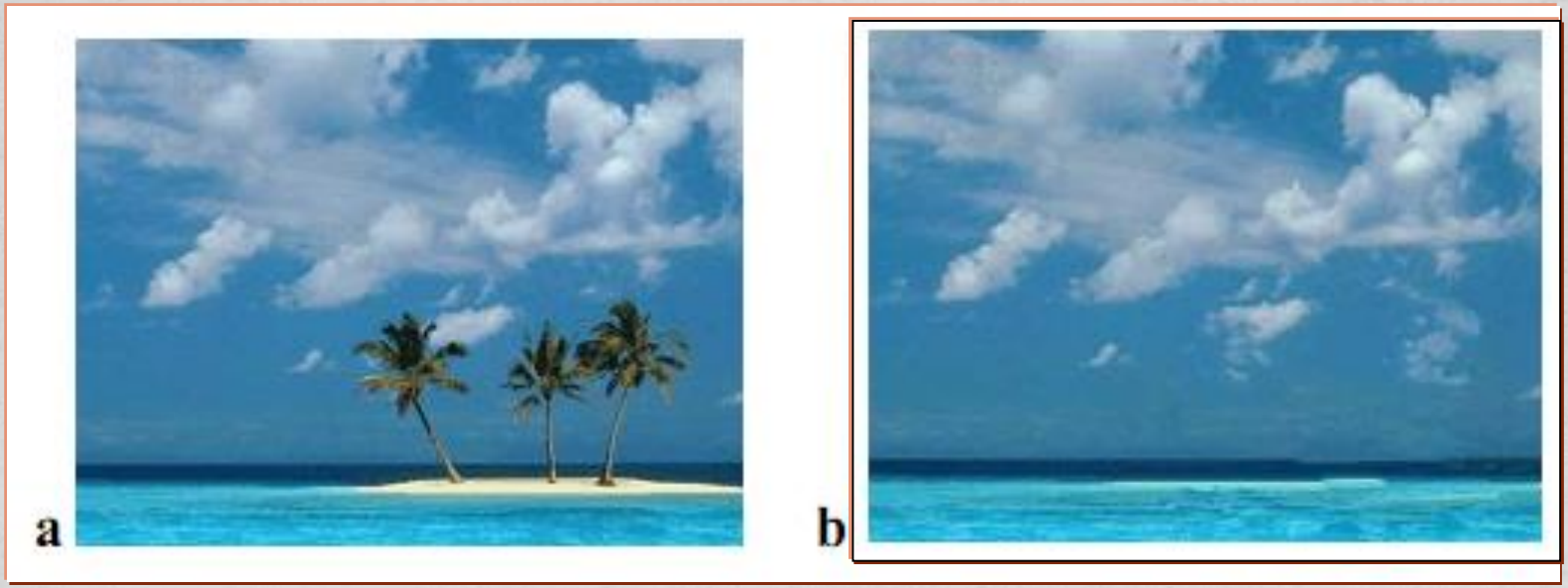
- 结果很好！
- 但是，为什么要画两个圈？

图像补全结果

- 自然场景

图像补全结果

- 自然场景



图像补全结果

- 复杂纹理背景

图像补全结果

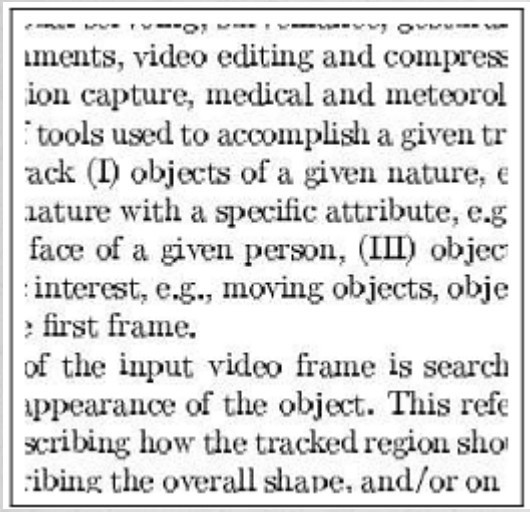
- 生活场景

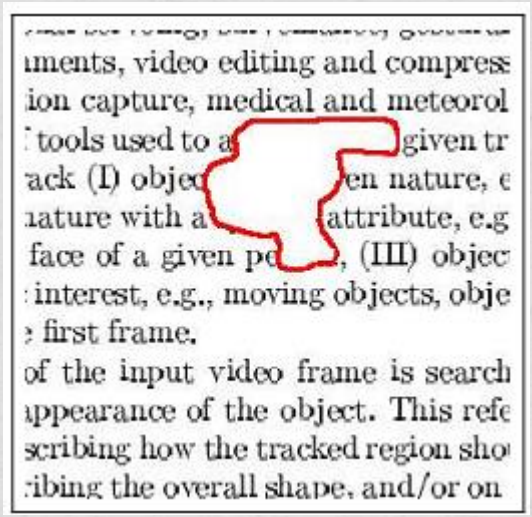
图像补全结果

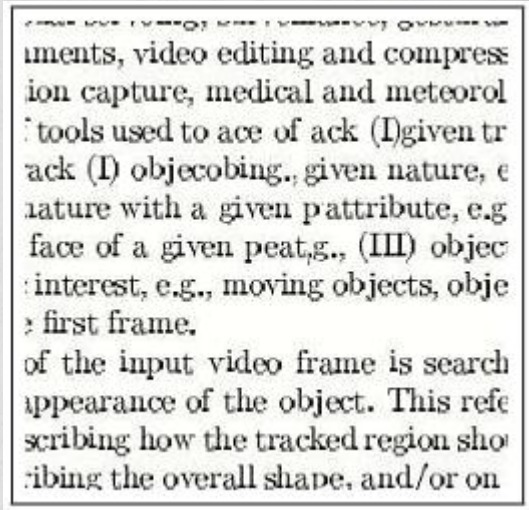
- 生活场景

图像补全结果

- 文字


uments, video editing and compress
ion capture, medical and meteorol
tools used to accomplish a given tr
ack (I) objects of a given nature, e
nature with a specific attribute, e.g
face of a given person, (III) objec
interest, e.g., moving objects, obje
first frame.
of the input video frame is search
appearance of the object. This refe
scribing how the tracked region sho
ribing the overall shape, and/or on


uments, video editing and compress
ion capture, medical and meteorol
tools used to a given tr
ack (I) objec en nature, e
nature with a attribute, e.g
face of a given pe , (III) objec
interest, e.g., moving objects, obje
first frame.
of the input video frame is search
appearance of the object. This refe
scribing how the tracked region sho
ribing the overall shape, and/or on


uments, video editing and compress
ion capture, medical and meteorol
tools used to ace of ack (I)given tr
ack (I) objecobing., given nature, e
nature with a given p attribute, e.g
face of a given peat,g., (III) objec
interest, e.g., moving objects, obje
first frame.
of the input video frame is search
appearance of the object. This refe
scribing how the tracked region sho
ribing the overall shape, and/or on

复杂度分析

- 共要补全M个像素
- 补全每一像素，要进行全图的邻域搜索
  - 邻域个数为全图像素个数N
- 每次计算能量函数时，要对邻域内所有像素求差
  - 邻域内像素个数为P

- 总共的计算复杂度为MNP，非常高！

时间消耗

3×3

423s

5×5

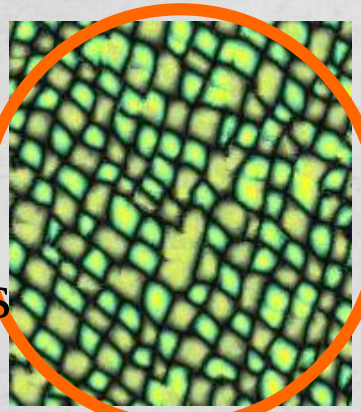
528s

7×7

739s

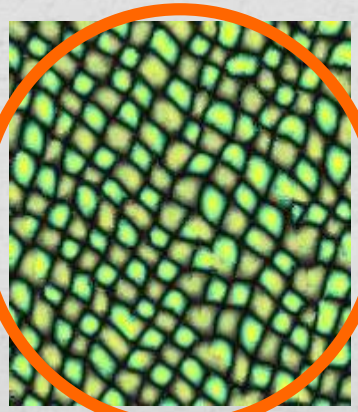
9×9

1020s



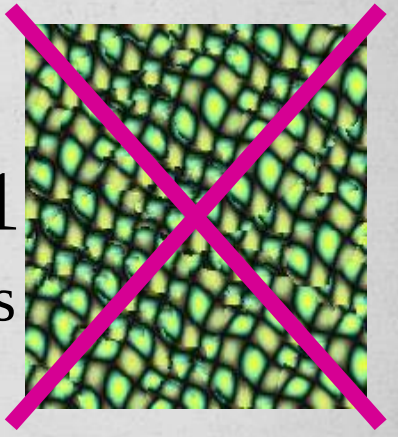
11×11

1445s



41×41

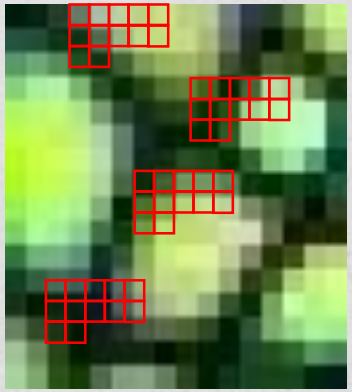
24350s



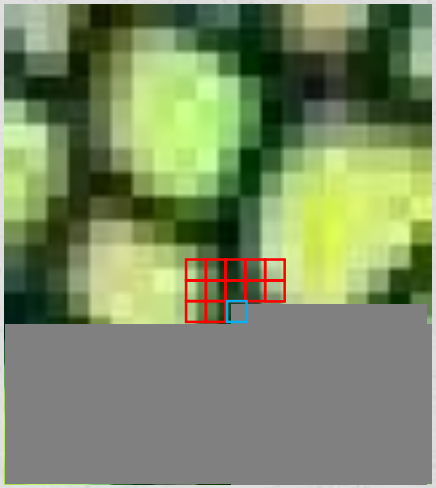
加速方法

- 计算瓶颈
  - 邻域搜索

源图像



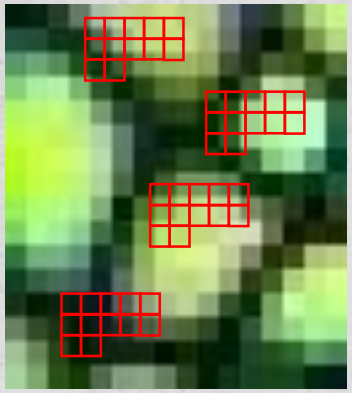
目标图像

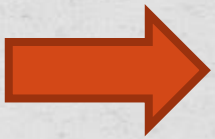


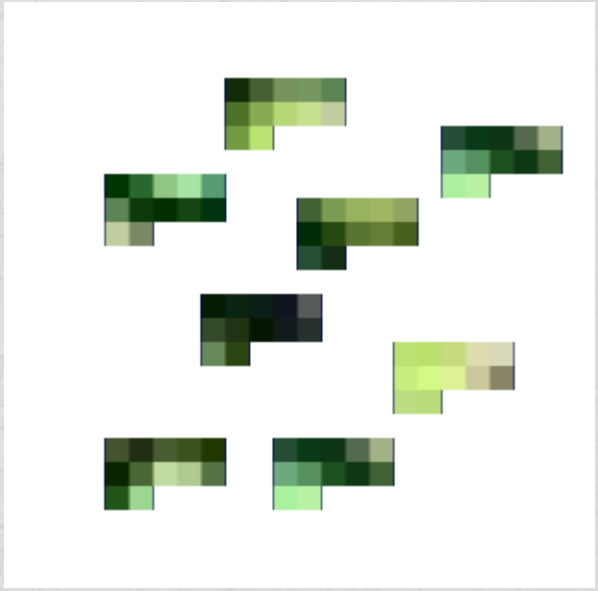
- 加速方法
  - 基于树结构的加速
  - 基于快速傅里叶变换的加速

加速方法1- 基于树结构的加速方法

源图像



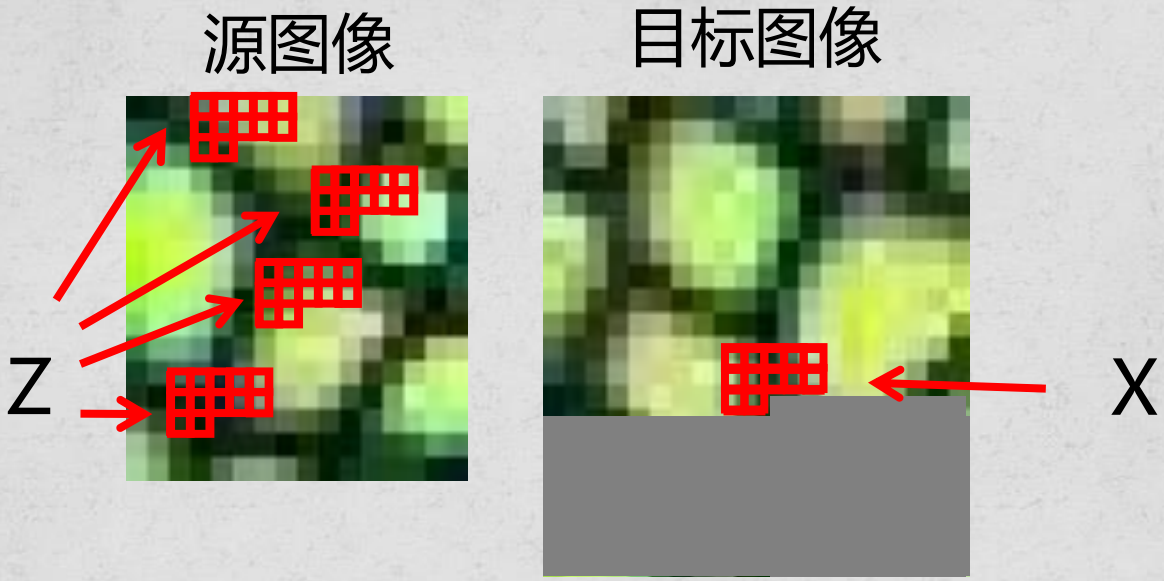




加速方法1- 基于树结构的加速方法

- 将邻域的像素值视为高维向量
- 采用传统的高维二叉树结构加速

加速方法2-基于快速傅里叶变换的加速



能量函数

$$E(Z)=\sum_p (X_p-Z_p)^2=\sum_p (X_p^2+Z_p^2-2X_pZ_p)$$

$$=\sum_p X_p^2+\sum_p Z_p^2-2\sum_p X_pZ_p$$

1. $\sum_p X_p^2$ 只用计算一次
2. $\sum_p Z_p^2$ 通过预积分表计算
3. $\sum_p X_pZ_p$是卷积形式，快速傅里叶变换

基于样本的图像补全

- 算法特点总结
  - 逐像素补全，搜索最匹配的邻域
  - 优先处在边界区域的像素
  - 优先邻域大的像素
  - 基于树结构的加速
  - 基于快速傅里叶变换的加速

基于块的补全方法

- Graphcut Textures: Image and Video Synthesis Using Graph Cuts. Vivek Kwatra et. al. SIGGRAPH 2003.

Sample Texture

Synthesized Texture

前一种方法的局限性

- 前一种方法需要逐像素求，速度很慢
- 每次实际上只使用了输入图像很少一部分的信息
- 为什么不能一次合成一整块的纹理呢？

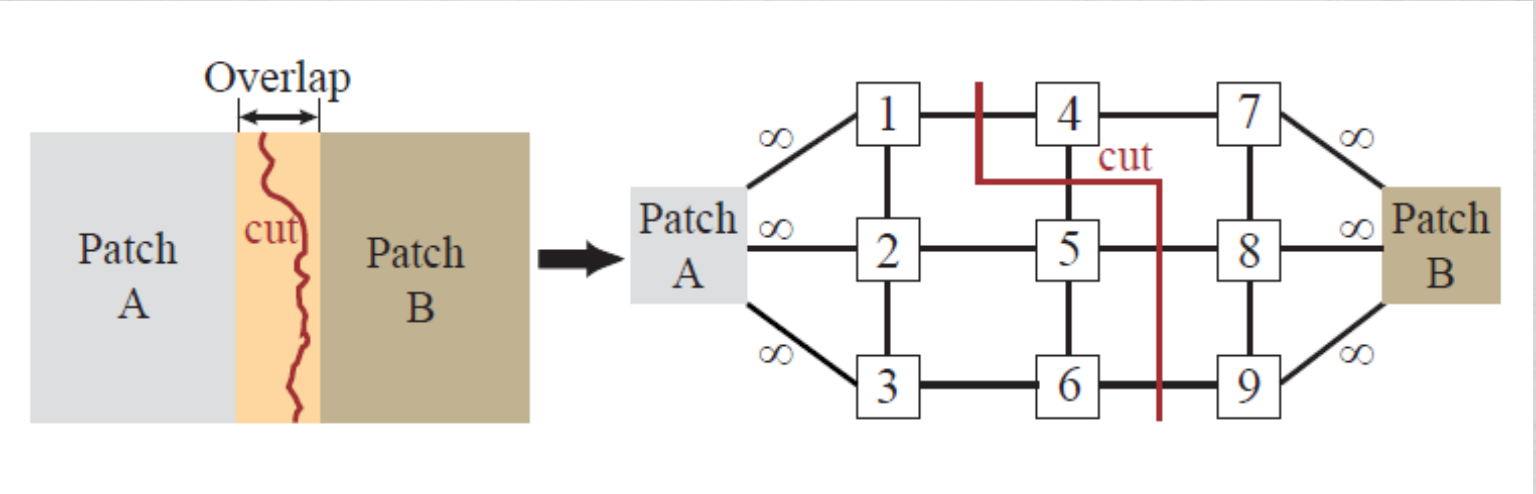
以块为单位进行纹理合成

以块为单位进行纹理合成

- 每次尝试将纹理的一个块（patch）拼接到现有的画面中
  - 这个块可能是整个输入纹理，或者是纹理中的一个块
- 拼接会产生重叠区域，需要决定拼接的边界
- 使用graph-cut算法计算出最佳的拼接边界
  - 一侧的像素采用现有画面的像素
  - 另一侧的像素采用新加入纹理的像素

Graph-cut的实现细节

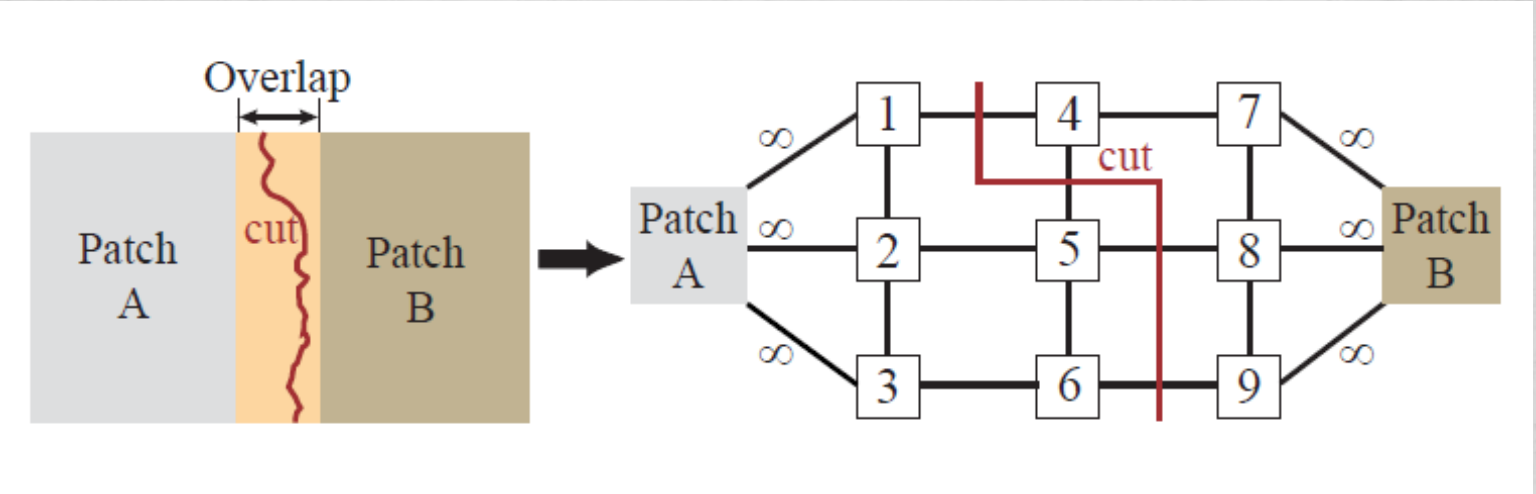
- 最简单的一种情况



- 采用图(Graph)表示
- 重叠部分每个像素点作为图中一个节点
- 相邻像素点之间连接一条边。

Graph-cut的实现细节

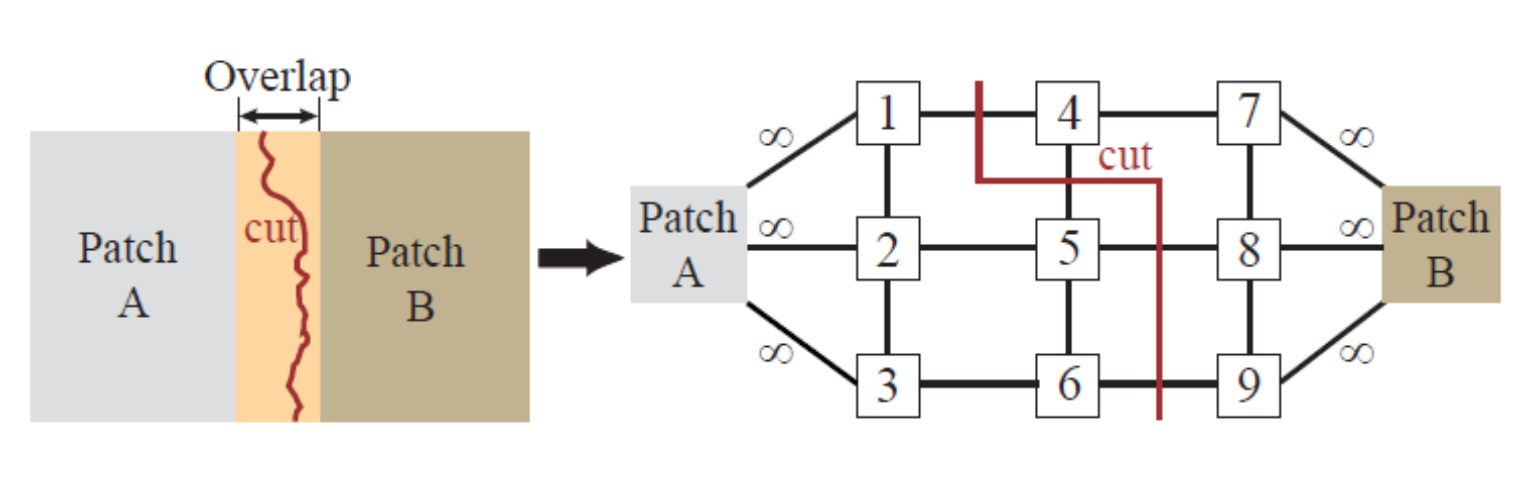
- 最简单的一种情况



- 假设只有两个patch，它们之间有部分区域重叠
- 另外增加起始和结束两个节点代表两个不同的patch
- 连接到起始和结束两个节点的边权值为正无穷

Graph-cut的实现细节

- 最简单的一种情况



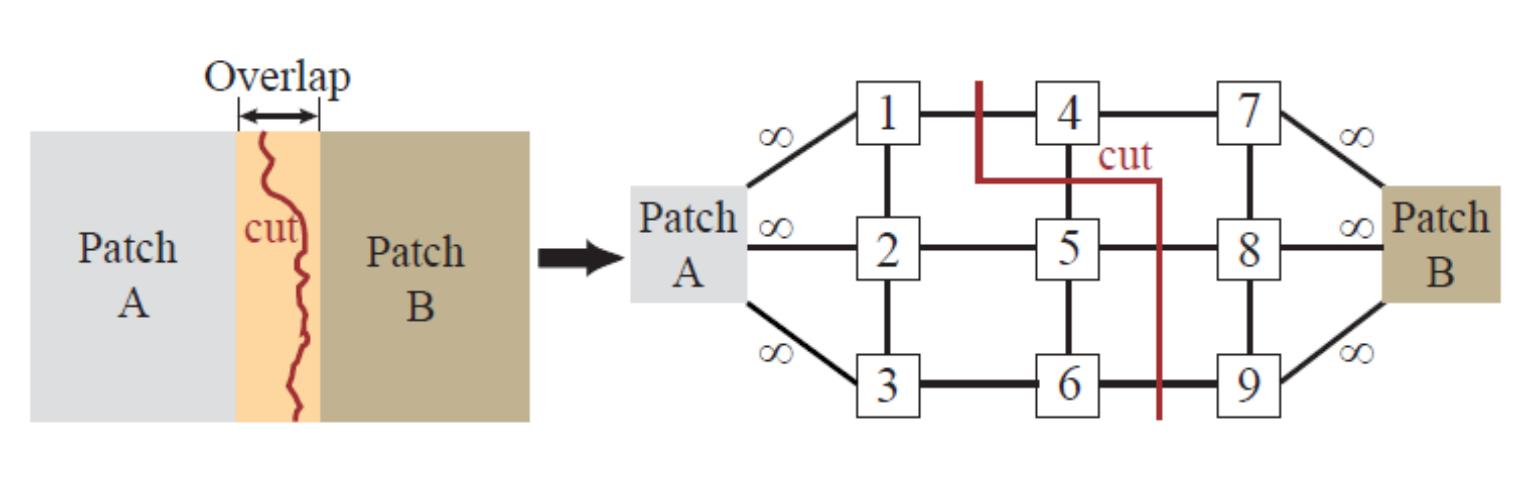
- 两个像素点之间的边权值通过下式计算

$$M(s,t,\mathbf{A},\mathbf{B}) = \|\mathbf{A}(s) - \mathbf{B}(s)\| + \|\mathbf{A}(t) - \mathbf{B}(t)\|$$

- 其中A和B分别表示左右两个patch，s和t表示相邻的两个像素；A(s)表示A在s像素处的颜色值

Graph-cut的实现细节

- 最简单的一种情况



- 计算图的最小割，cut左边的像素从patch A取，cut右边的像素从patch B取

Graph-cut的实现细节

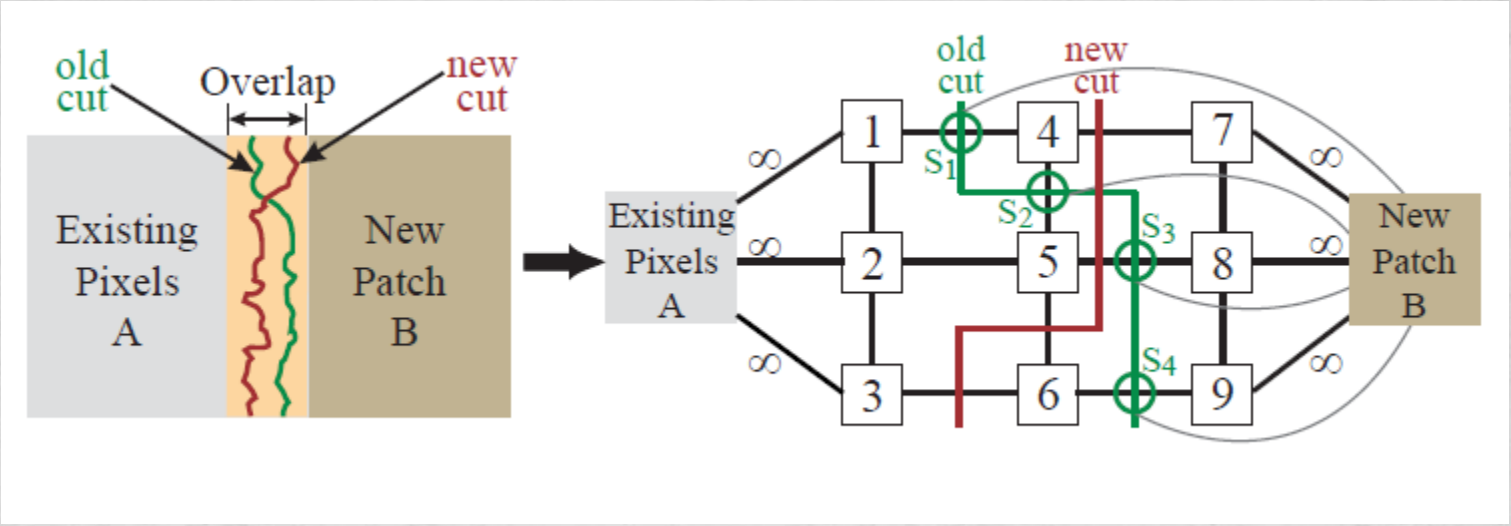
- 实际上有可能加入新patch的位置已经有原来计算的边界

Sample Texture

Synthesized Texture

Graph-cut的实现细节

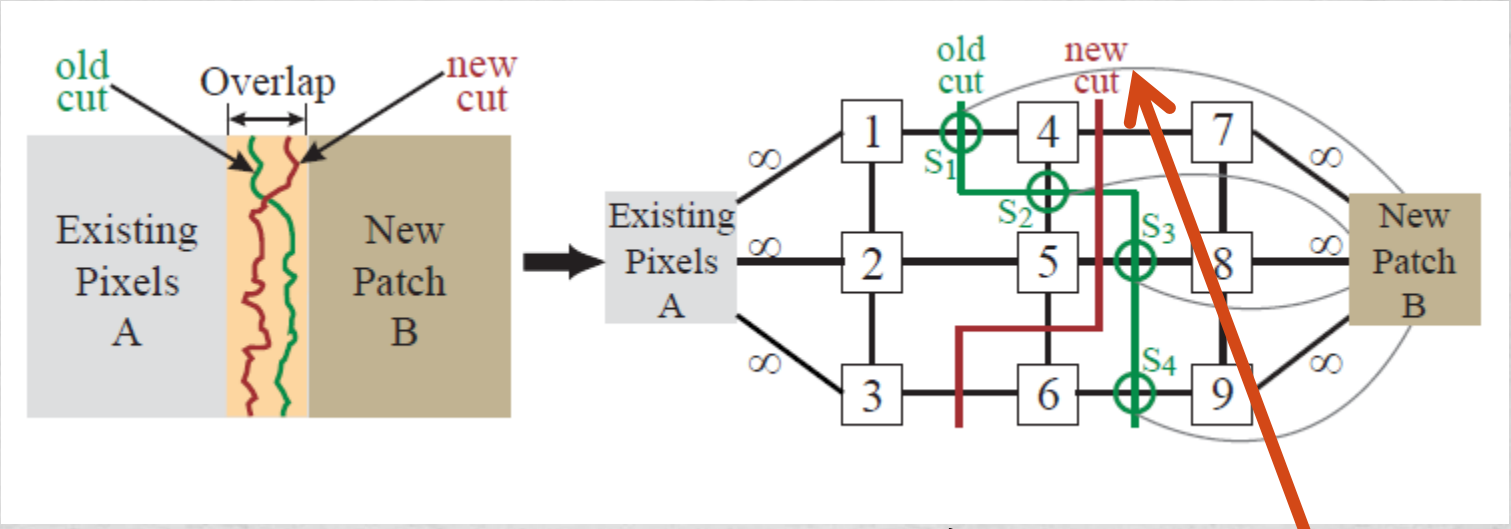
- 扩展原有的graph-cut算法



- 起始节点 Existing pixels A：现有画面已存在的像素
  - 有可能属于多个不同的patch
  - 以符号A_i表示像素i所属的patch
- 结束节点 New Patch B: 新加入的patch B

Graph-cut的实现细节

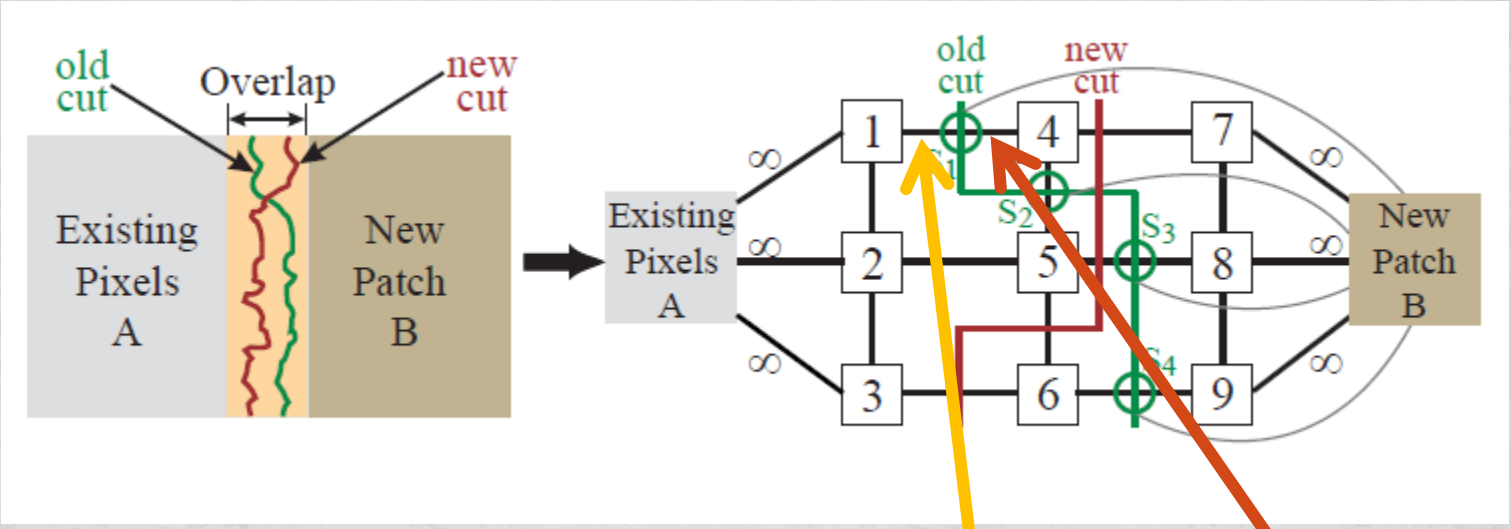
- 扩展原有的graph-cut算法



- old cut: 为现有画面上的割建立节点，如图中s_1~s_4，简称割节点。为每一个割节点引一条边至结束节点，新边的边权值为所割边原来的边权值。以割节点s_1为例，新加边的权值为$M(1,4,A_1,A_4)$。

Graph-cut的实现细节

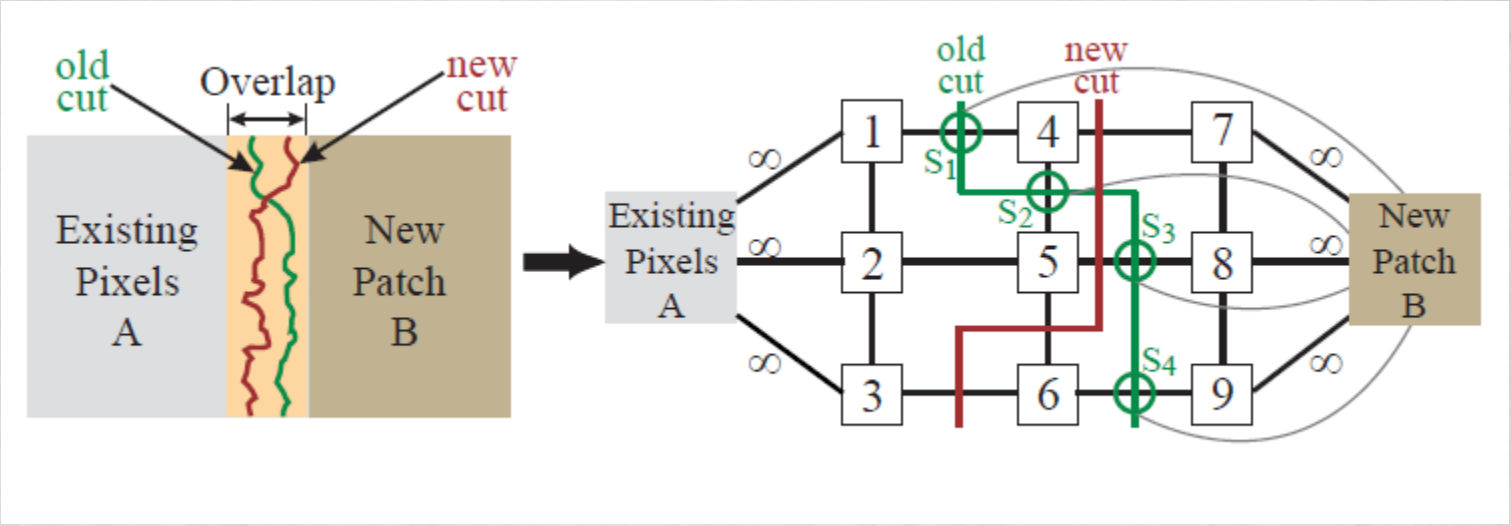
- 扩展原有的graph-cut算法



- old cut: 边分为两段：以割节点s_1为例，s_1将原来的边变为两段。这两段边的权值分别为$M(1,4,A_1,B)$和$M(1,4,B,A_4)$。

Graph-cut的实现细节

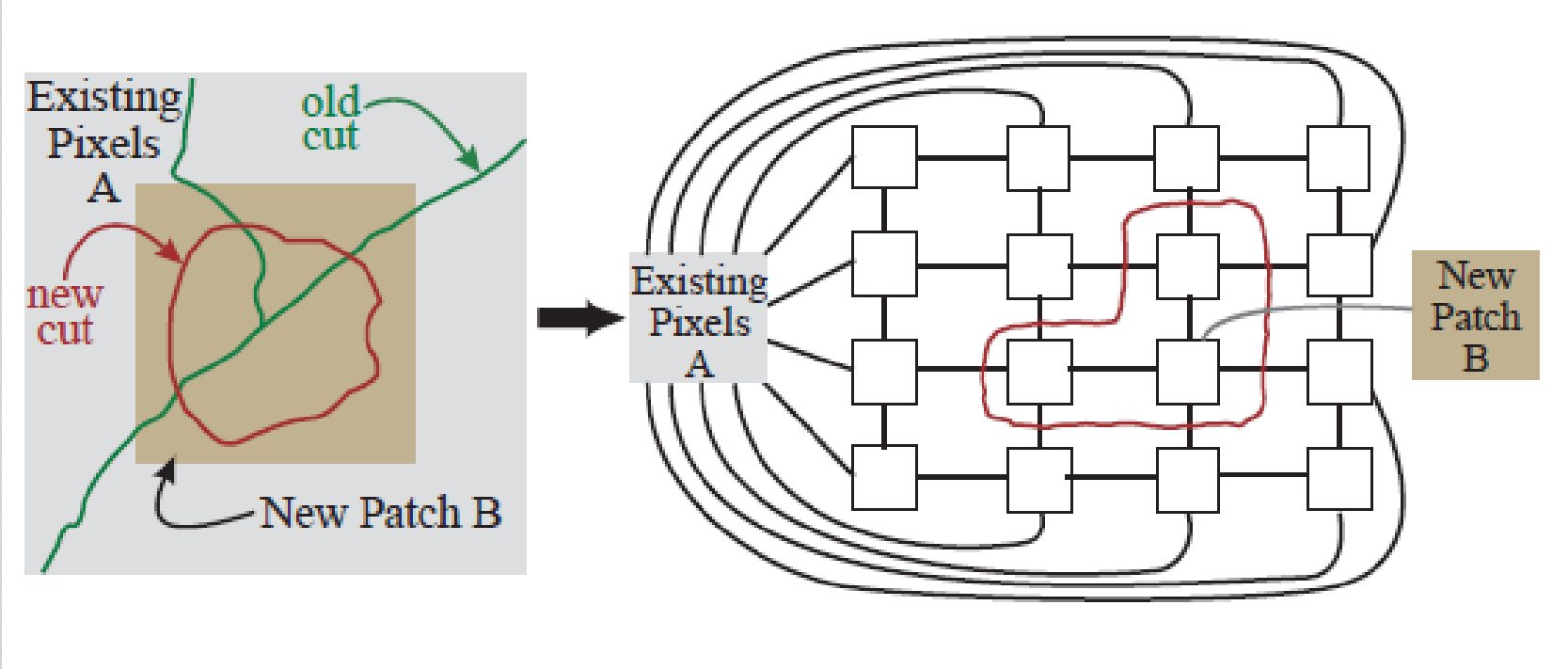
- 扩展原有的graph-cut算法



- 重新计算最小割，割的左边的像素点从属情况不变，割右边的像素点取自新的patch。
- 最小割的边权值之和：new cut + 余下的old cut

Graph-cut的实现细节

- 也可能出现这样的情况



- 实际计算与前面一样，不作详解

块偏移生成算法

- 每次尝试将纹理的一个块（patch）拼接到现有的画面中
  - 块从哪里选？这个块可能是整个输入纹理，或者是纹理中的一个块
  - 块放在画面的哪个位置？

块偏移生成算法

- 怎样决定新的块放在哪个位置？
  - 方法一：随机生成
    - 速度最快
    - 对于随机纹理而言有较好地效果

块偏移生成算法

- 怎样决定新的块放在哪个位置？
  - 方法二：匹配整个块的最佳位置
    - 某个位置的代价通过下式计算

$$C(t) = \frac{1}{|\mathbf{A}_t|} \sum_{p \in \mathbf{A}_t} |\mathbf{I}(p) - \mathbf{O}(p+t)|^2$$

其中t表示偏移，I表示输入纹理，O表示已有画面，A_t表示偏移为t时，新块与已有画面的重叠部分。

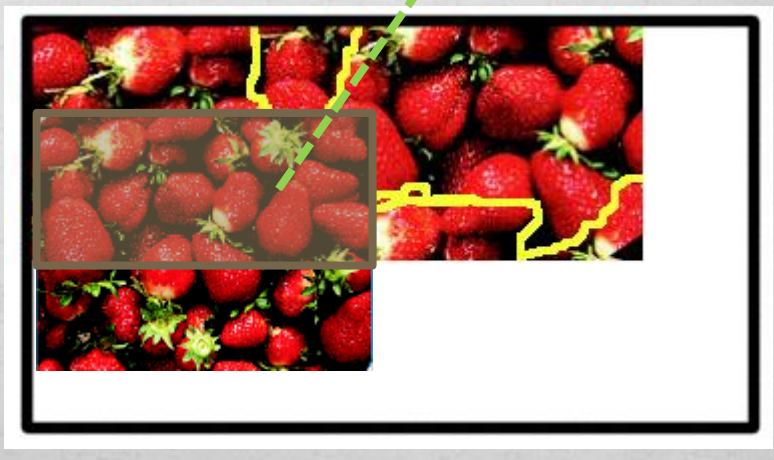
    - 每次选择位置时，选择偏移t的概率$P(t) \propto e^{-\frac{C(t)}{k\sigma^2}}$。

块偏移生成算法

- 怎样决定新的块放在哪个位置？
  - 方法二：匹配整个块的最佳位置
    - 某个位置的代价通过下式计算

$$C(t) = \frac{1}{|\mathbf{A}_t|} \sum_{p \in \mathbf{A}_t} |\mathbf{I}(p) - \mathbf{O}(p+t)|^2$$

$t = t_1$

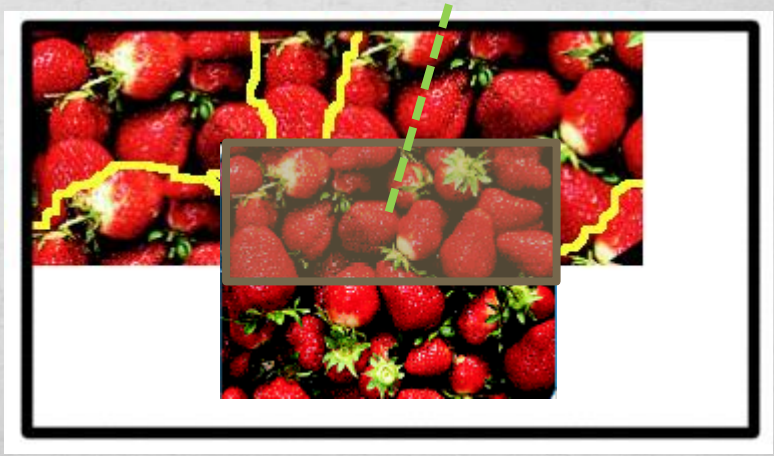


块偏移生成算法

- 怎样决定新的块放在哪个位置？
  - 方法二：匹配整个块的最佳位置
    - 某个位置的代价通过下式计算

$$C(t) = \frac{1}{|\mathbf{A}_t|} \sum_{p \in \mathbf{A}_t} |\mathbf{I}(p) - \mathbf{O}(p+t)|^2$$

$t = t_2$

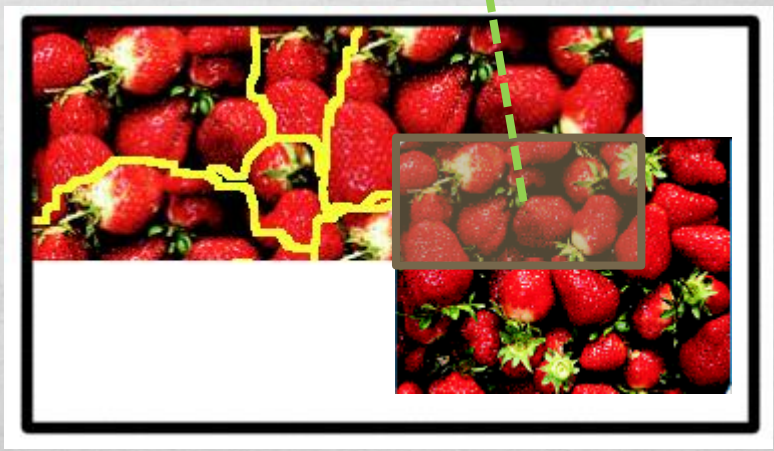


块偏移生成算法

- 怎样决定新的块在那个位置？
  - 方法二：匹配整个块的最佳位置
    - 某个位置的代价通过下式计算

$$C(t) = \frac{1}{|\mathbf{A}_t|} \sum_{p \in \mathbf{A}_t} |\mathbf{I}(p) - \mathbf{O}(p+t)|^2$$

$t = t_3$



块偏移生成算法

- 怎样决定新的块放在哪个位置？
  - 方法三：匹配子块的最佳位置
    - 每次在已有画面中选择一个小于输入纹理大小的子块
    - 然后在输入纹理中找出最佳匹配位置t
    - 通过下式计算代价

$$C(t) = \sum_{p \in \mathbf{S_O}} |\mathbf{I}(p-t) - \mathbf{O}(p)|^2$$

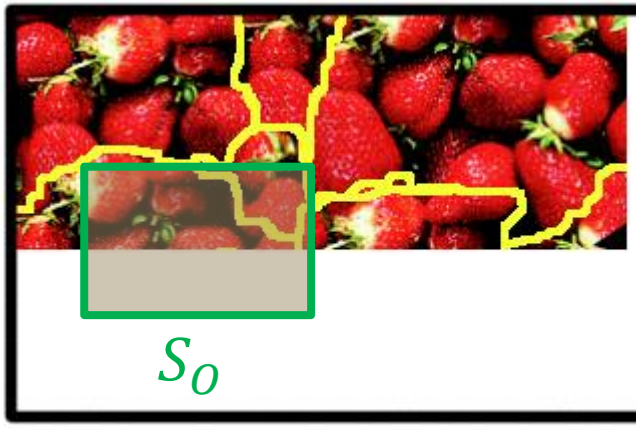
      - 其中S_o代表的是子块，t代表的是偏移量，在实际计算时只计算重叠部分
    - 通过同样的方法以概率$P(t) \propto e^{-\frac{C(t)}{k\sigma^2}}$选择偏移

块偏移生成算法

- 怎样决定新的块放在哪个位置？
  - 方法三：匹配子块的最佳位置
    - 每次在已有画面中选择一个小于输入纹理大小的子块
    - 然后在输入纹理中找出最佳匹配位置t

$$C(t) = \sum_{p \in \mathbf{S_O}} |\mathbf{I}(p-t) - \mathbf{O}(p)|^2$$

$t = t_1$

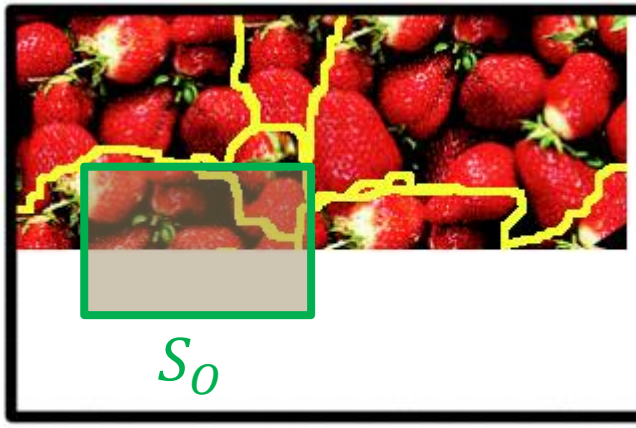


块偏移生成算法

- 怎样决定新的块放在哪个位置？
  - 方法三：匹配子块的最佳位置
    - 每次在已有画面中选择一个小于输入纹理大小的子块
    - 然后在输入纹理中找出最佳匹配位置t

$$C(t) = \sum_{p \in \mathbf{S_O}} |\mathbf{I}(p-t) - \mathbf{O}(p)|^2$$

$t = t_2$

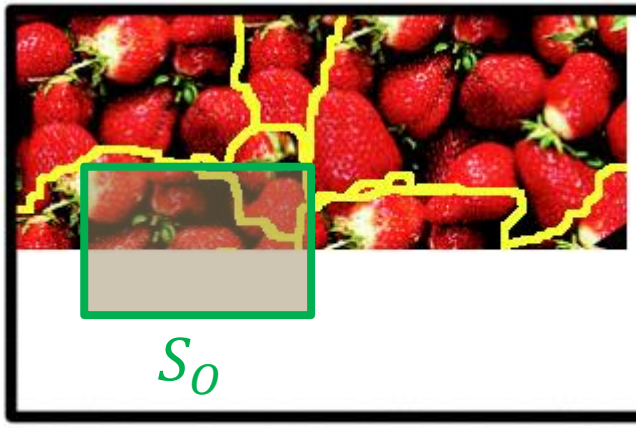


块偏移生成算法

- 怎样决定新的块放在哪个位置？
  - 方法三：匹配子块的最佳位置
    - 每次在已有画面中选择一个小于输入纹理大小的子块
    - 然后在输入纹理中找出最佳匹配位置t

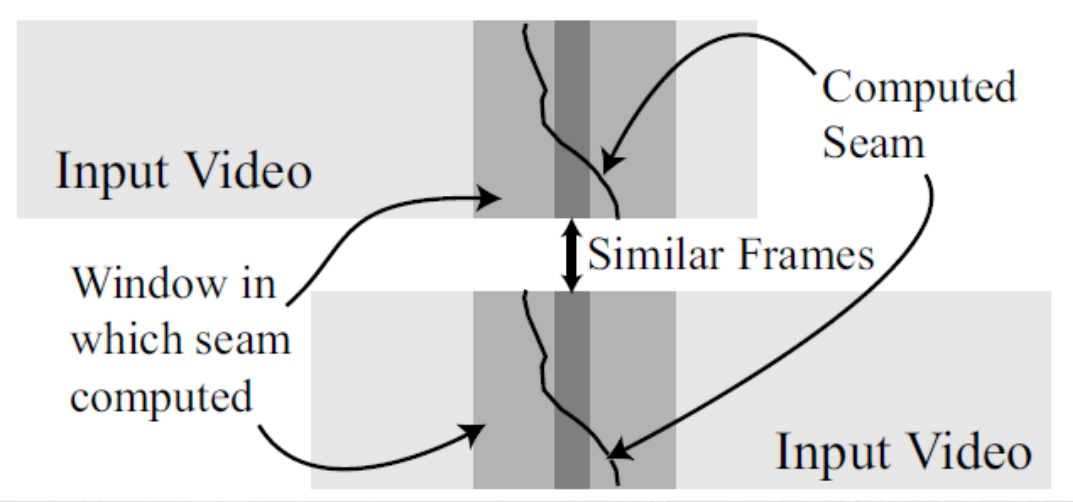
$$C(t) = \sum_{p \in \mathbf{S_O}} |\mathbf{I}(p-t) - \mathbf{O}(p)|^2$$

$t = t_3$



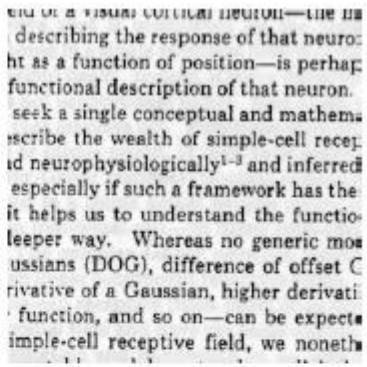
其他实现细节

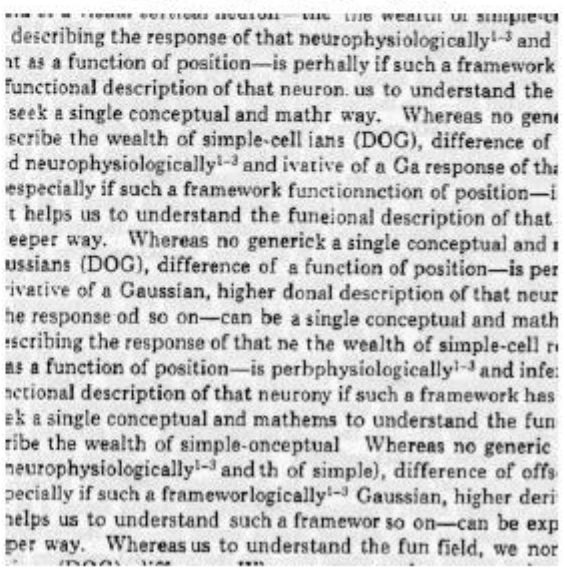
- 直接用graph-cut计算出来的图像融合边界可能还是存在毛疵，可以利用边缘羽化等方法优化最终结果。
- 加上时间维度的考虑可以很容易地将算法扩展到视频合成上。



- 为了保证在有限时间内完成纹理合成，需要保证每次新块都会覆盖掉一部分未合成区域。

实验结果





实验结果

实验结果

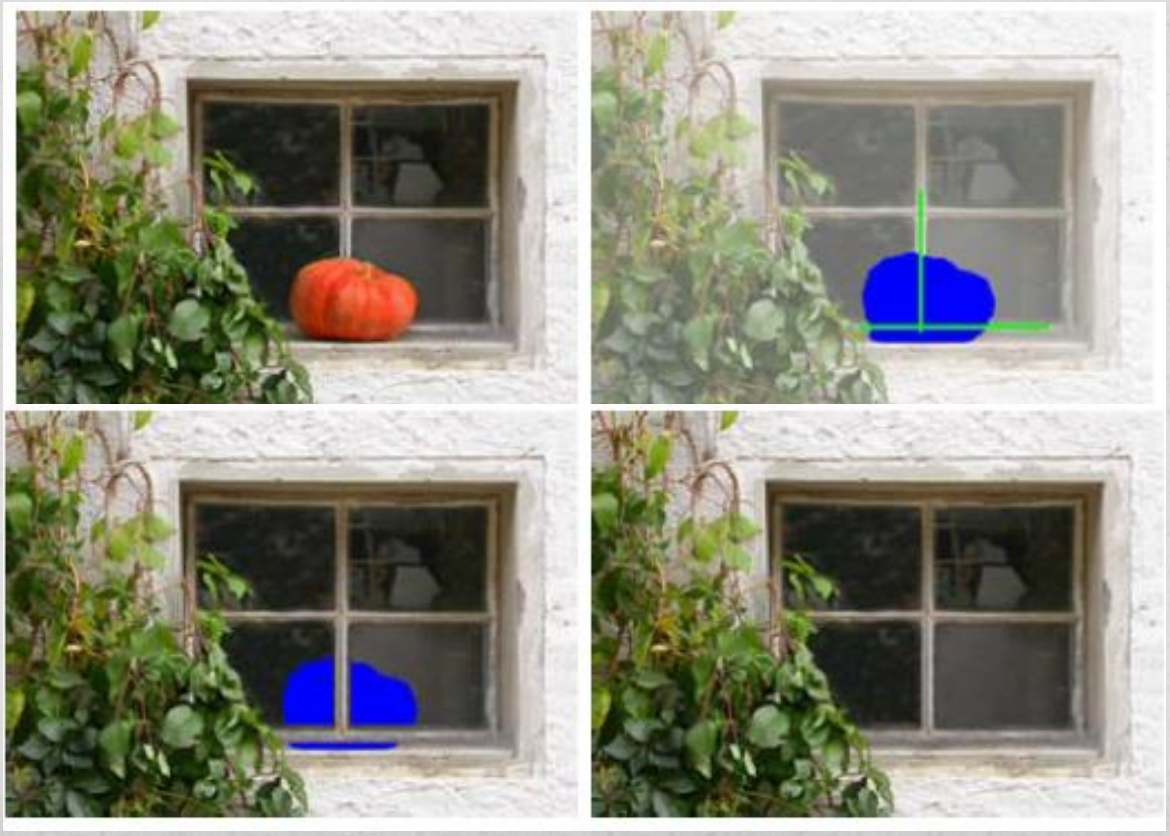
实验结果

算法特点总结

- 以块为单位进行补全
- 对于非随机性纹理也能处理
- 除了图像还能够应用在视频上

带交互的结构补全方法

- Image Completion with Structure Propagation.
  Jian Sun et. al. SIGGRAPH 2005.



前面两种方法的局限性

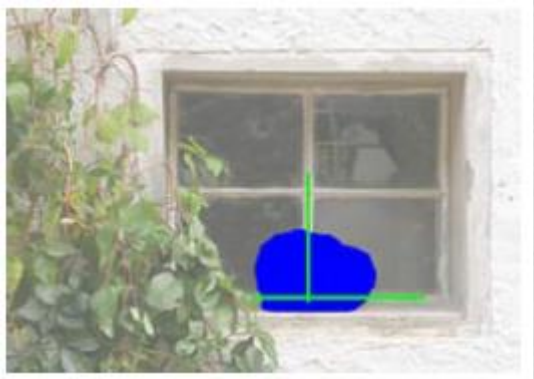
- 第一种方法解决了简单纹理的补全问题，第二种方法以块为单位进一步加速并且优化了补全结果
- 但假如待补全区域中存在显著结构，用前两种方法补全的效果会很差。如下图在抠出南瓜后，用前两种补全方法无法恢复窗框的结构。

本方法的三个步骤

- Step 1：通过用户交互输入粗略的结构信息。
- Step 2：补全未知区域中的结构部分。
- Step 3：补全未知区域中剩下的纹理部分。

Step 1：通过交互输入结构信息

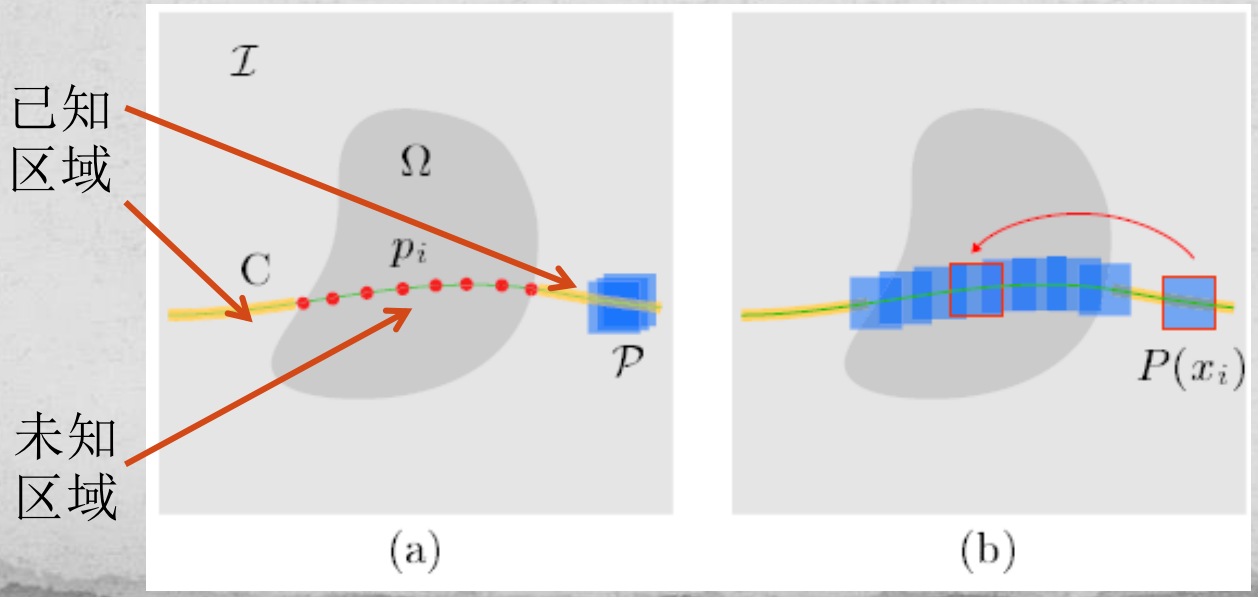
- 为了解决这一问题，本方法加入了用户对结构信息的输入。如下图所示，结构线（绿色）标出了窗框的大致结构。



- 然后，在这些结构线上离散采样，得到一系列的采样点。其中绿线的交点必须作为采样点。

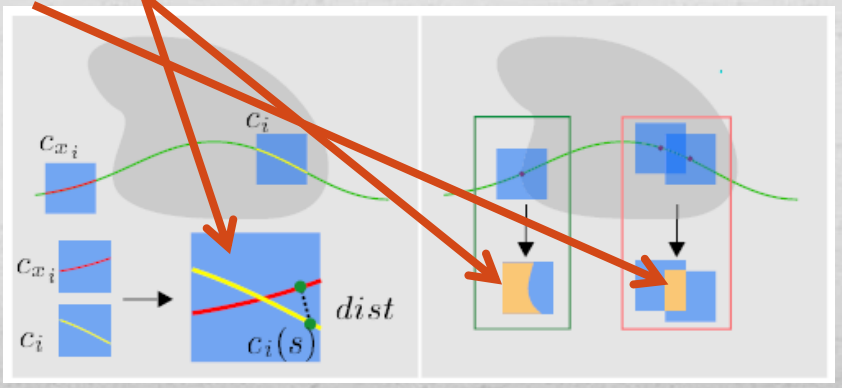
Step 2：结构补全

- 用p_i（图中红点）表示未知区域中的第i采样点(共L个)
- 已知区域：编号为1,2,...,N，共N个采样点(patch)
- 结构补全：针对每个未知区域的采样点，从已知区域结构线上选取一个采样点(patch)复制过去
- $P(x_i)$表示未知区域第i个采样点所取patch的编号为x_i



Step 2：结构补全

- 如何选择最匹配的采样点(patch)？
  - 优化目标：$E(X) = \sum_i E_1(x_i) + \sum_{i,j} E_2(x_i, x_j)$
    - $E_1(x_i) = k_s E_s(x_i) + k_i E_I(x_i)$
    - $E_s(x_i) = d(c_i, c_{x_i}) + d(c_{x_i}, c_i)$，$d$的定义为：一条线段上的所有点到另一条线段的最短距离之和。
    - E_I是某个patch超出未知区域部分与已知区域之间的平方差之和
    - E_2是相邻的两个patch重叠部分的平方差之和



Step 2：结构补全

- 如何选择最匹配的采样点(patch)？
  - 有了能量方程之后，可以使用动态规划计算出每个采样点应该选取的patch编号。
  - 以$M_i(x_i)$表示当第i个采样点选取patch编号为x_i时，从第1个采样点到第i个采样点计算的累计最小值

$$M_i(x_i) = E_1(x_i) + \min_{x_{i-1}}\{E_2(x_{i-1}, x_i) + M_{i-1}(x_{i-1})\},$$

最后一个采样点的patch编号取值为

$$x_L^* = \arg\min_{x_L} M_L(x_L)$$

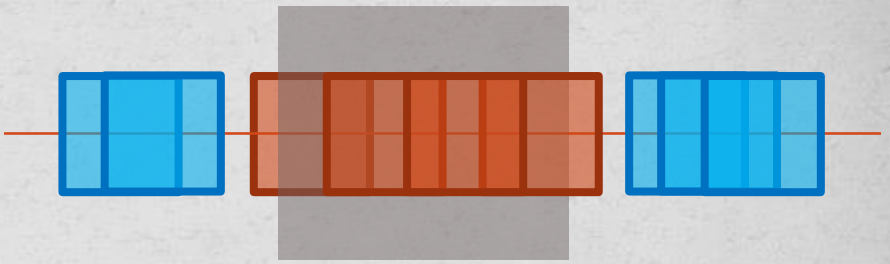
然后，再逆向计算出各个采样点的patch编号取值。

Step 2：结构补全

- 如何选择最匹配的采样点(patch)？

$$M_i(x_i) = E_1(x_i) + \min_{x_{i-1}}\{E_2(x_{i-1}, x_i) + M_{i-1}(x_{i-1})\},$$

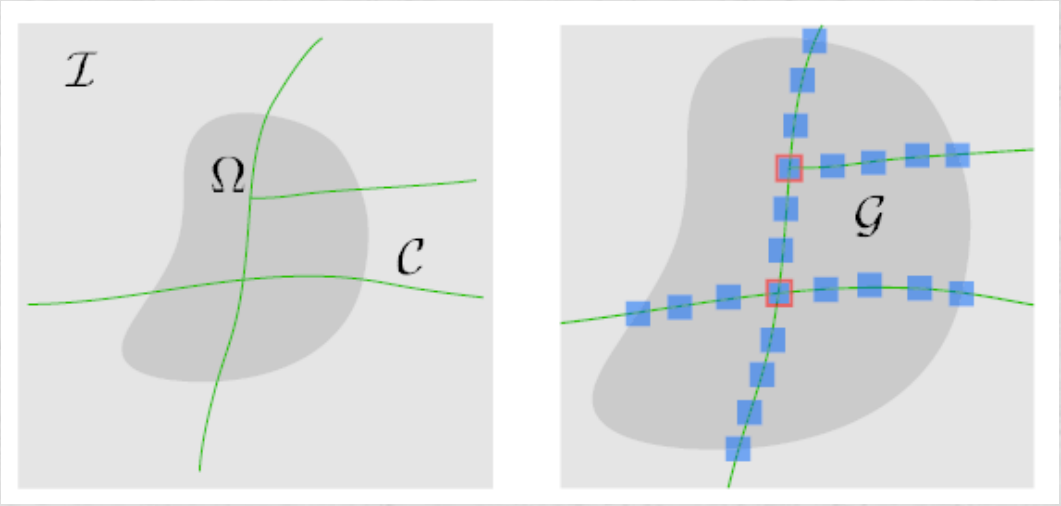
- 动态规划举例：
  - $L = 4$，即有4个待补全的patch
  - x_i的取值情况有5种，即5个样本patch
  - 初始化$M_1(1) = E_1(1)$，$M_1(2) = E_1(2)$，……，$M_1(5) = E_1(5)$
  - 然后依次计算$M_2(1)$~$M_2(5)$，$M_3(1)$~$M_3(5)$，$M_4(1)$~$M_4(5)$
  - 最后令x_4的取值使M_4最小，确定x_4后，再依次确定x_3, x_2, x_1



未知区域

Step 2：结构补全

- 如果多条结构线相交怎么办？
  - 对于多条相交结构线的情况，动态规划的方法复杂度为$O(LN^{2+K})$，K为结构线交点个数，L为未知采样点的个数，N为已知采样点（patch）的个数。

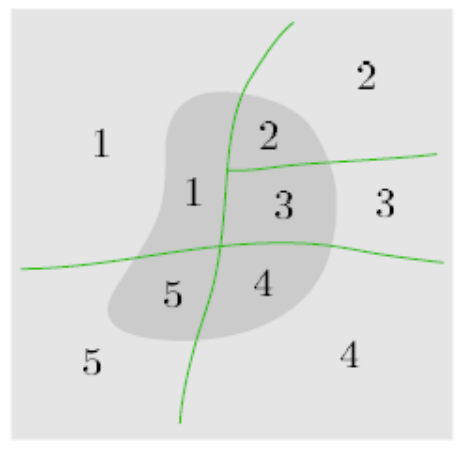


  - 这时，需要改用置信传播(belief propagation)的方法来计算最佳patch的位置，复杂度可以降至$O(LN^2)$。

Step 3：纹理补全

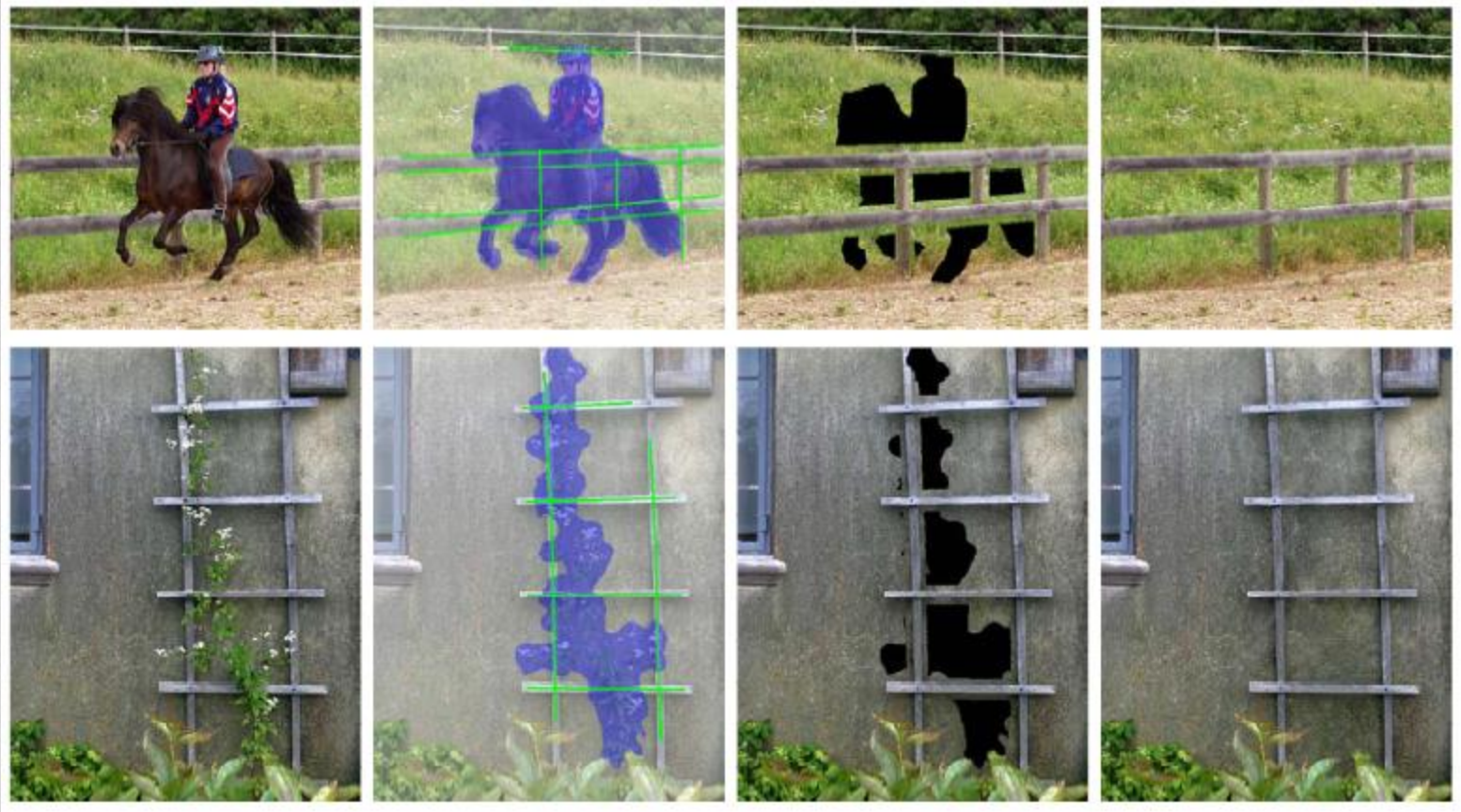
- 在补全了结构信息之后，可以得到下图

- 对于剩下的未知区域，使用类似前一种纹理补全算法的方法，补全纹理。
- 需要注意的是，不同未知区域选取patch的范围不同，如右图。

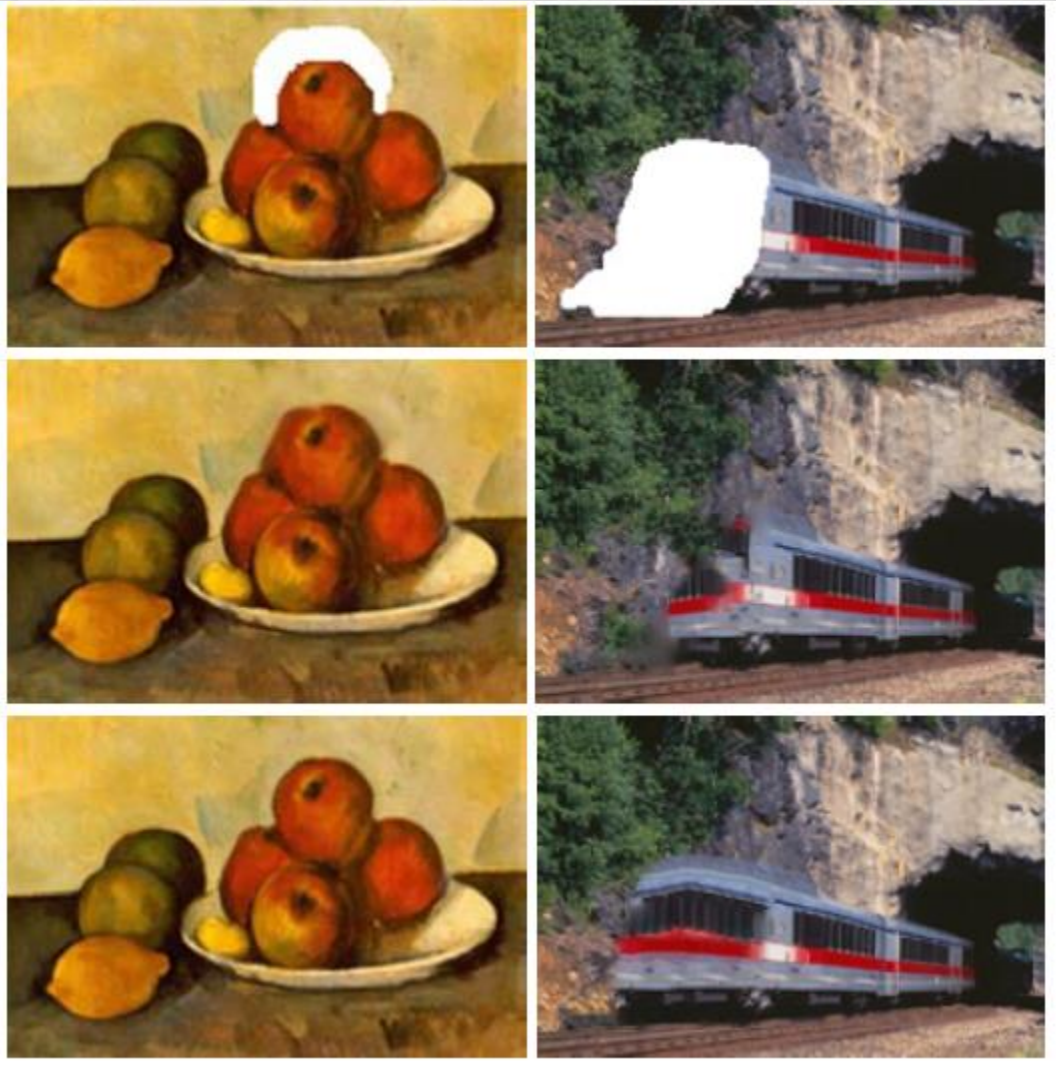


补全效果

补全效果



对比有无结构信息的补全效果



算法特点总结

- 本算法引入了用户交互，能够处理未知区域存在显著结构的情况。
- 先补全结构部分，再补全纹理部分。