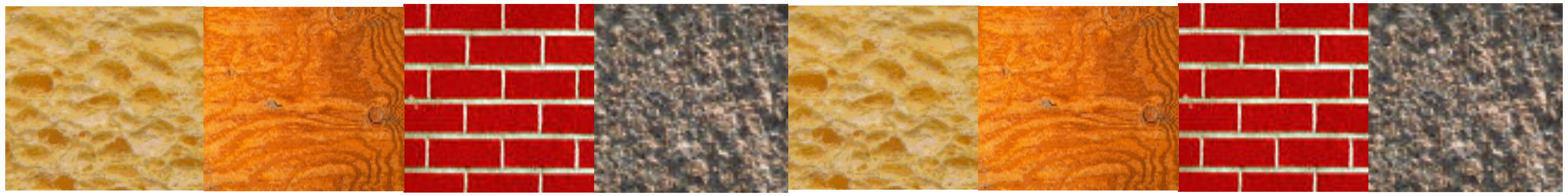
THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 11: Texture

(unless otherwise stated slides are taken or adopted from **Bob Woodham, Jim Little** and **Fred Tung**)

Texture

What is **texture**?

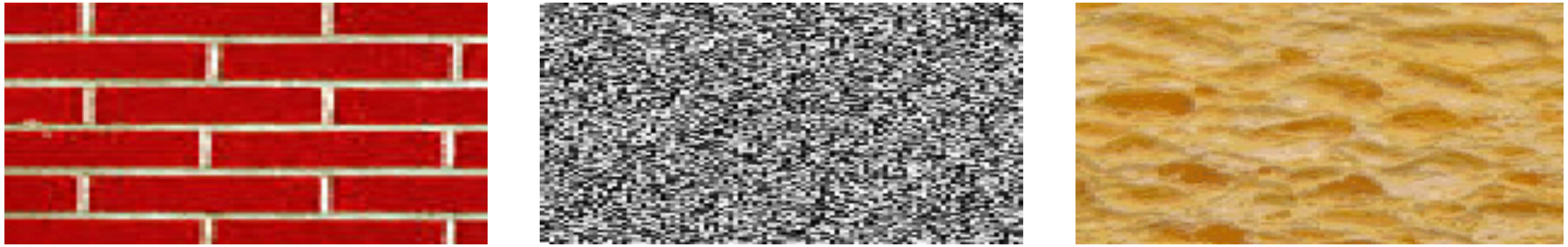


Figure Credit: Alexei Efros and Thomas Leung

Texture is widespread, easy to recognize, but hard to define

Views of large numbers of small objects are often considered textures
— e.g. grass, foliage, pebbles, hair

Patterned surface markings are considered textures
— e.g. patterns on wood

Definition of **Texture**

(Functional) **Definition**:

Texture is detail in an image that is at a scale too small to be resolved into its constituent elements and at a scale large enough to be apparent in the spatial distribution of image measurements

Definition of **Texture**

(Functional) **Definition**:

Texture is detail in an image that is at a scale too small to be resolved into its constituent elements and at a scale large enough to be apparent in the spatial distribution of image measurements

Sometimes, textures are thought of as patterns composed of repeated instances of one (or more) identifiable elements, called **textons**.

— e.g. bricks in a wall, spots on a cheetah

Uses of **Texture**

Texture can be a strong cue to **object identity** if the object has distinctive material properties

Texture can be a strong cue to an **object's shape** based on the deformation of the texture from point to point.

— Estimating surface orientation or shape from texture is known as "**shape from texture**"

Texture

We will look at two main questions:

1. How do we represent texture?
   → Texture **analysis**

2. How do we generate new examples of a texture?
   → Texture **synthesis**

We begin with texture synthesis to set up **Assignment 3**

Texture **Synthesis**

Texture **Synthesis**

Why might we want to synthesize texture?

Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

— Art directors might want to remove telephone wires. Restorers might want to remove scratches or marks.

Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

— Art directors might want to remove telephone wires. Restorers might want to remove scratches or marks.

— We need to find something to put in place of the pixels that were removed

Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

— Art directors might want to remove telephone wires. Restorers might want to remove scratches or marks.

— We need to find something to put in place of the pixels that were removed

— We synthesize regions of texture that fit in and look convincing

Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

— Art directors might want to remove telephone wires. Restorers might want to remove scratches or marks.

— We need to find something to put in place of the pixels that were removed

— We synthesize regions of texture that fit in and look convincing

2. To produce large quantities of texture for computer graphics

Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

— Art directors might want to remove telephone wires. Restorers might want to remove scratches or marks.

— We need to find something to put in place of the pixels that were removed

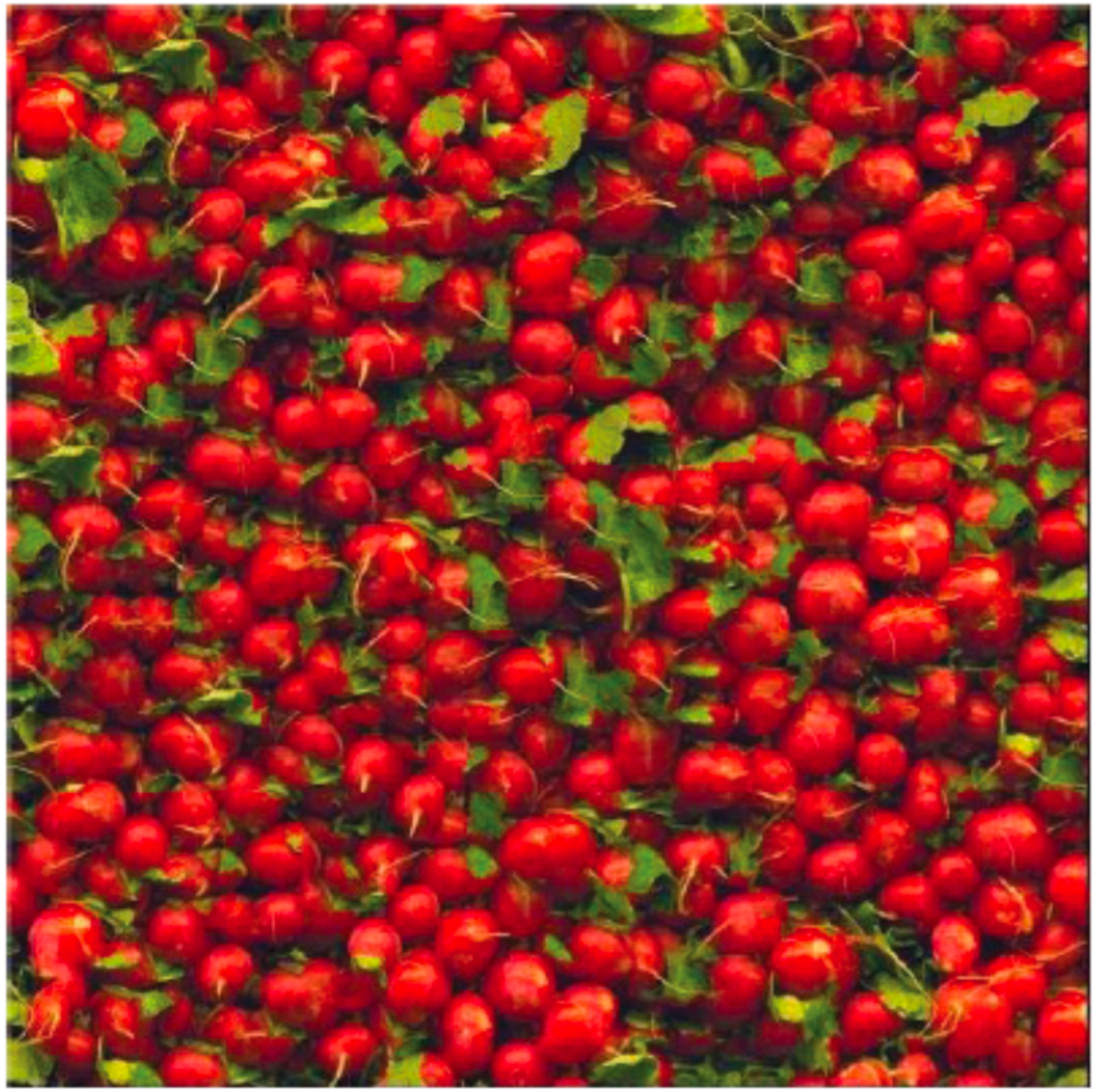
— We synthesize regions of texture that fit in and look convincing

2. To produce large quantities of texture for computer graphics

— Good textures make object models look more realistic

Texture **Synthesis**

radishes



lots more radishes

Szeliski, Fig. 10.49

Texture **Synthesis**

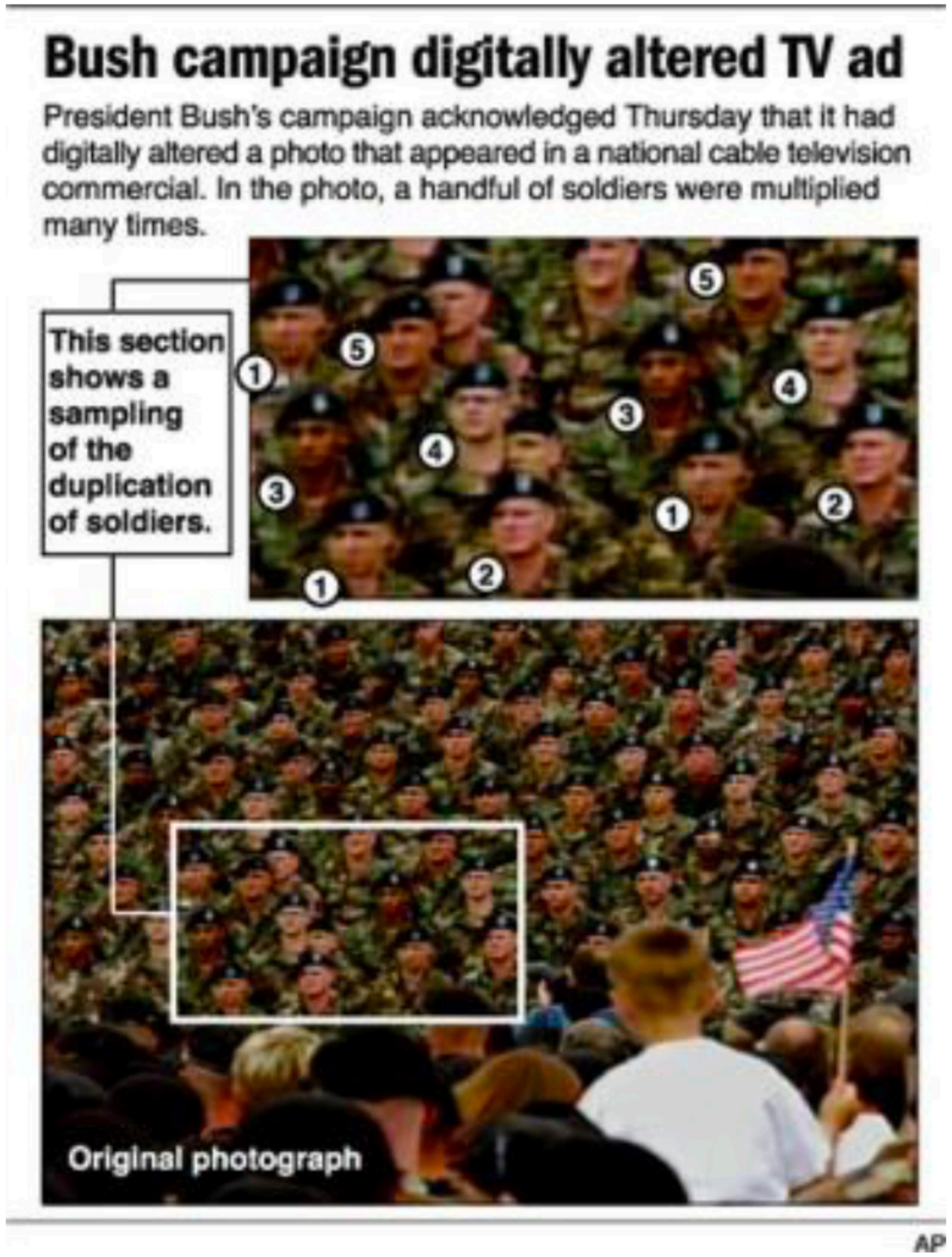


Photo Credit: Associated Pres

Texture **Synthesis**

Cover of "The Economist," June 19, 2010

Photo Credit (right): Reuters/Larry Downing

Assignment 3 Preview: Texture Synthesis

Task: Make donkey vanish

Assignment 3 Preview: Texture Synthesis

Task: Make donkey vanish

Method: Fill-in regions using texture from the white box

Assignment 3 Preview: Texture Synthesis

Task: Make donkey vanish

Method: Fill-in regions using texture from the white box

Texture Synthesis

Objective: Generate new examples of a texture. We take a “data-driven" approach

Idea: Use an image of the texture as the source of a probability model

— Draw samples directly from the actual texture

— Can account for more types of structure

— Very simple to implement

— Success depends on choosing a correct “distance”

Texture Synthesis by Non-parametric Sampling

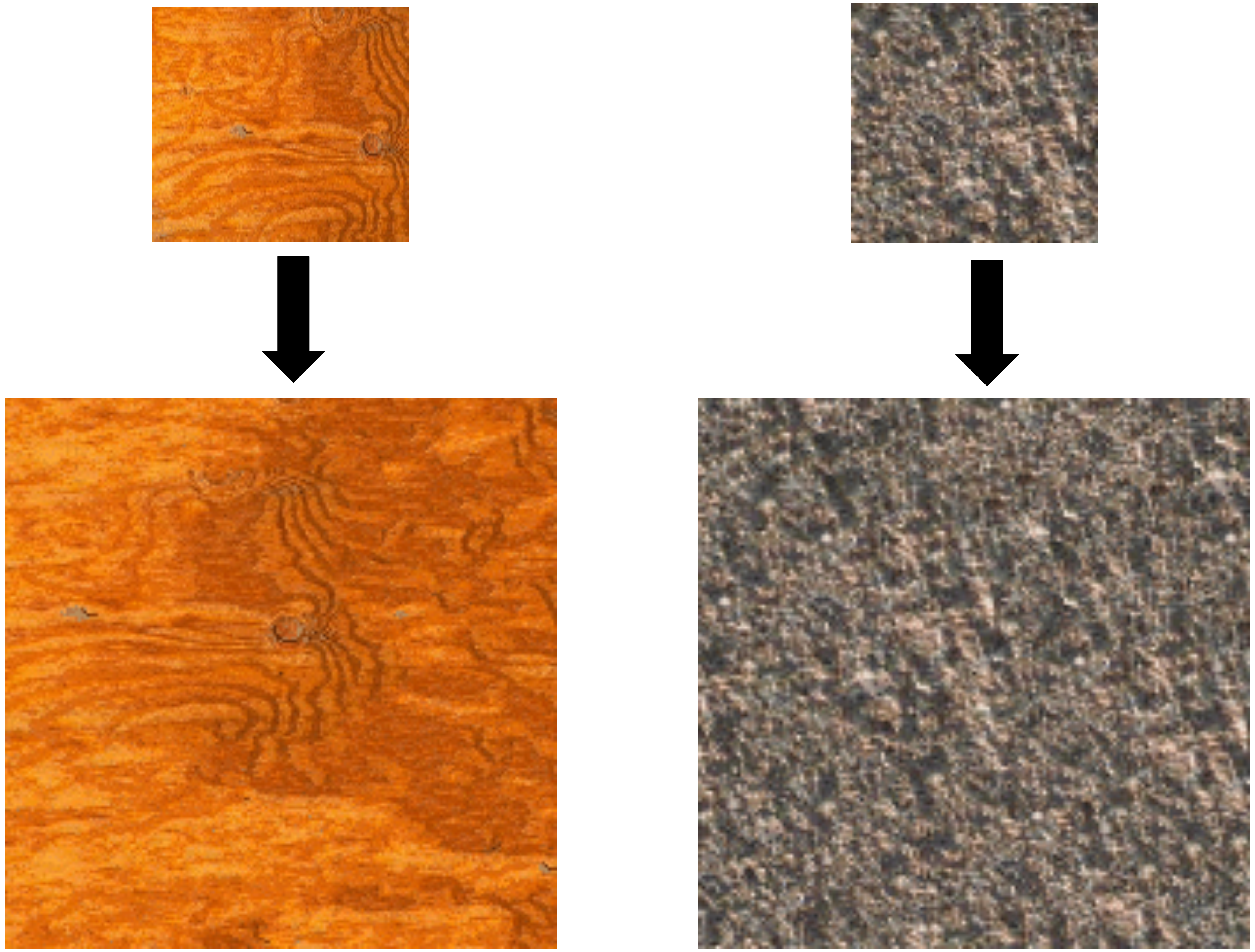
→

Alexei Efros and Thomas Leung

UC Berkeley

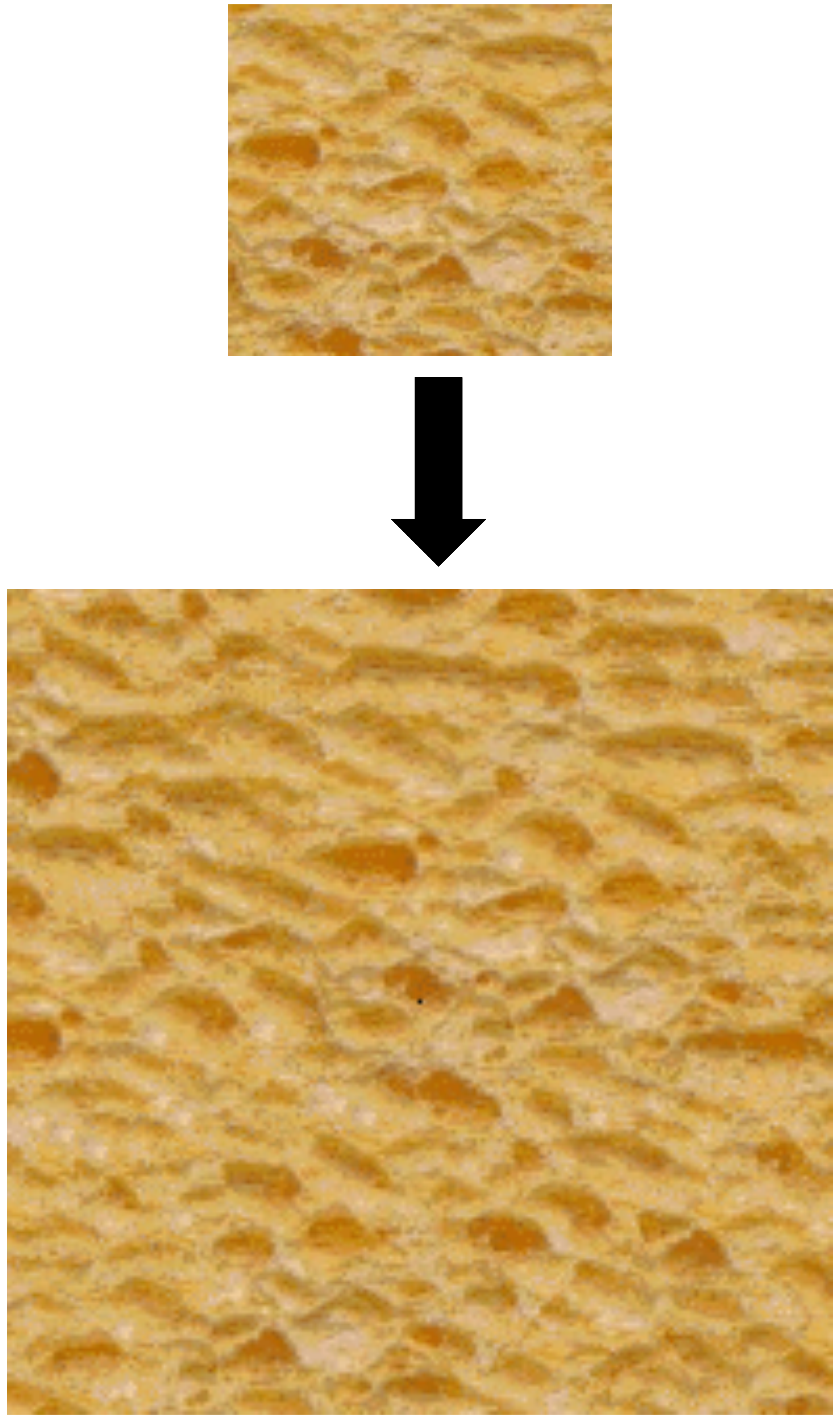
Slide Credit: http://graphics.cs.cmu.edu/people/efros/research/NPS/efros-iccv99.ppt

Efros and Leung

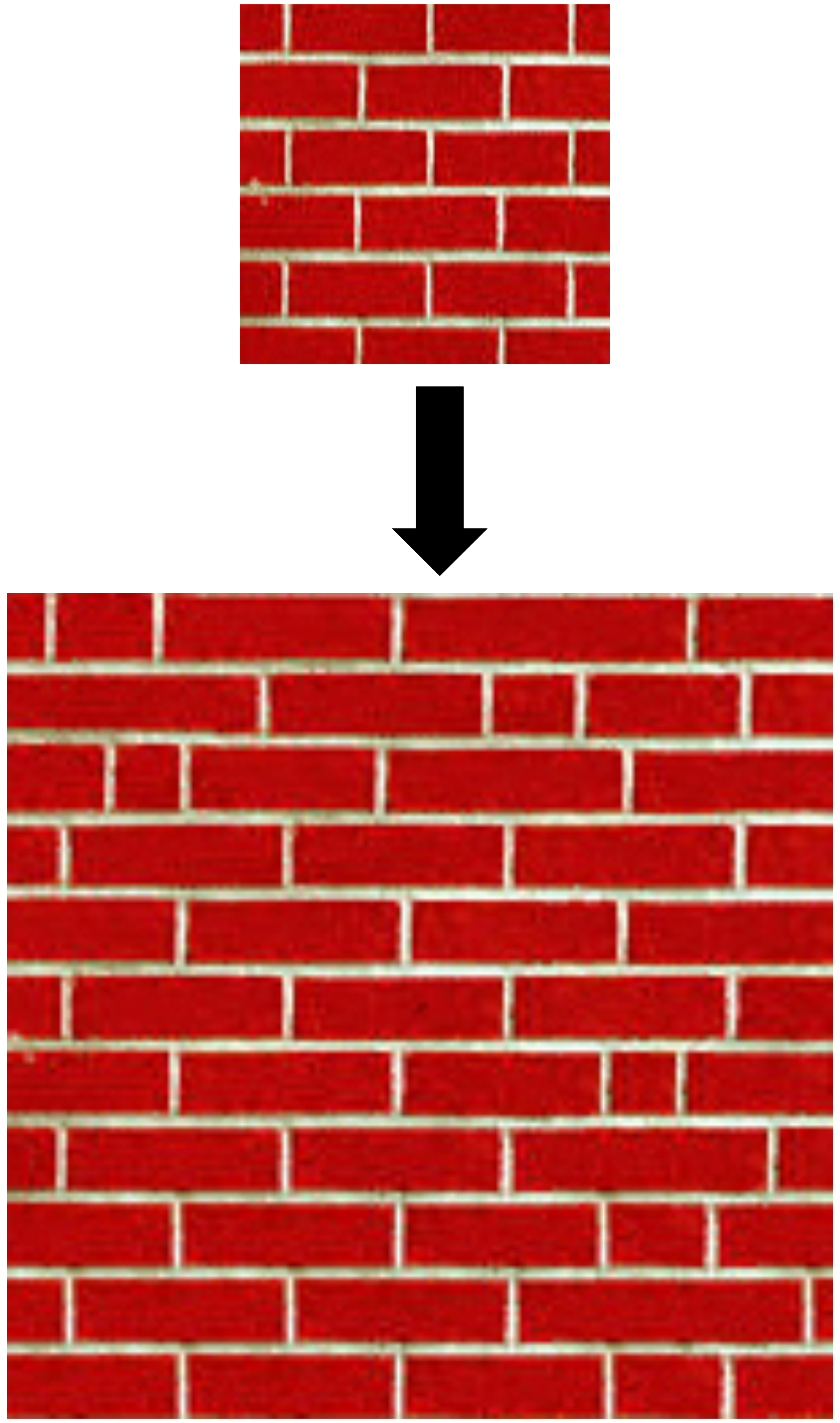


wood granite

Efros and Leung

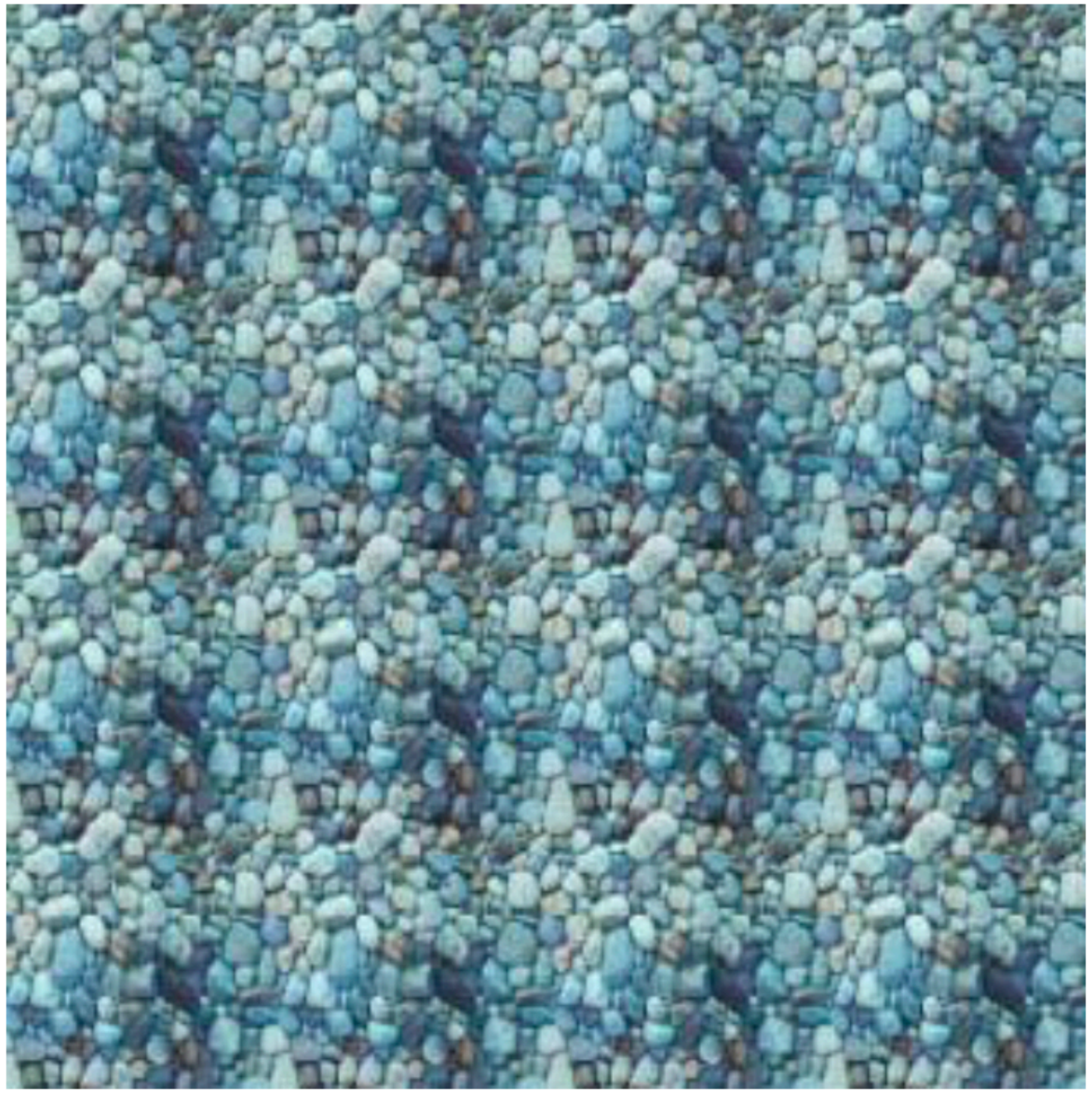


white bread

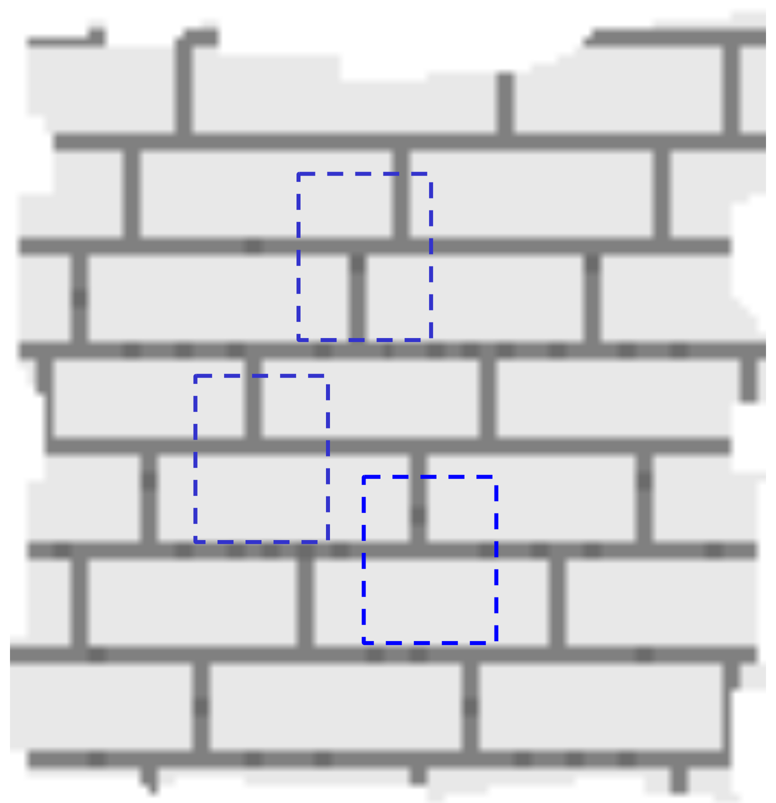


brick wall

Like **Copying**, But not Just Repetition

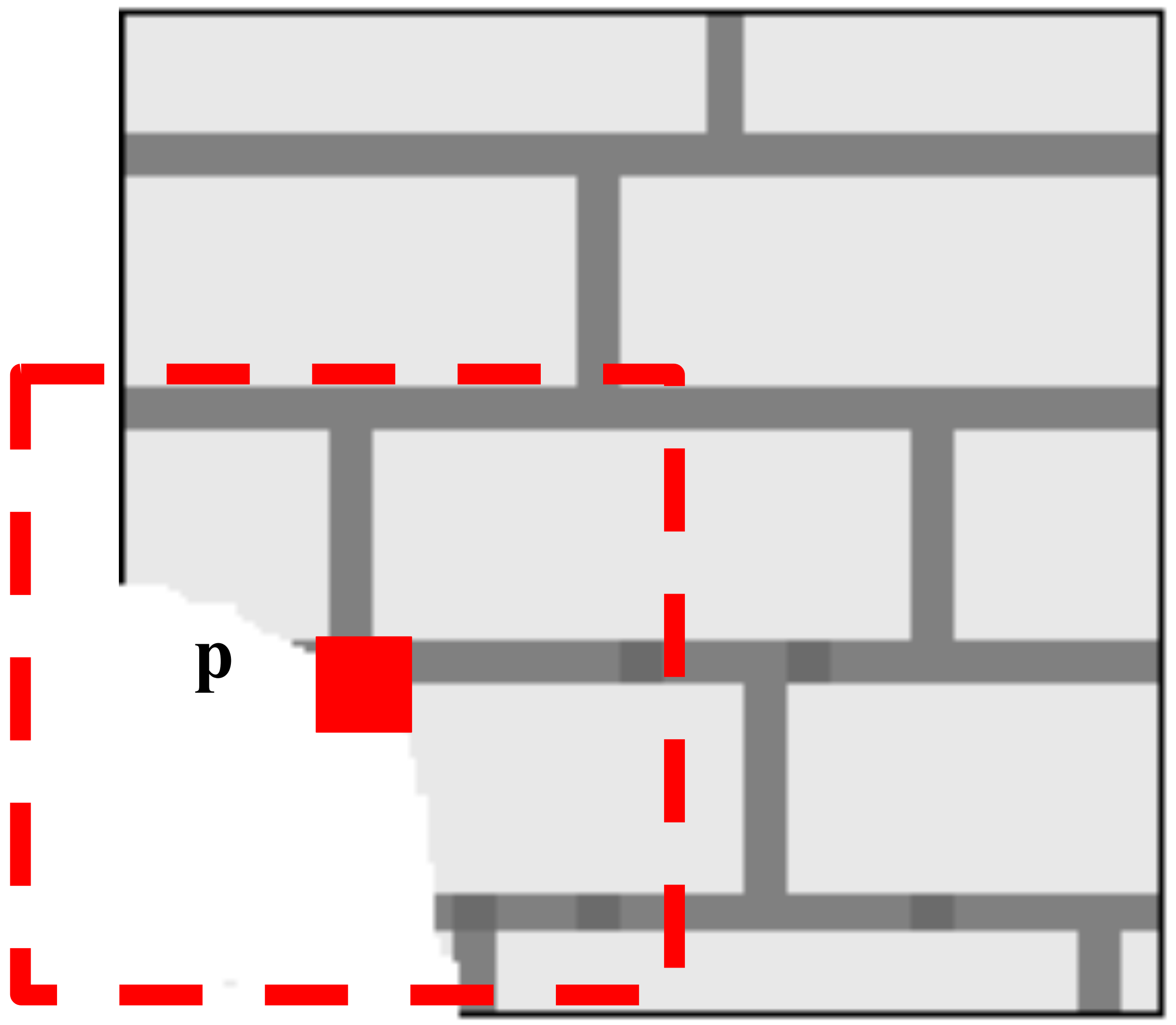


Efros and Leung: Synthesizing One Pixel



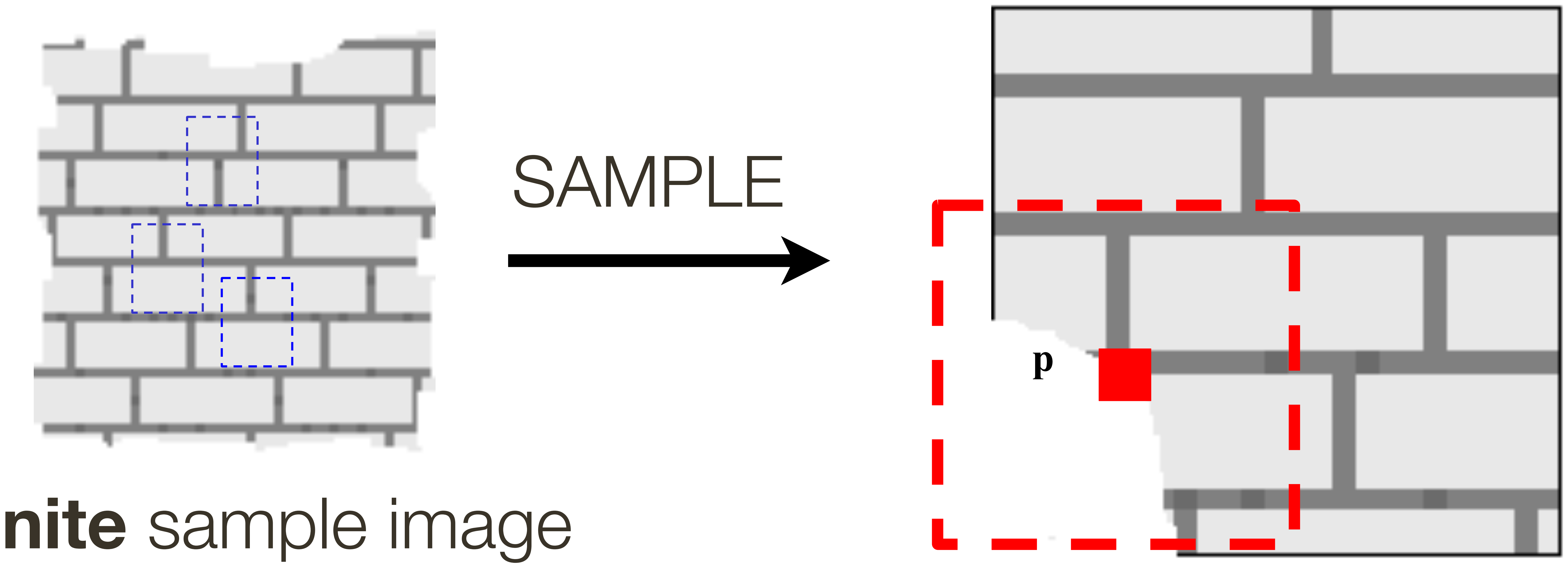
Infinite sample image

SAMPLE



— What is **conditional** probability distribution of *p*, given the neighbourhood window?

Efros and Leung: Synthesizing One Pixel

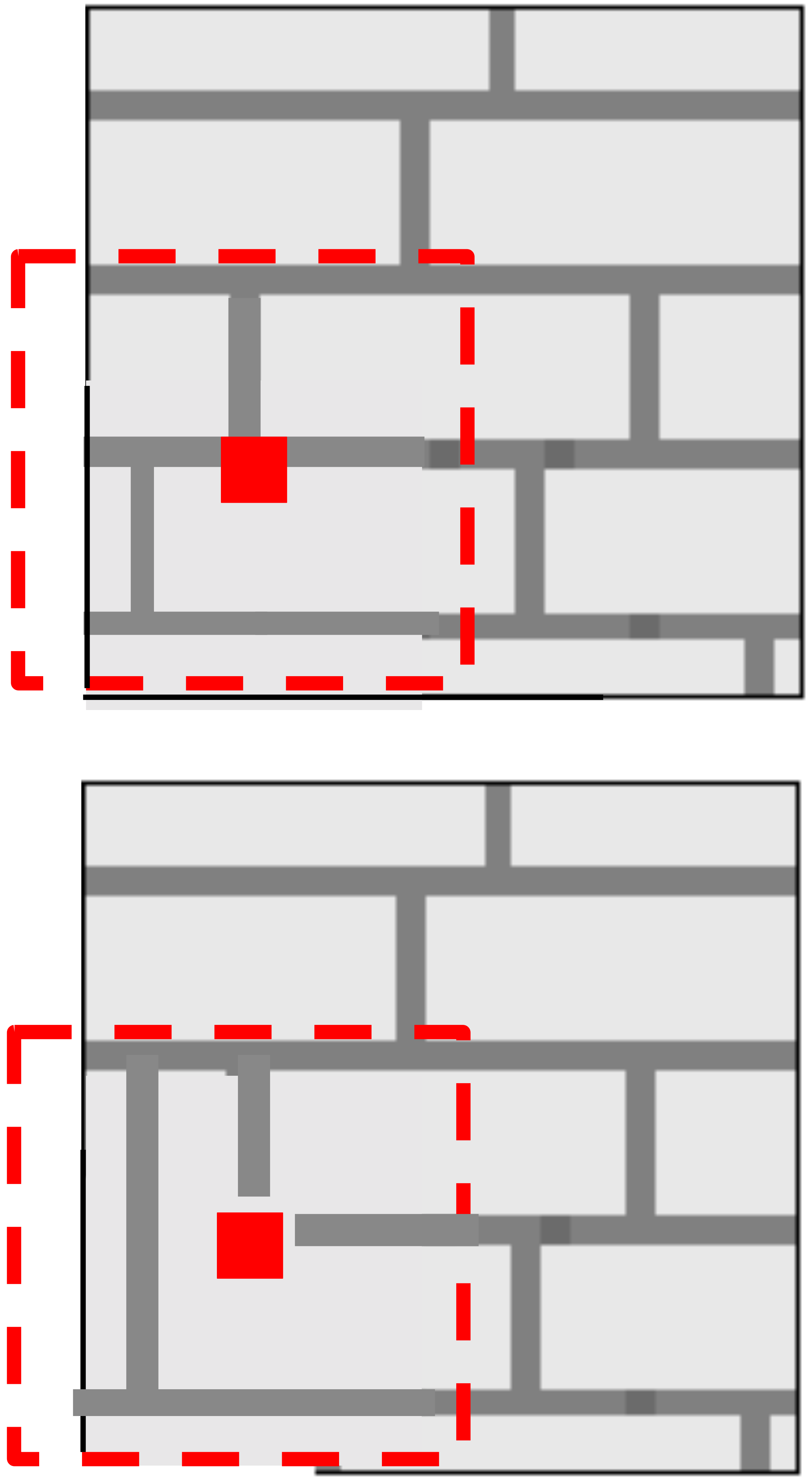


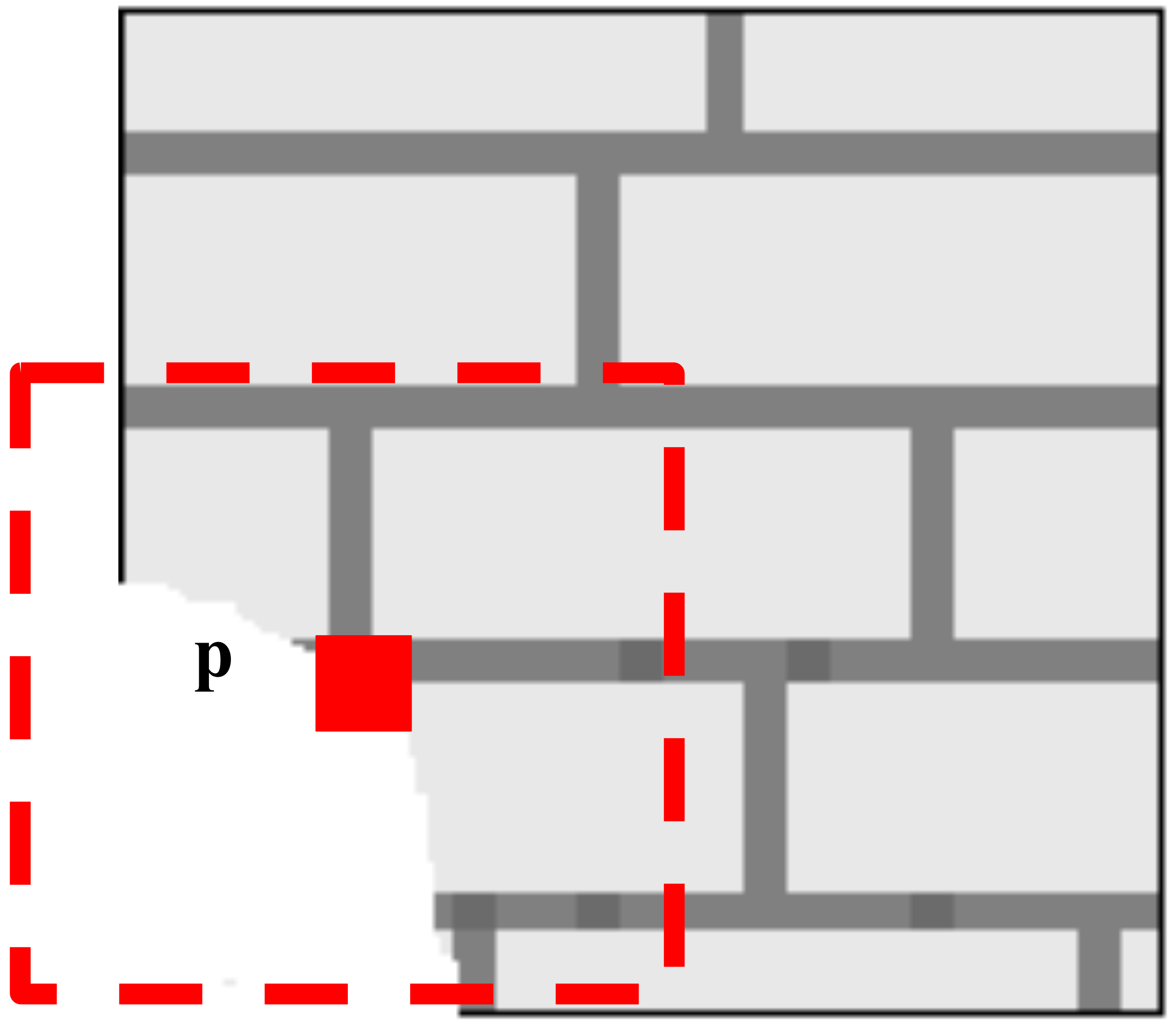
Infinite sample image

— What is **conditional** probability distribution of *p*, given the neighbourhood window?

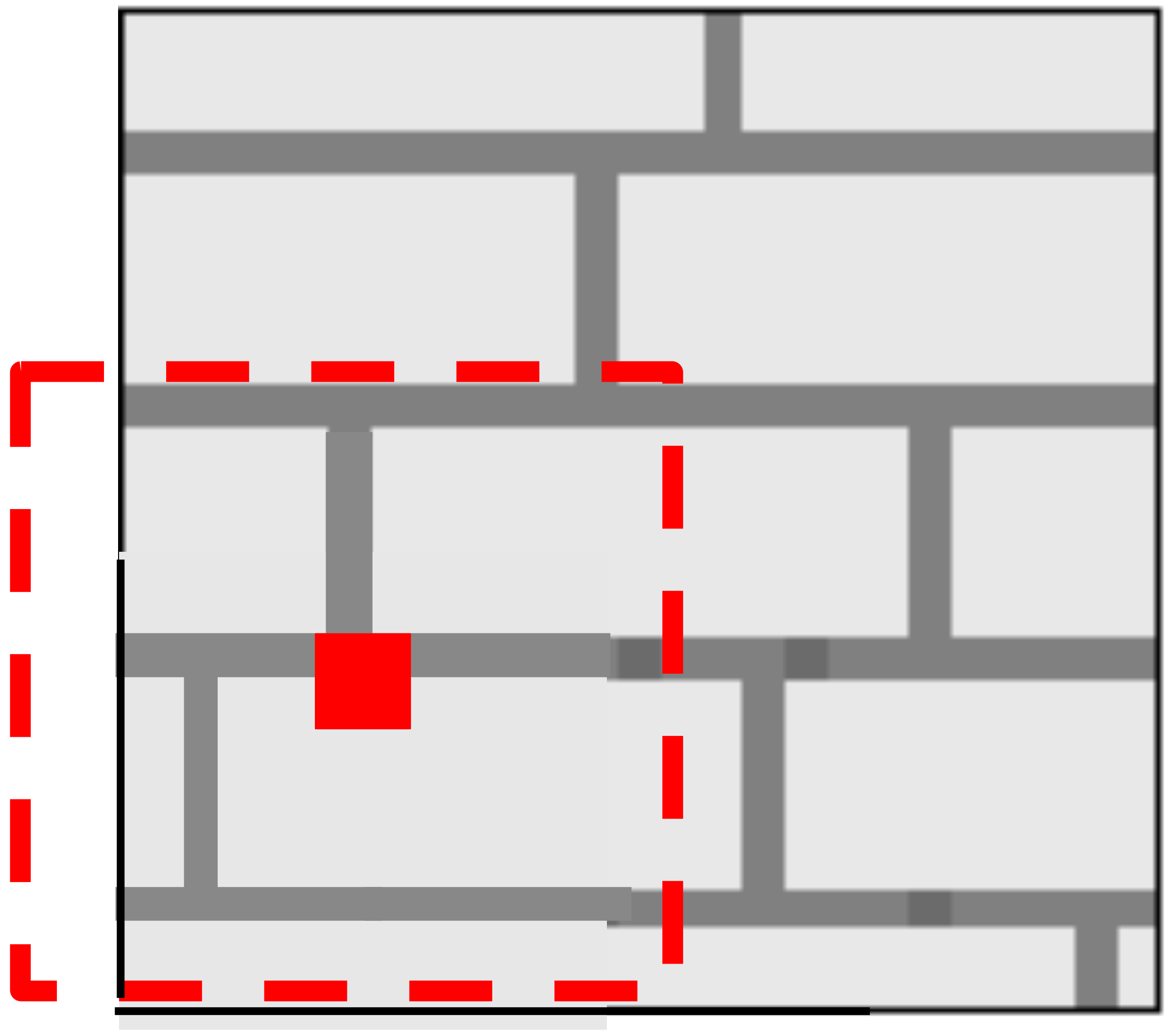
— Directly search the input image for all such neighbourhoods to produce a **histogram** for *p*

Efros and Leung: Synthesizing One Pixel

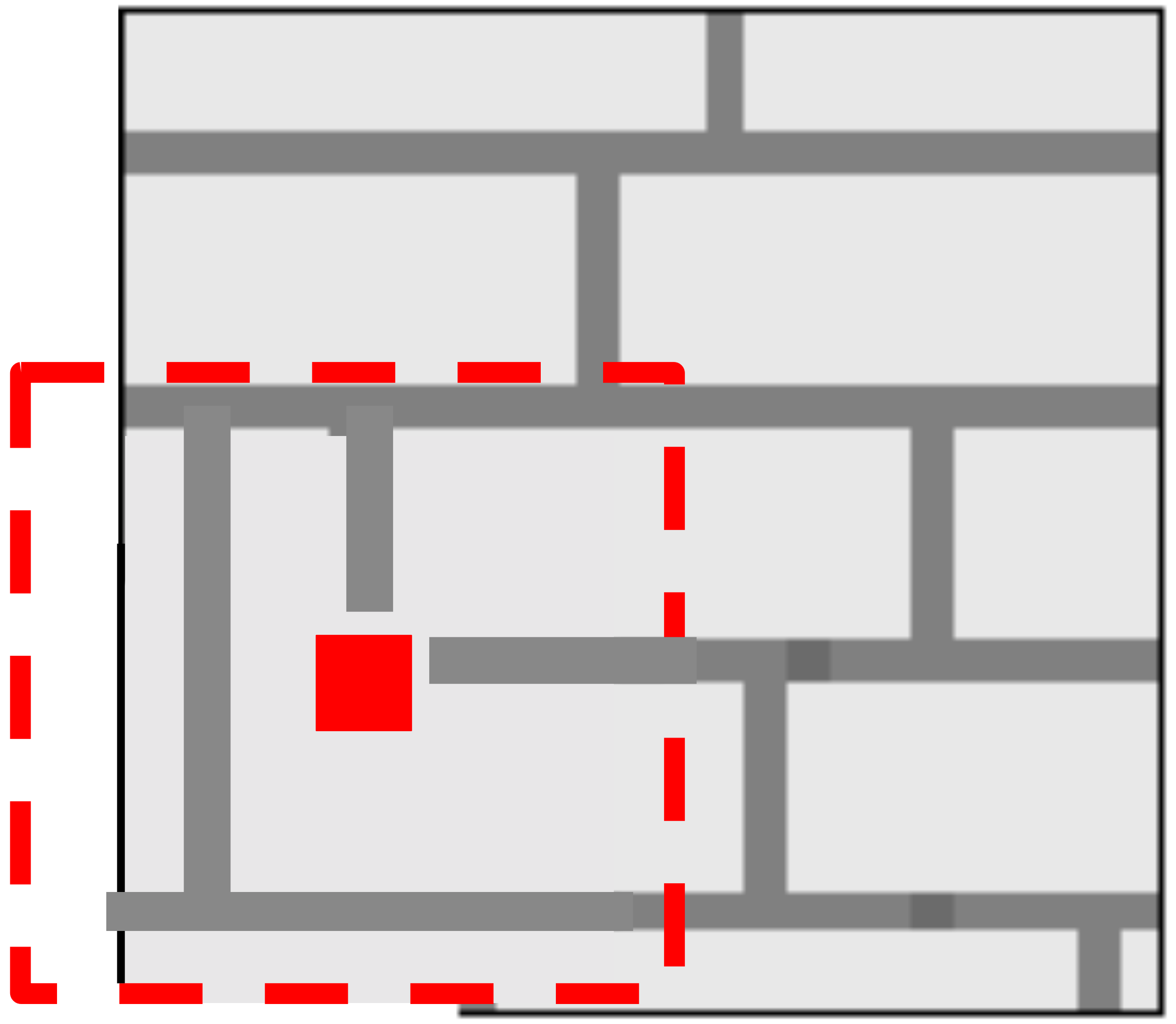




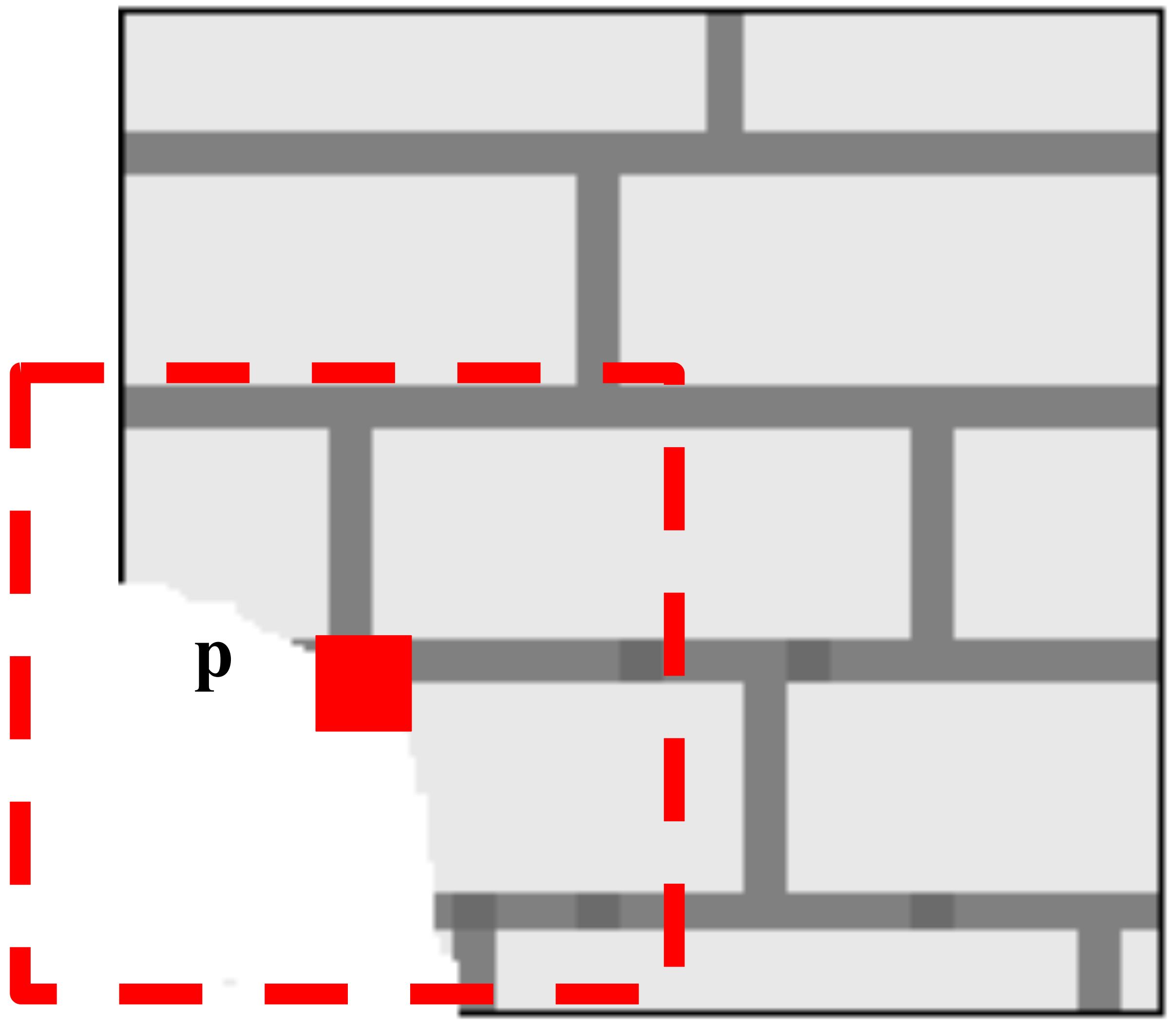
Efros and Leung: Synthesizing One Pixel



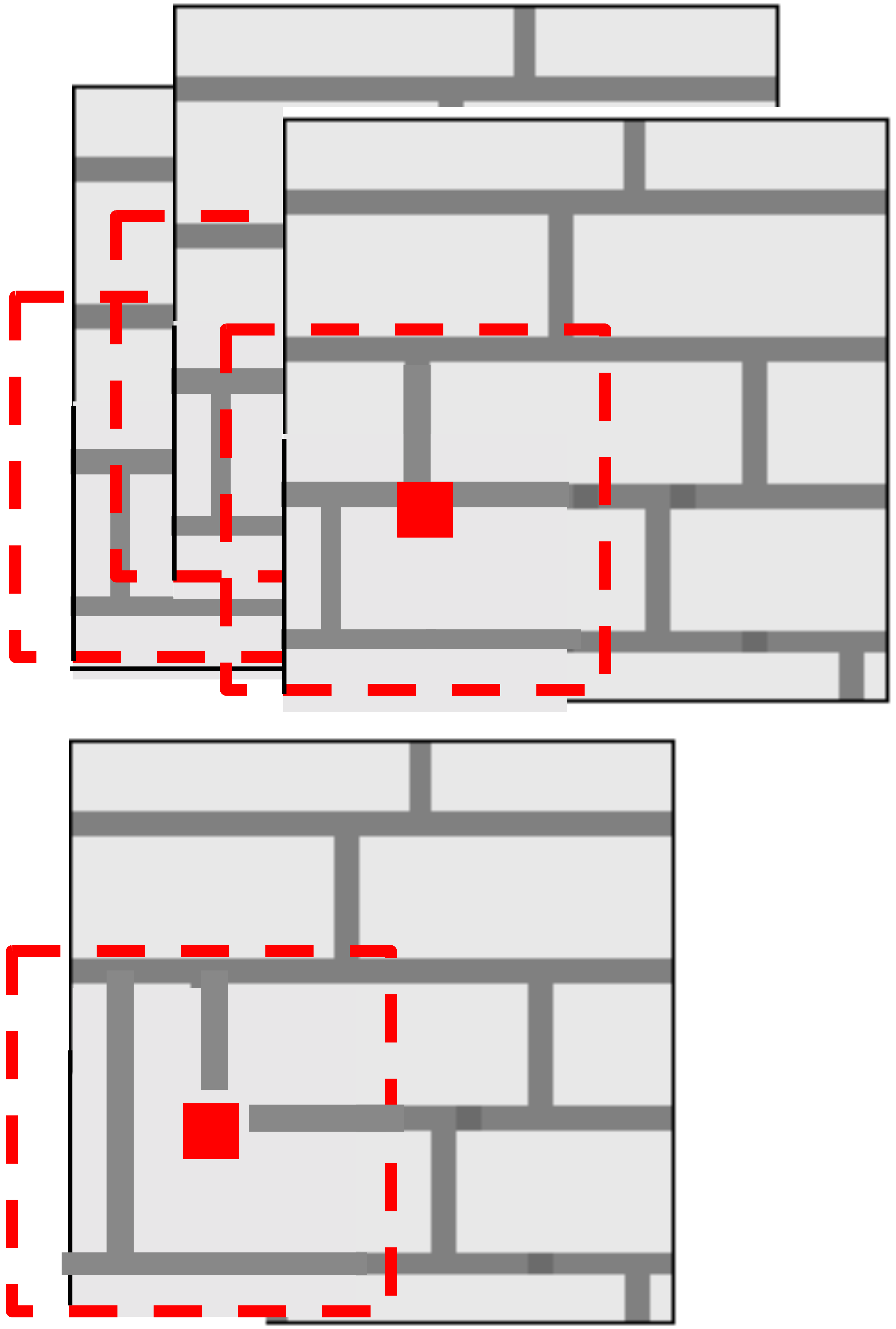
p(dark gray) = 0.5

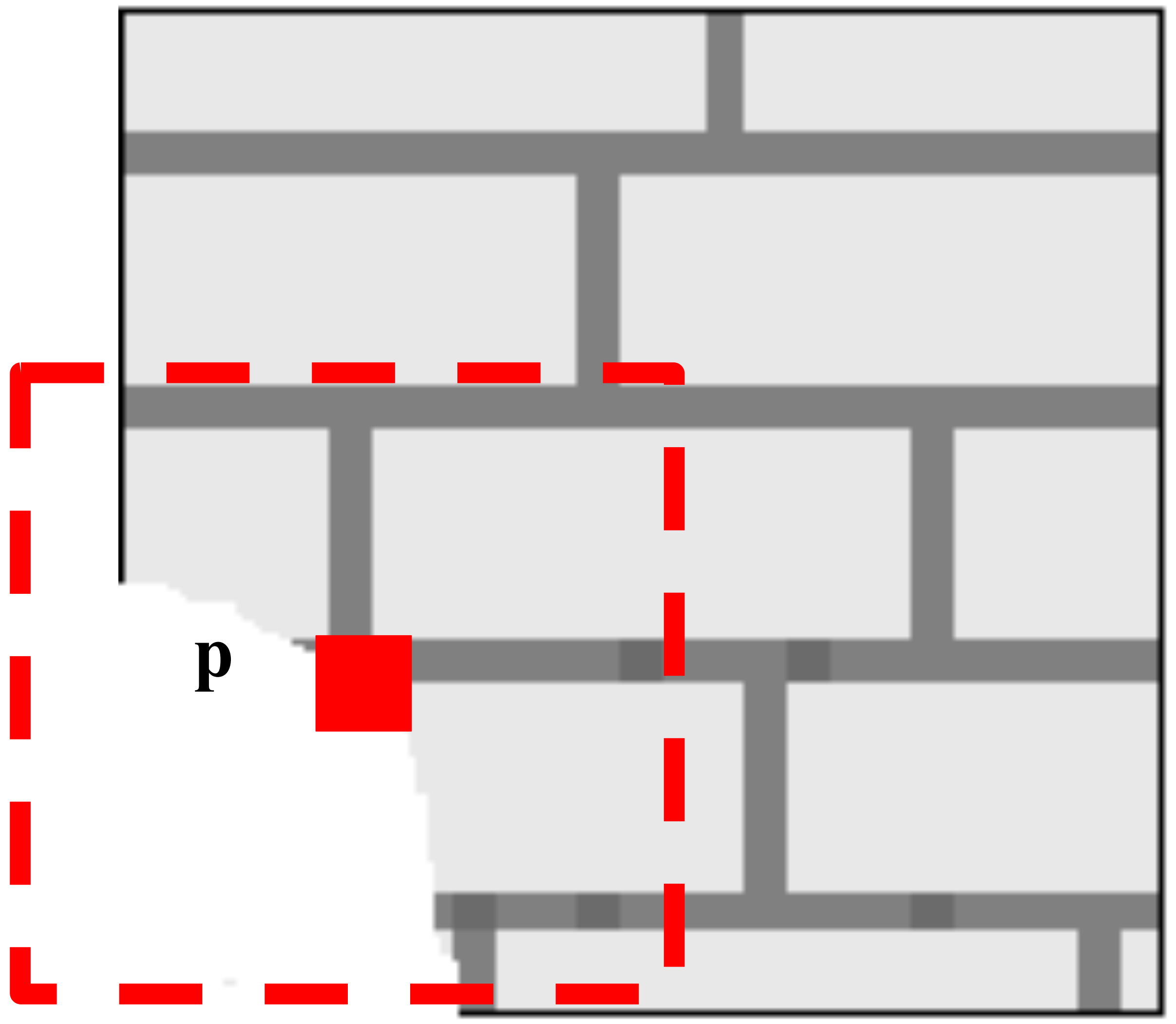


p(light gray) = 0.5

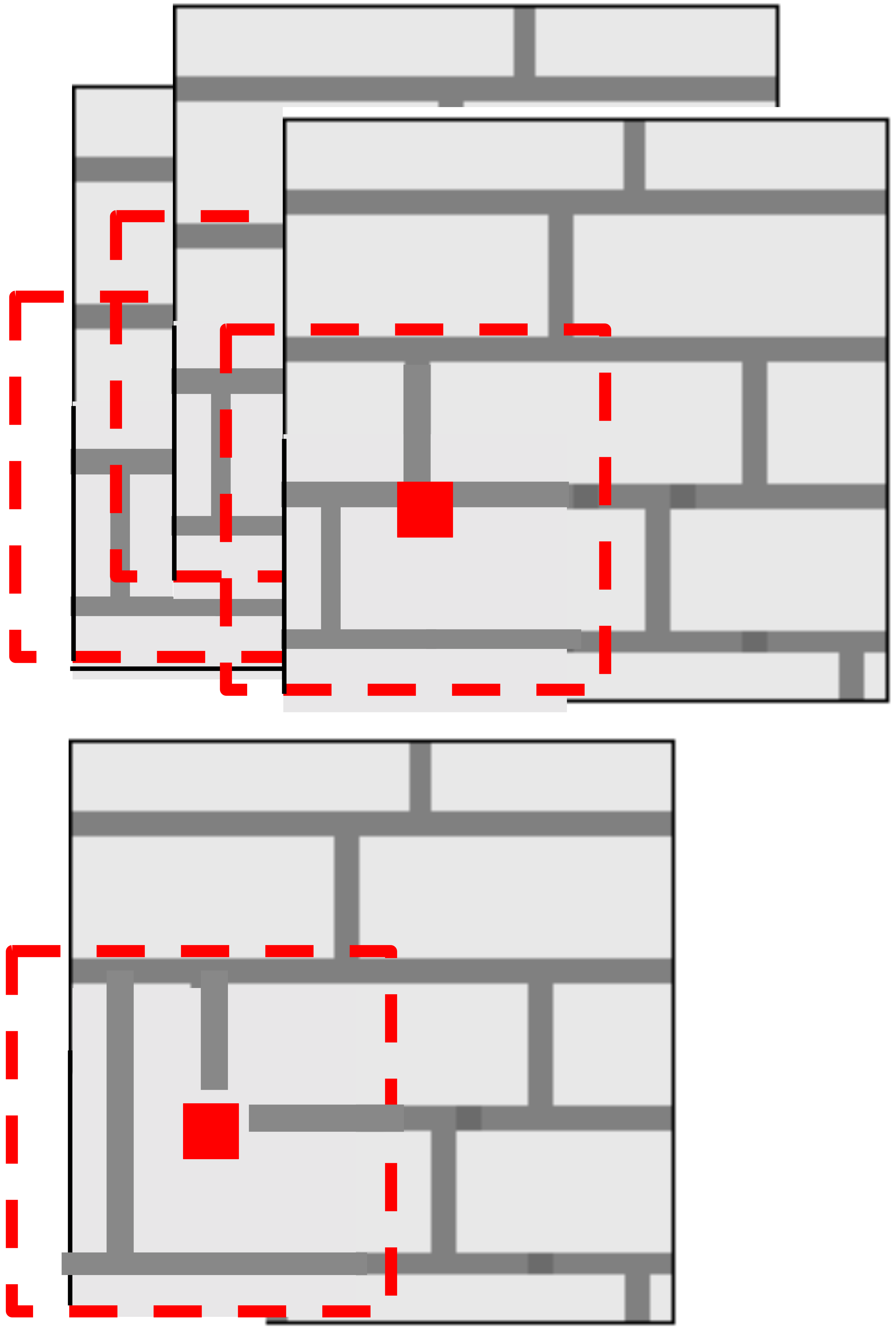


Efros and Leung: Synthesizing One Pixel



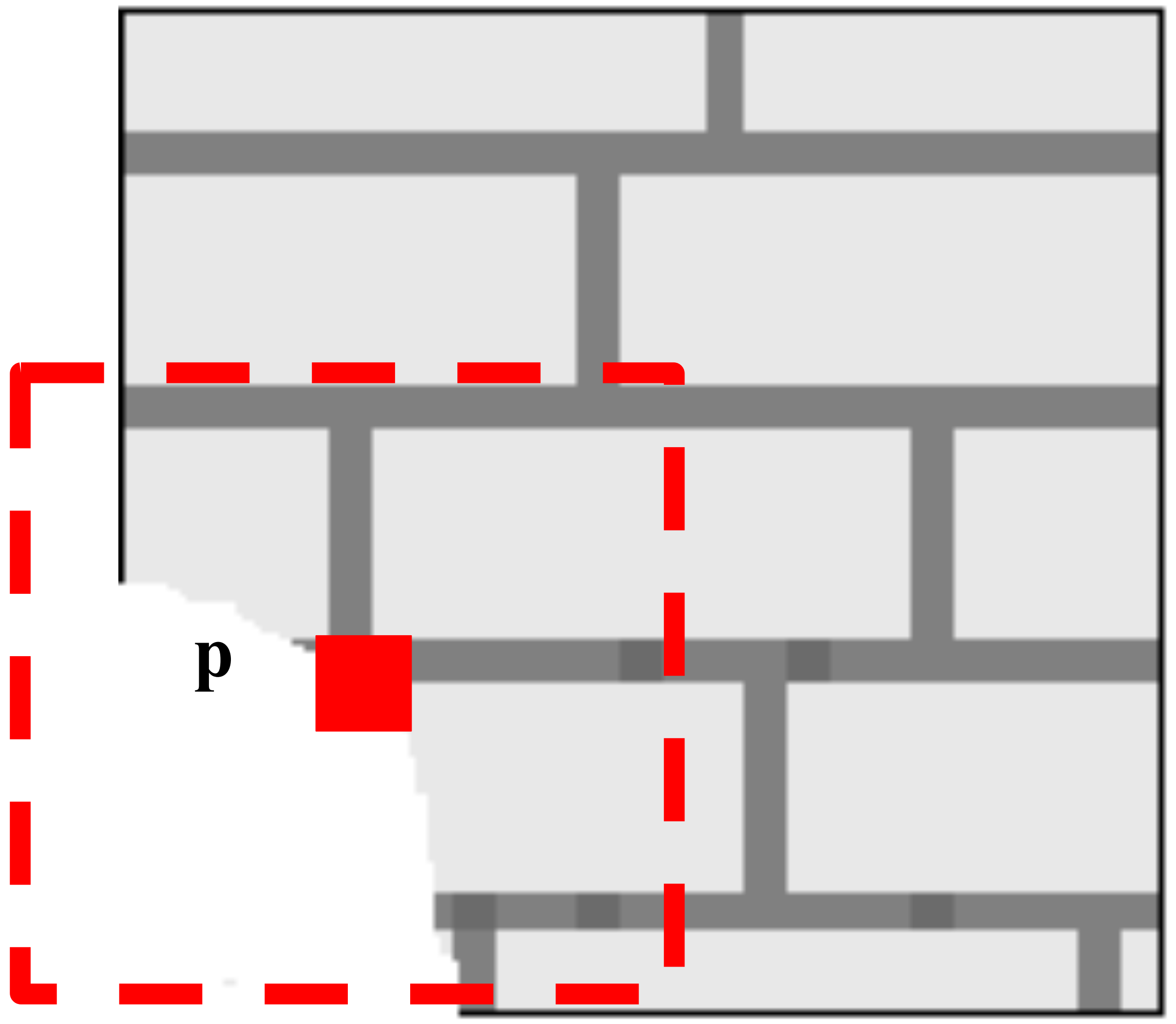


Efros and Leung: Synthesizing One Pixel

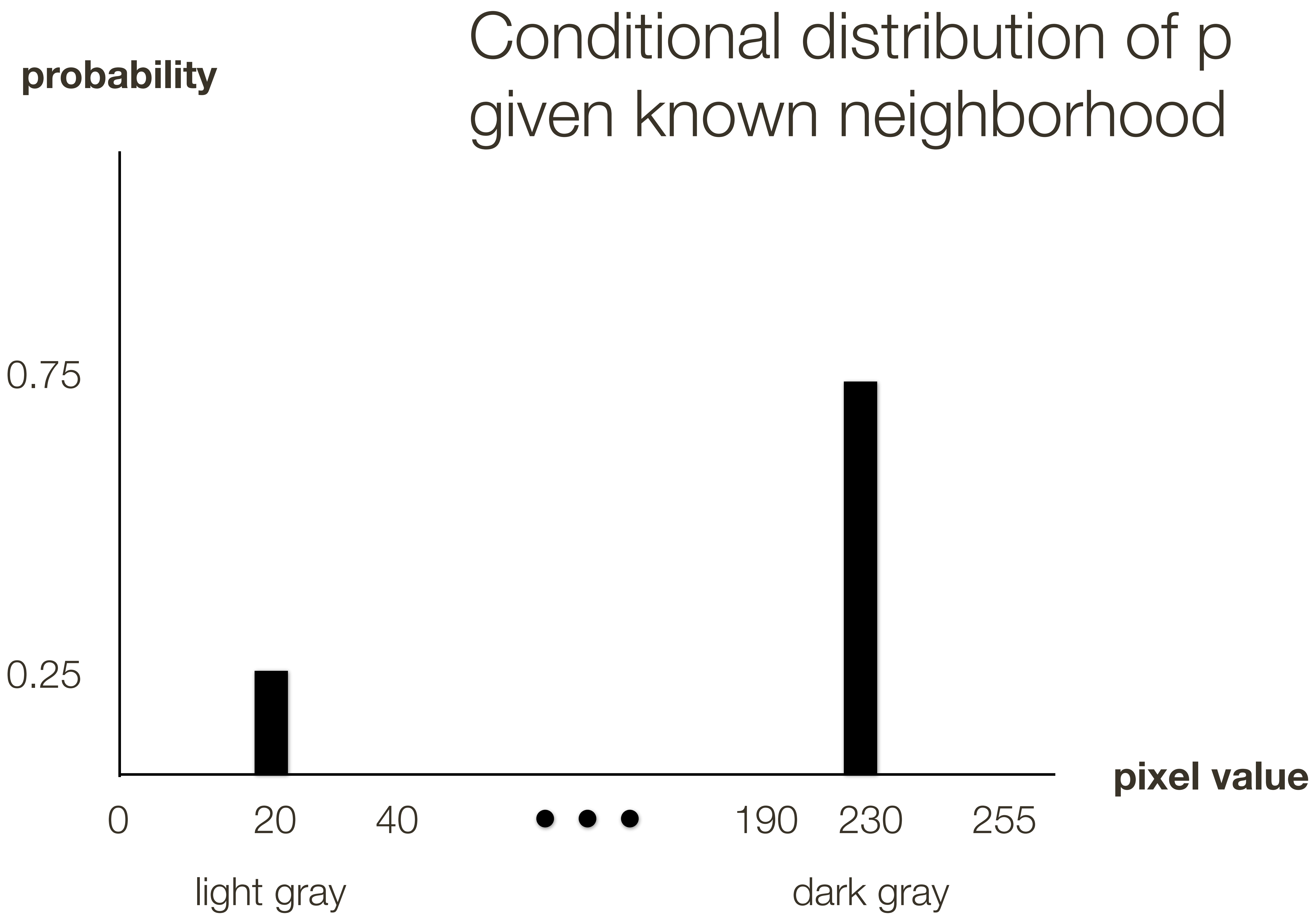


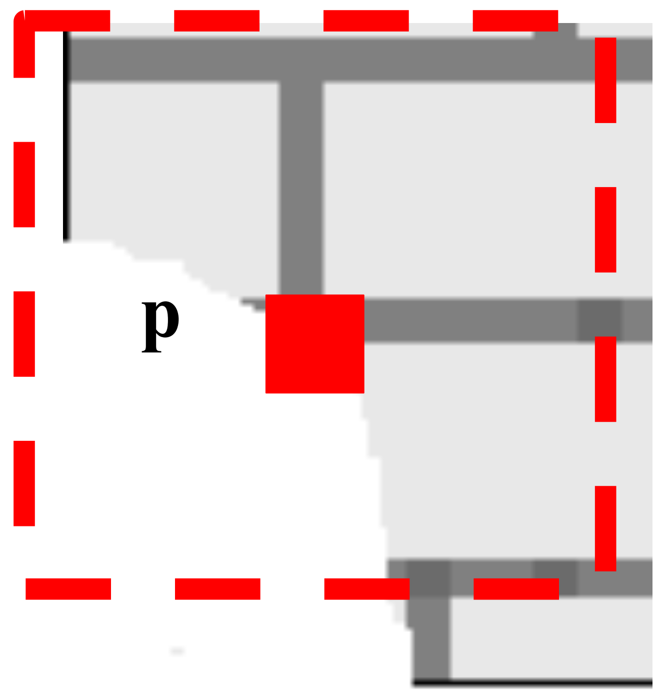
p(dark gray) = 0.75

p(light gray) = 0.25

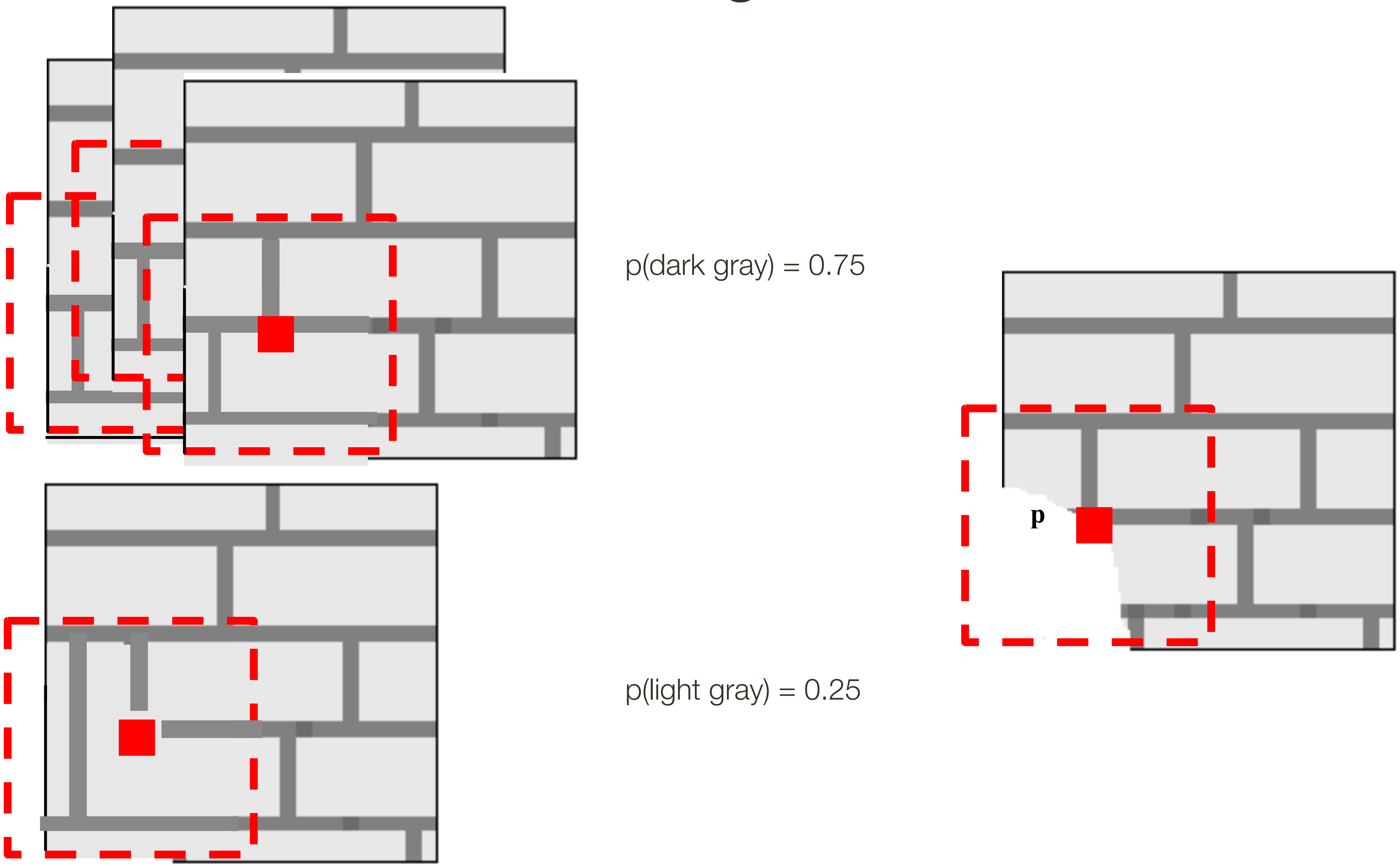


Efros and Leung: Synthesizing One Pixel





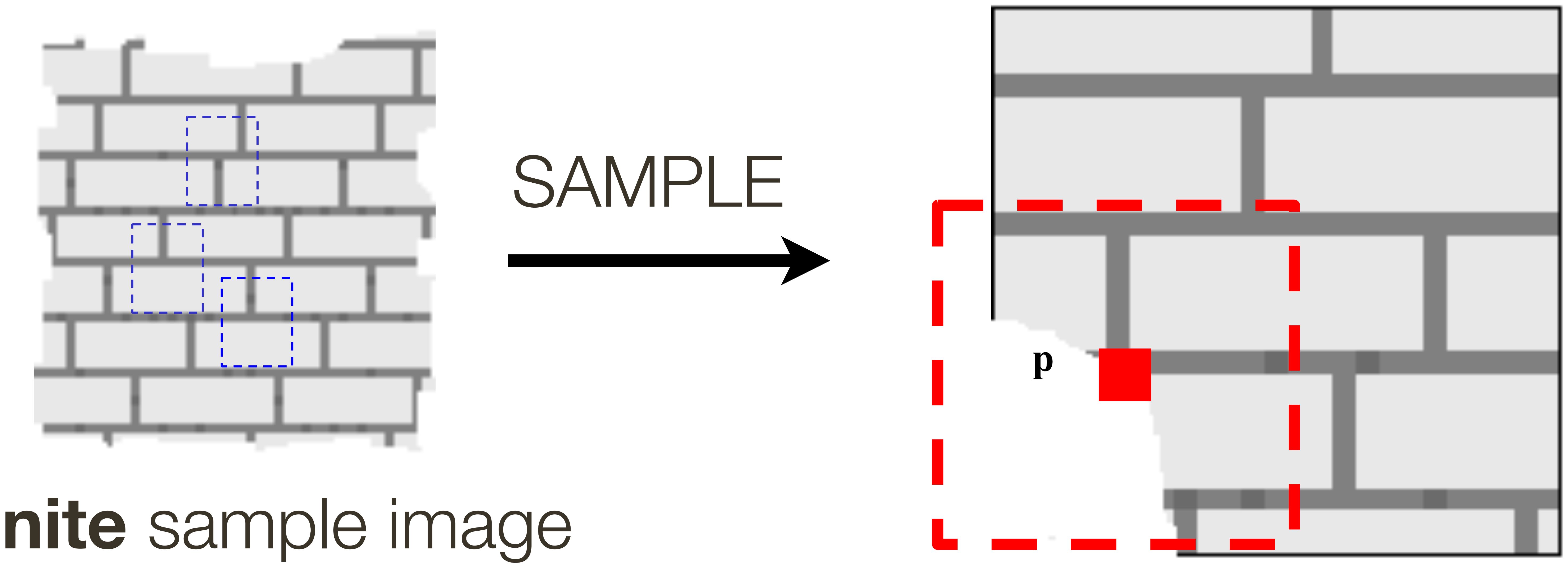
Efros and Leung: Synthesizing One Pixel



p(dark gray) = 0.75

p(light gray) = 0.25

Efros and Leung: Synthesizing One Pixel



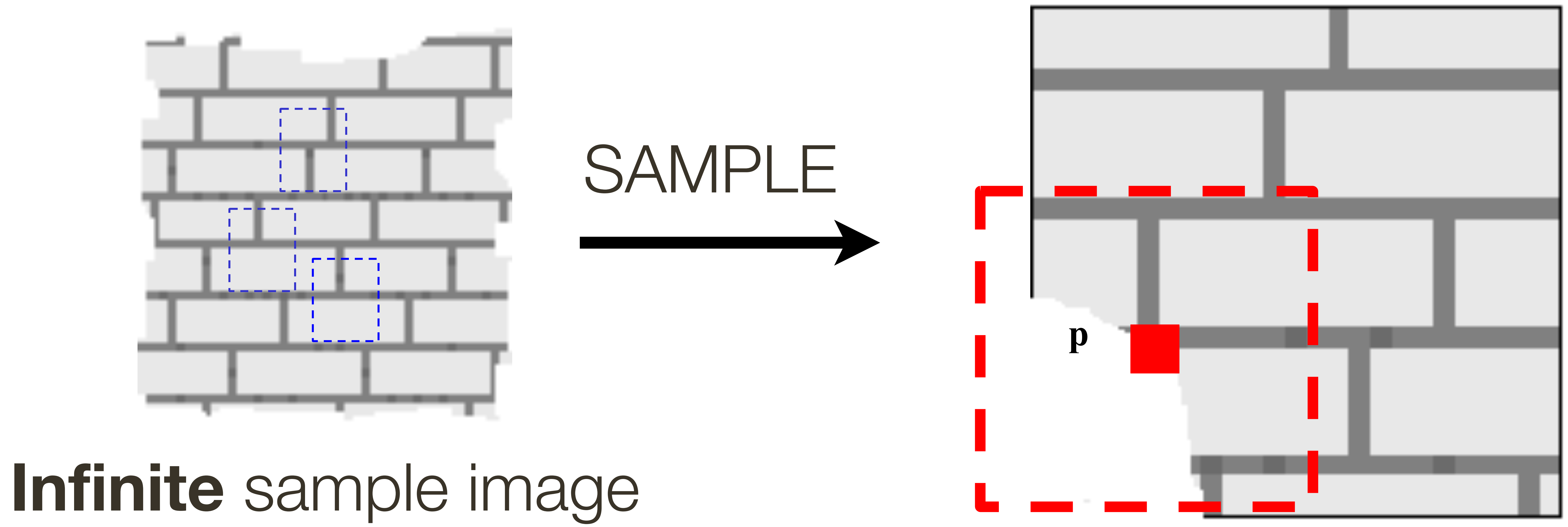
Infinite sample image

— What is **conditional** probability distribution of *p*, given the neighbourhood window?

— Directly search the input image for all such neighbourhoods to produce a **histogram** for *p*

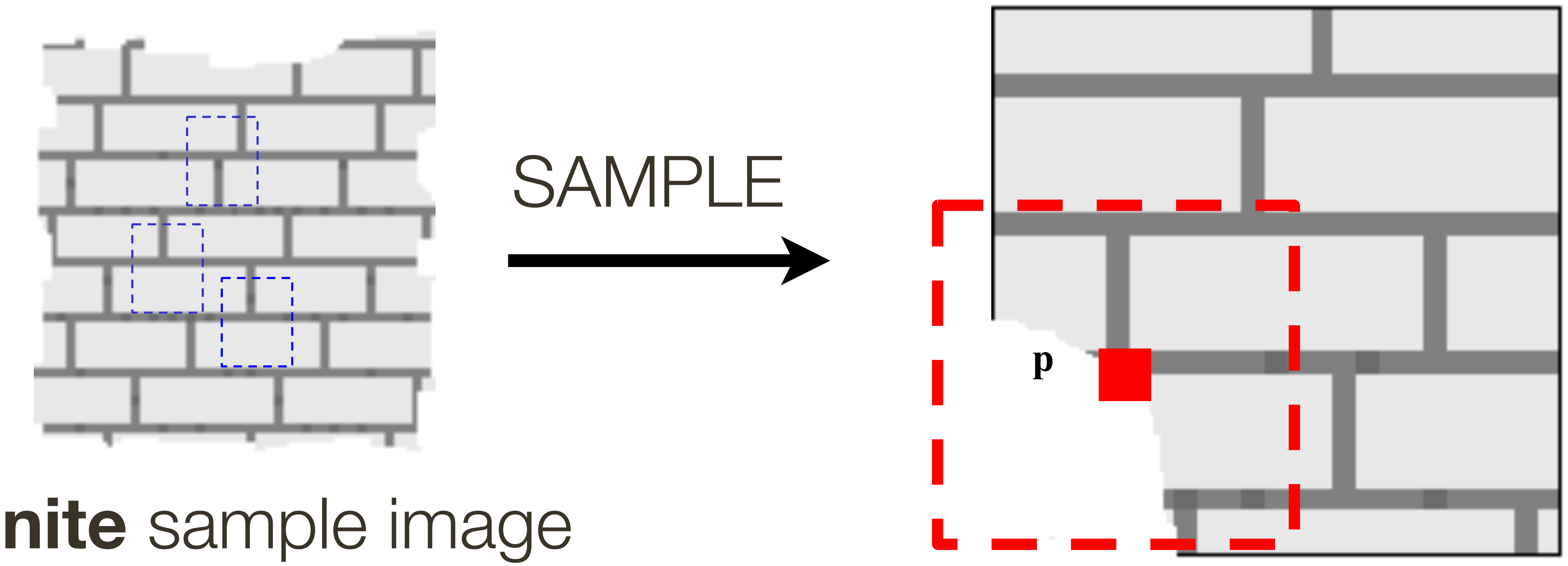
— To **synthesize** *p*, pick one match at random

Efros and Leung: Synthesizing One Pixel



— Since the sample image is finite, an exact neighbourhood match might not be present

Efros and Leung: Synthesizing One Pixel

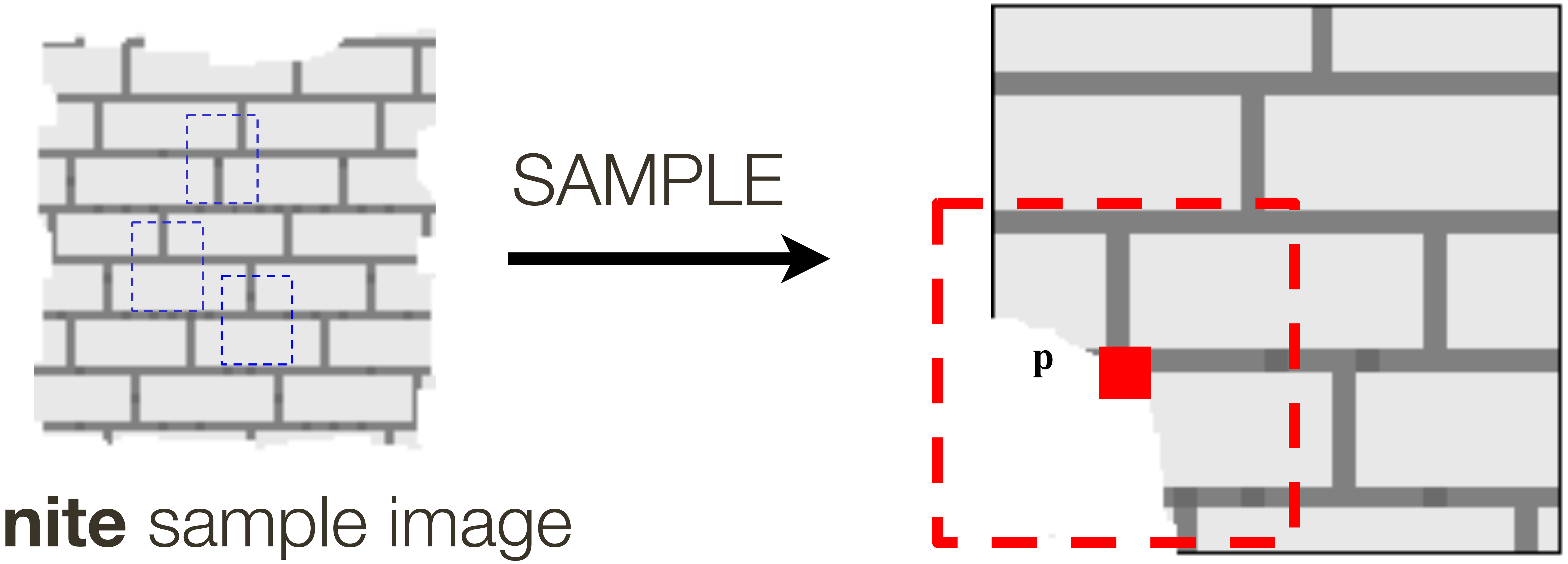


Infinite sample image

— Since the sample image is finite, an exact neighbourhood match might not be present

— Find the **best match** using SSD error, weighted by Gaussian to emphasize local structure, and take all samples within some distance from that match

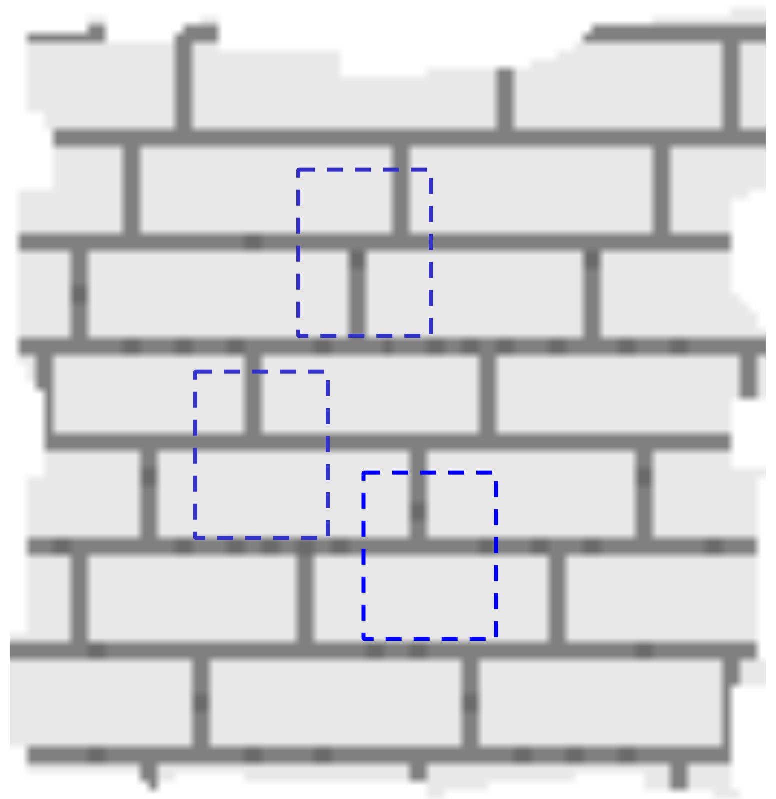
Efros and Leung: Synthesizing One Pixel



Infinite sample image

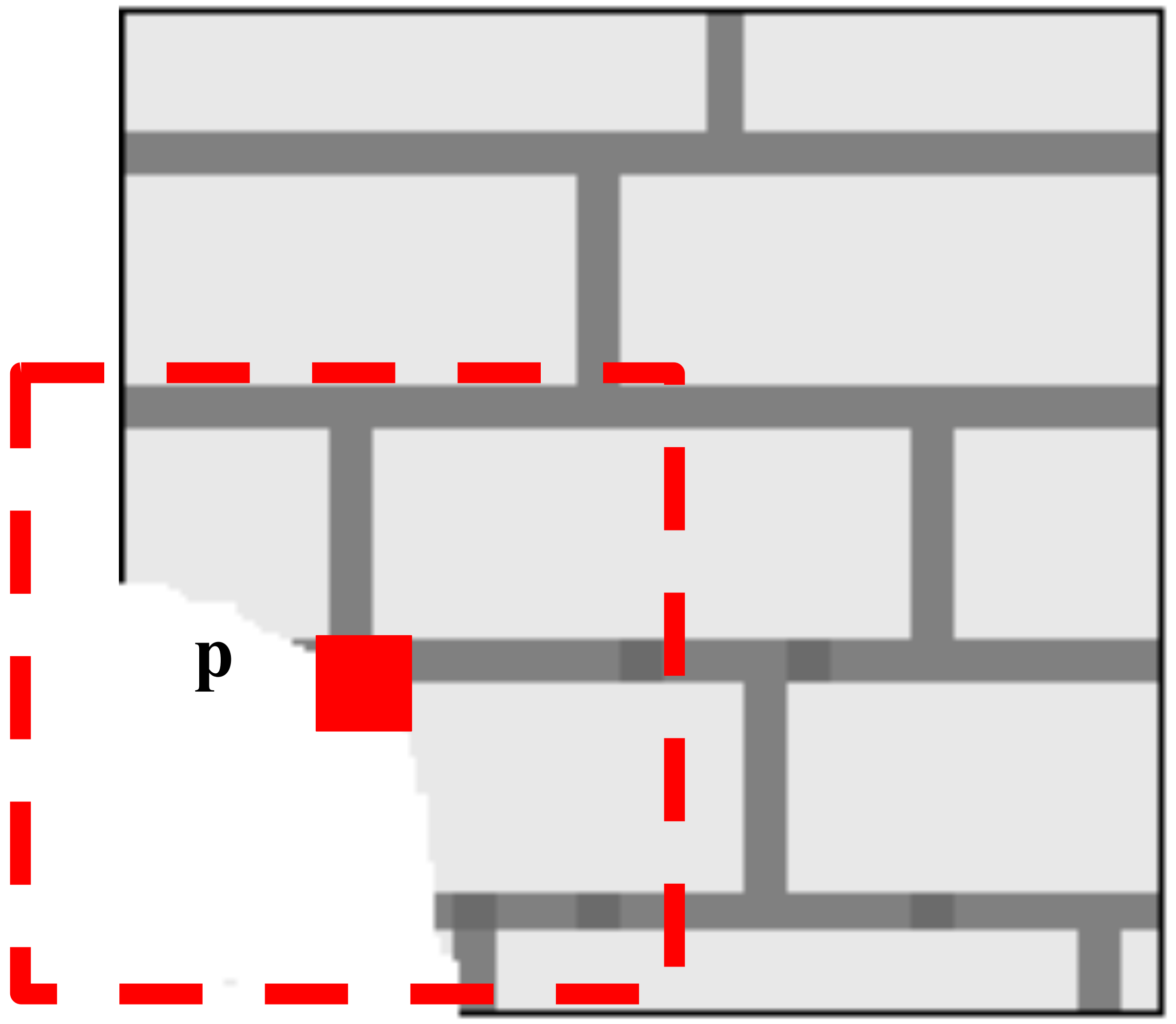
Ranked List	**Similarity (cos)**
x = 5, y = 17	0.87
x = 63, y = 4	0.75
x = 3, y = 44	0.72
x = 123, y = 54	0.64
x = 4, y = 57	0.60
⋮	⋮

Efros and Leung: Synthesizing One Pixel



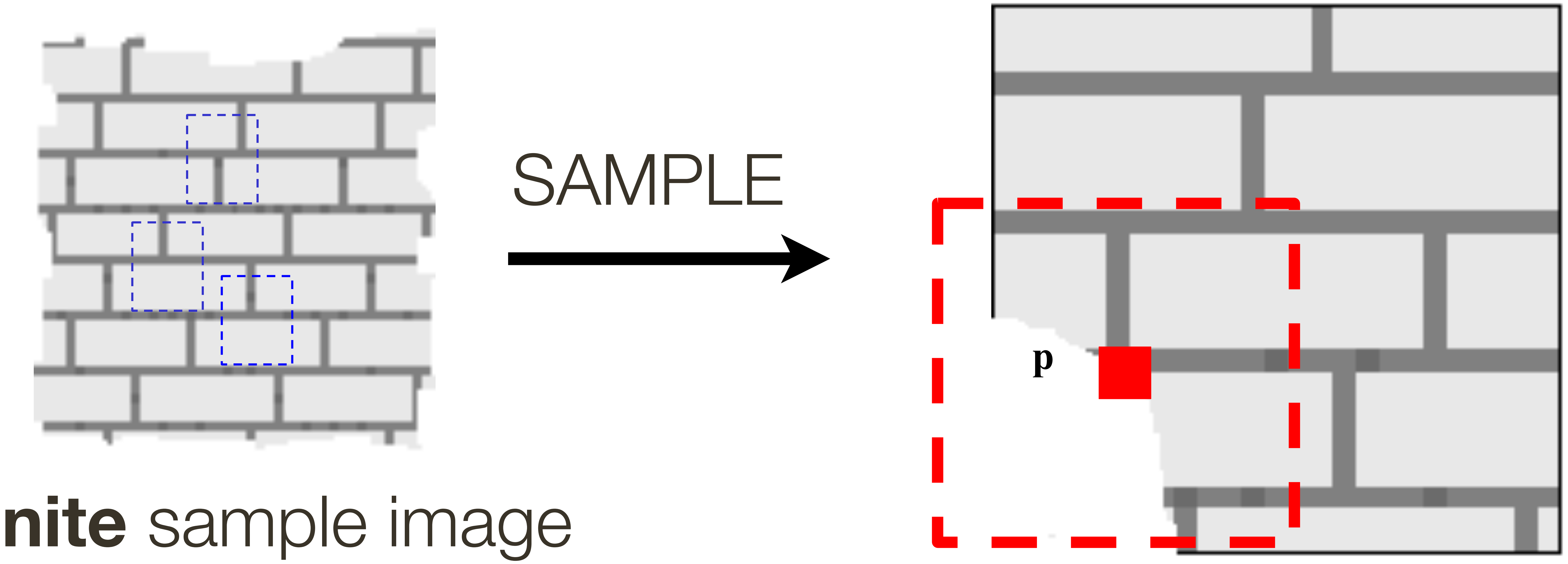
Infinite sample image

SAMPLE



Ranked List	**Similarity (cos)**	
x = 5, y = 17	0.87	⟵ best match
x = 63, y = 4	0.75	
x = 3, y = 44	0.72	
x = 123, y = 54	0.64	
x = 4, y = 57	0.60	
⋮	⋮	

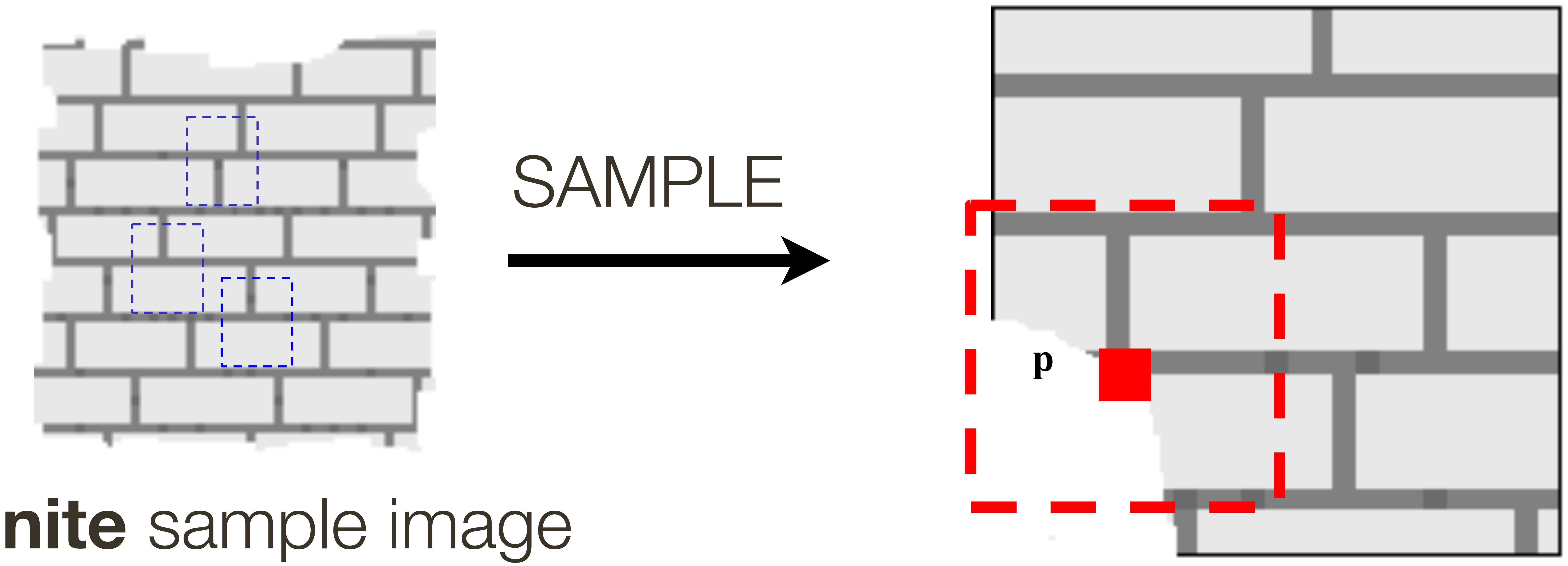
Efros and Leung: Synthesizing One Pixel



Infinite sample image

Ranked List	**Similarity (cos)**	
x = 5, y = 17	0.87	← best match
x = 63, y = 4	0.75	
x = 3, y = 44	0.72	
		threshold = best match * **0.8** = 0.696
x = 123, y = 54	0.64	
x = 4, y = 57	0.60	
⋮	⋮	

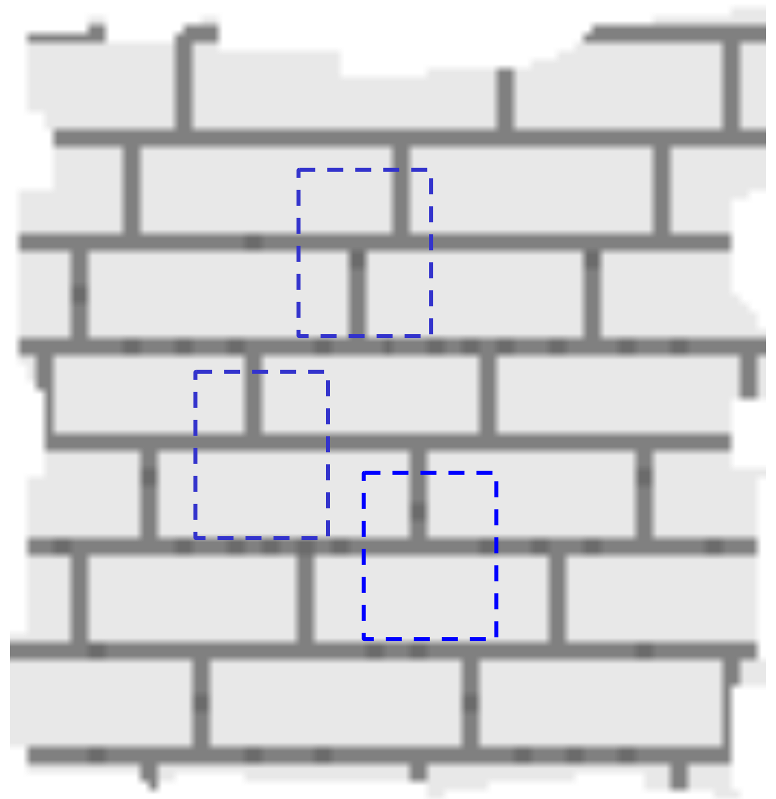
Efros and Leung: Synthesizing One Pixel



Infinite sample image

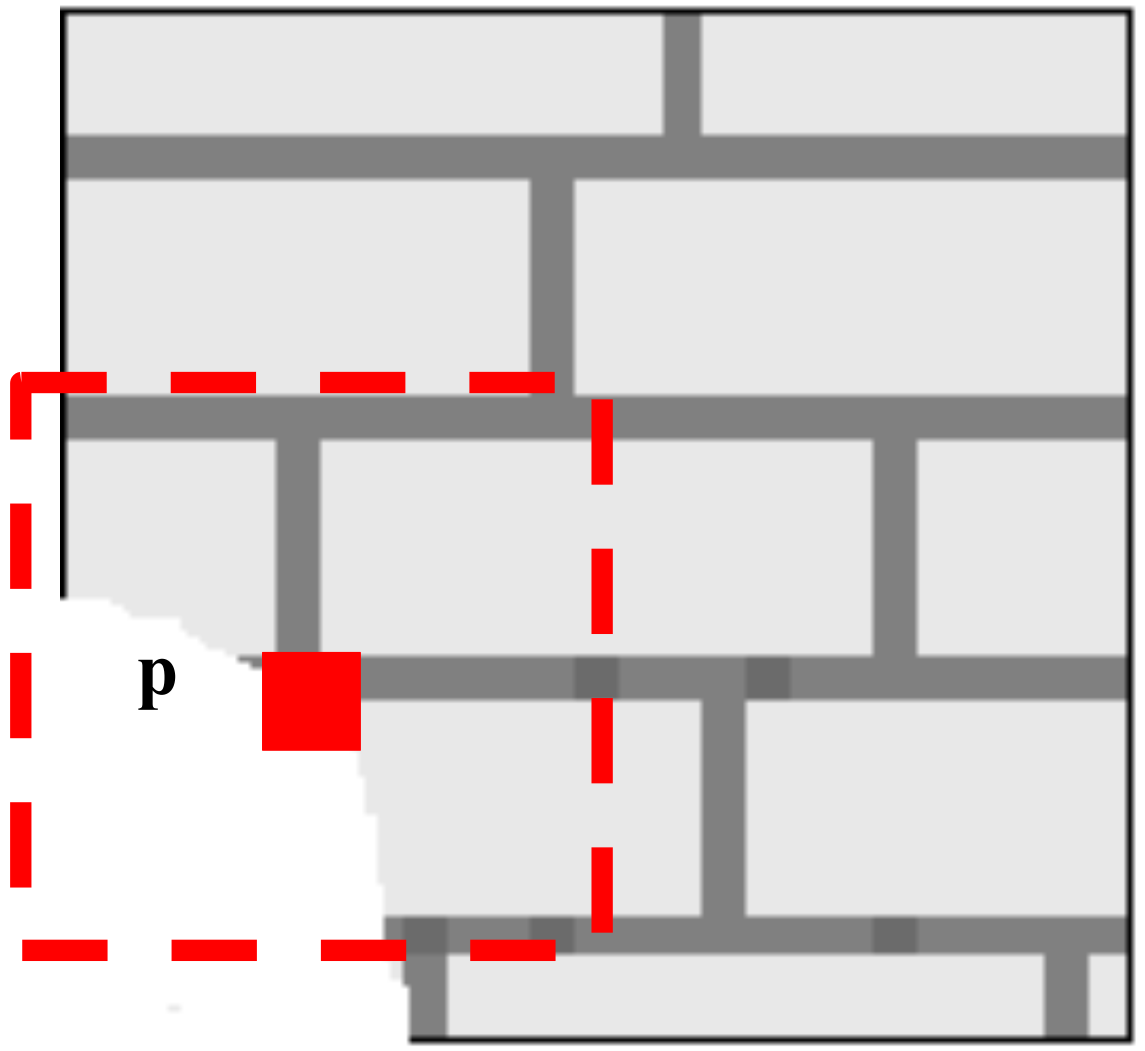
Ranked List	**Similarity (cos)**	
x = 5, y = 17	**0.87**	← best match
x = 63, y = 4	**0.75**	
x = 3, y = 44	**0.72**	
		threshold = best match * **0.8** = 0.696
x = 123, y = 54	0.64	
x = 4, y = 57	0.60	
⋮	⋮	

Efros and Leung: Synthesizing One Pixel



Infinite sample image

SAMPLE

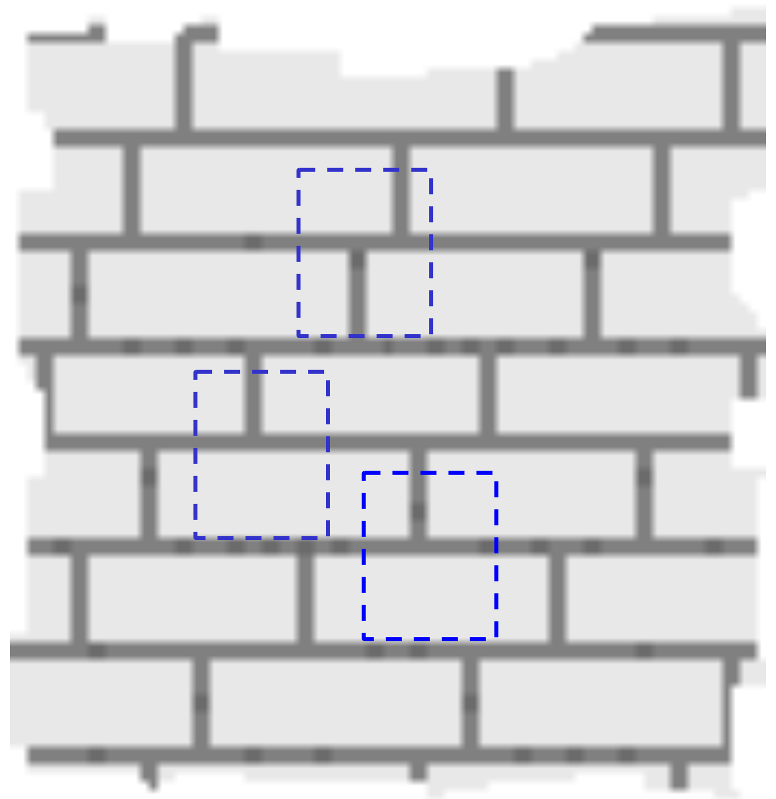


Ranked List	**Similarity (cos)**
x = 5, y = 17	**0.87**
x = 63, y = 4	**0.75**
x = 3, y = 44	**0.72**
x = 123, y = 54	0.64
x = 4, y = 57	0.60
⋮	⋮

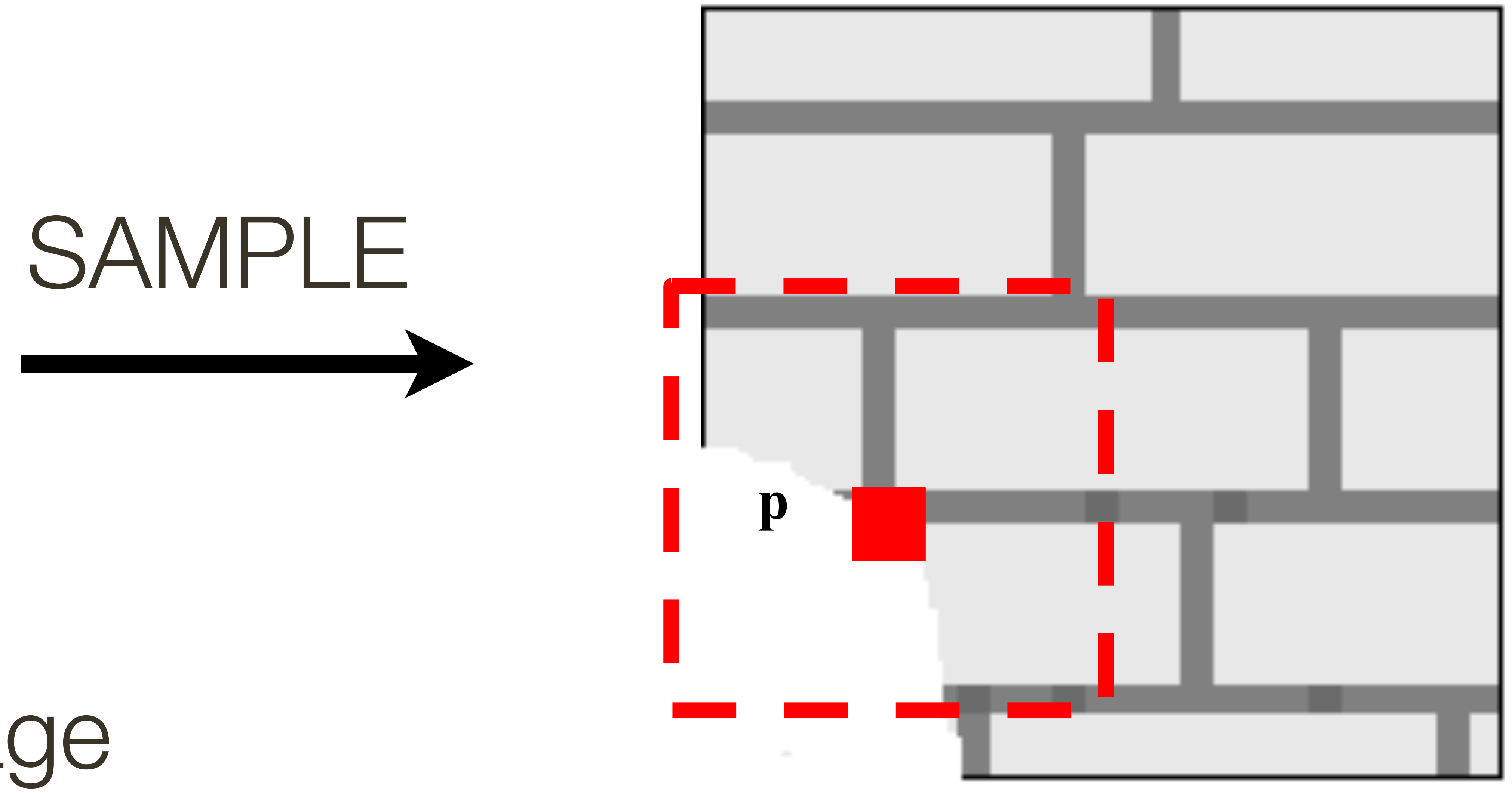
pick one at random and copy target pixel from it

threshold = best match * **0.8** = 0.696

Efros and Leung: Synthesizing One Pixel



Infinite sample image



Ranked List	**Similarity (ssd)**
x = 5, y = 17	**0.13**
x = 63, y = 4	**0.25**
x = 3, y = 44	**0.28**
x = 123, y = 54	0.36
x = 4, y = 57	0.40
⋮	⋮

pick one at random and copy target pixel from it

threshold = best match * **2.5** = 0.325

Efros and Leung: Synthesizing Many Pixels

For multiple pixels, "grow" the texture in layers

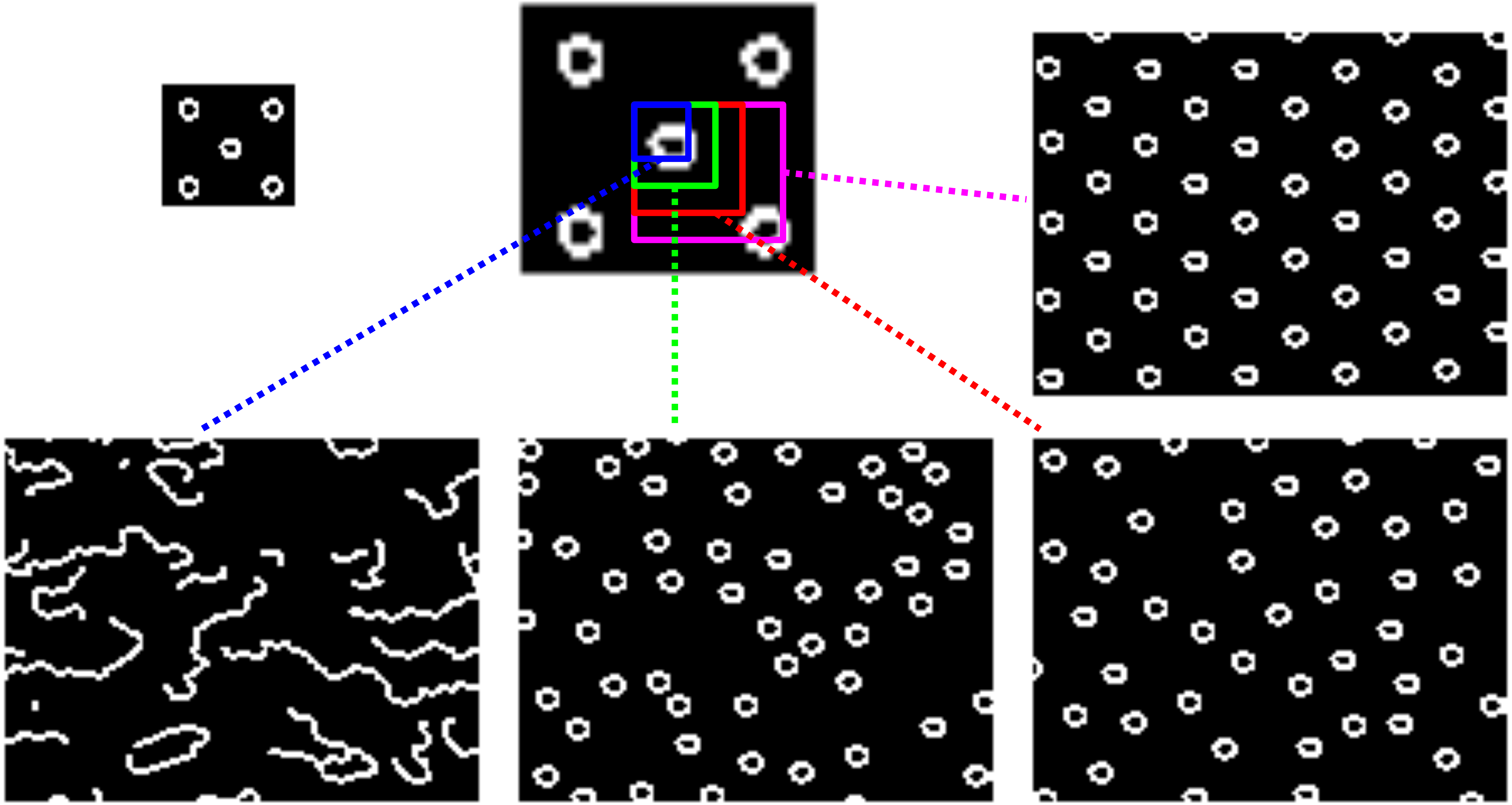
— In the case of hole-filling, start from the edges of the hole

For an interactive demo, see

https://una-dinosauria.github.io/efros-and-leung-js/

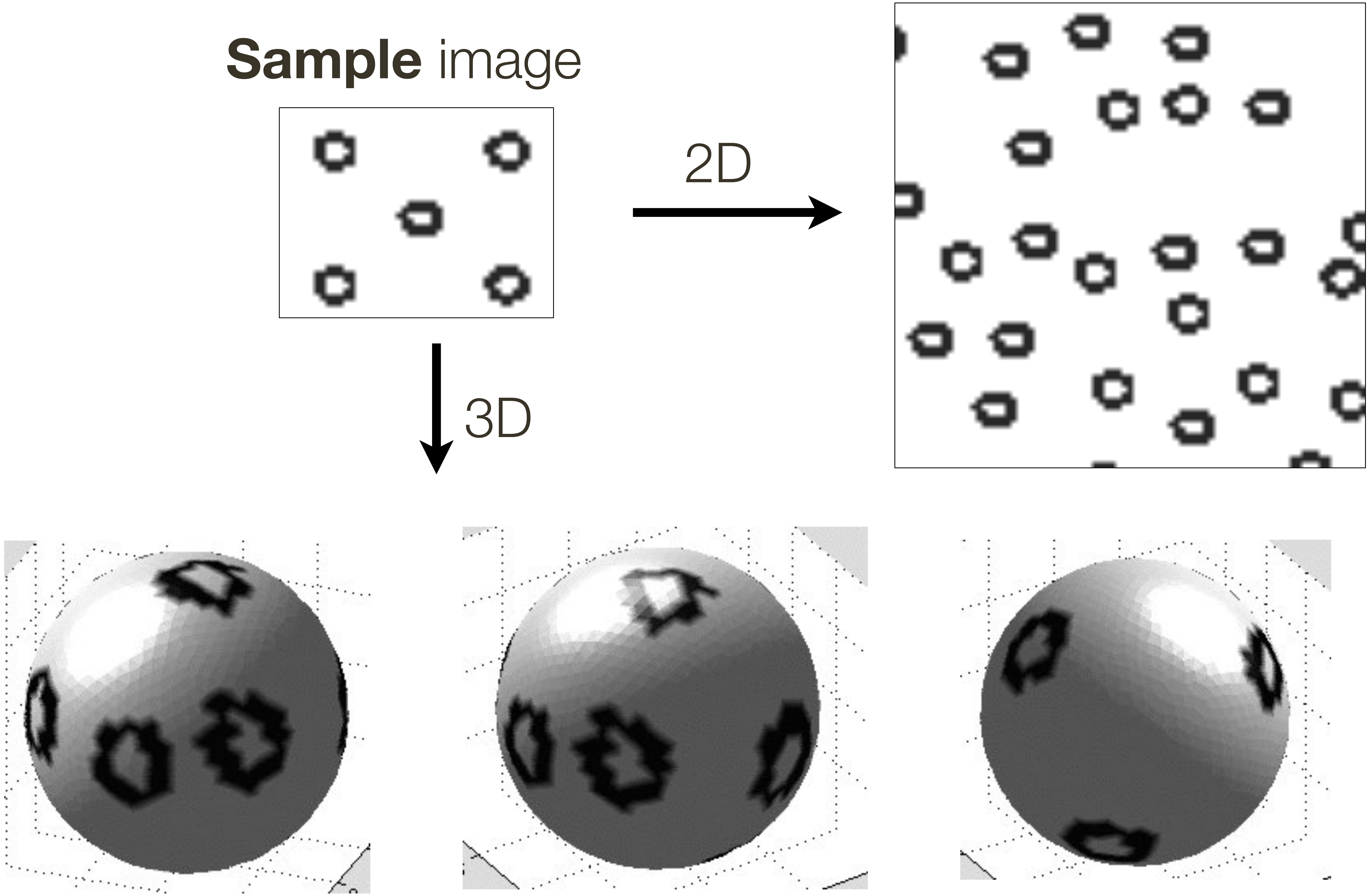
(written by Julieta Martinez, a previous CPSC 425 TA)

Randomness Parameter



Slide Credit: http://graphics.cs.cmu.edu/people/efros/research/NPS/efros-iccv99.ppt

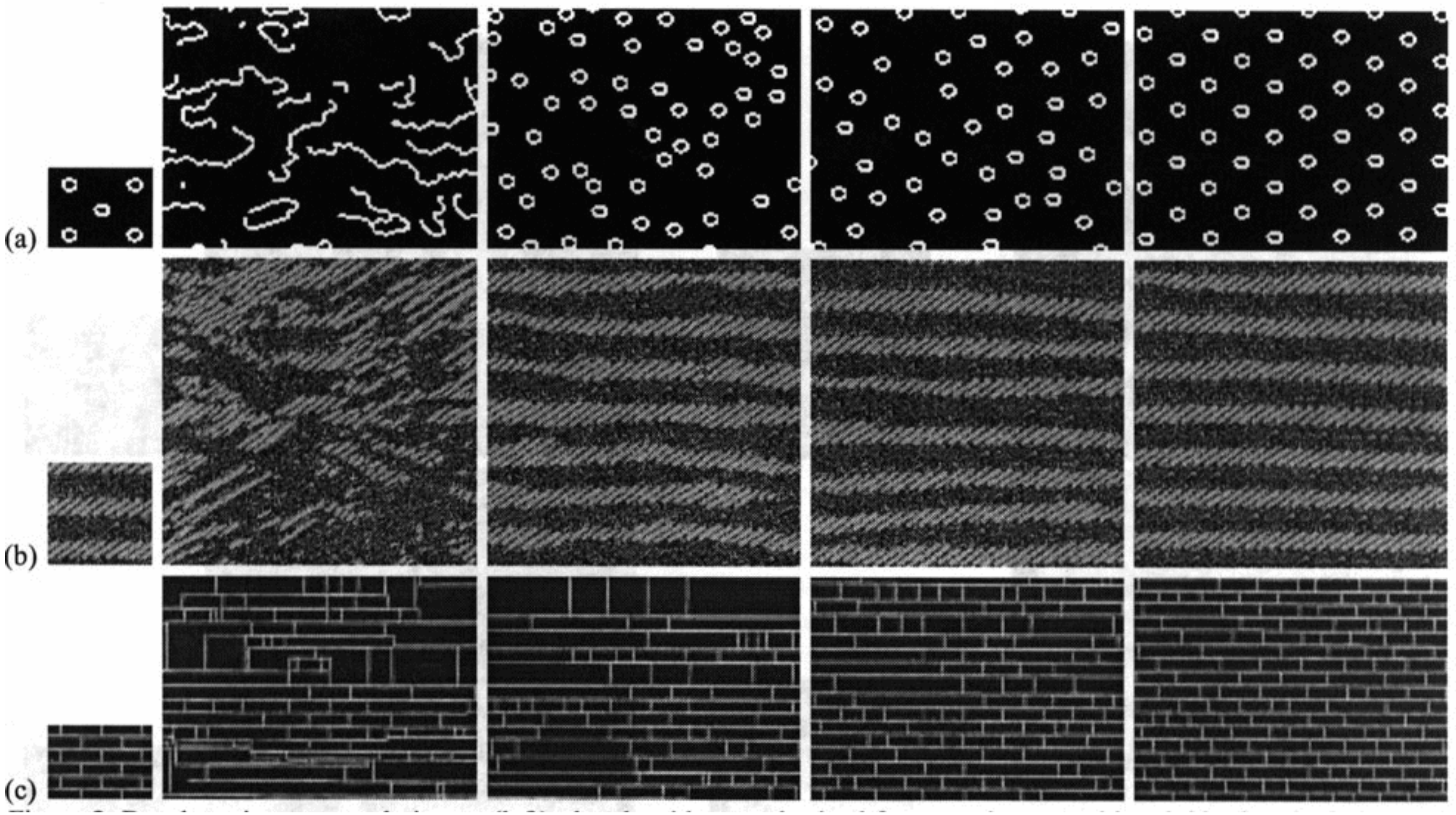
Texturing a Sphere



Slide Credit: http://graphics.cs.cmu.edu/people/efros/research/NPS/efros-iccv99.ppt

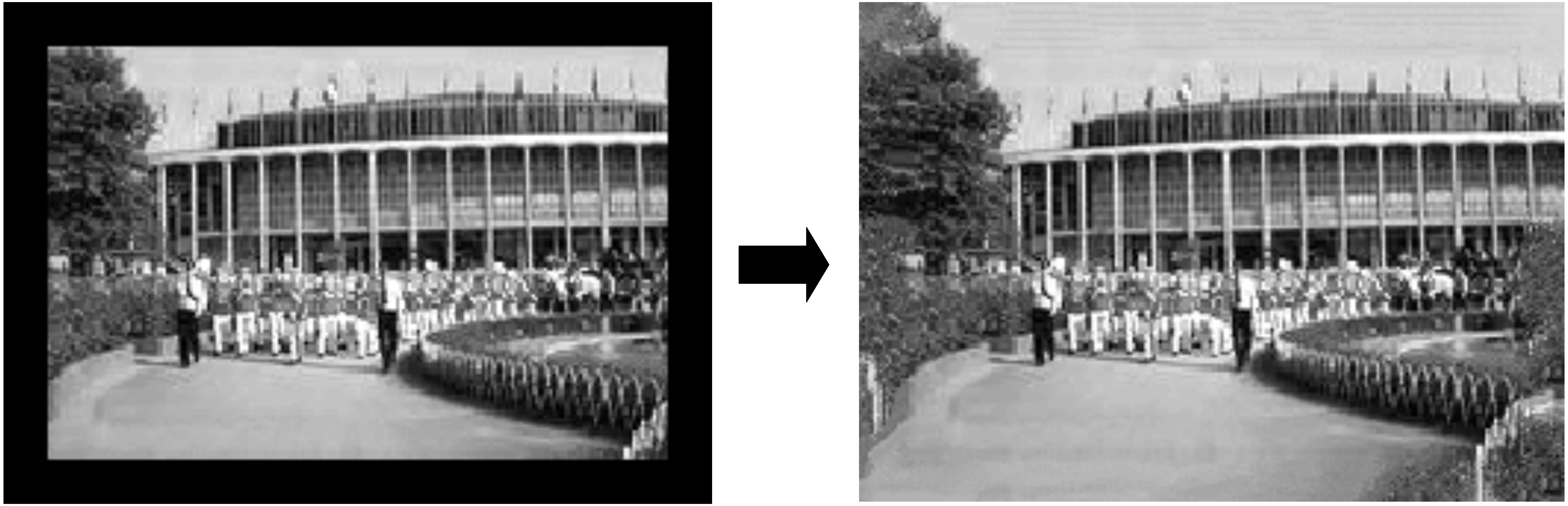
Efros and Leung: More Synthesis Results

Window Size



Forsyth & Ponce (2nd ed.) Figure 6.12

Efros and Leung: Image Extrapolation



Slide Credit: http://graphics.cs.cmu.edu/people/efros/research/NPS/efros-iccv99.ppt

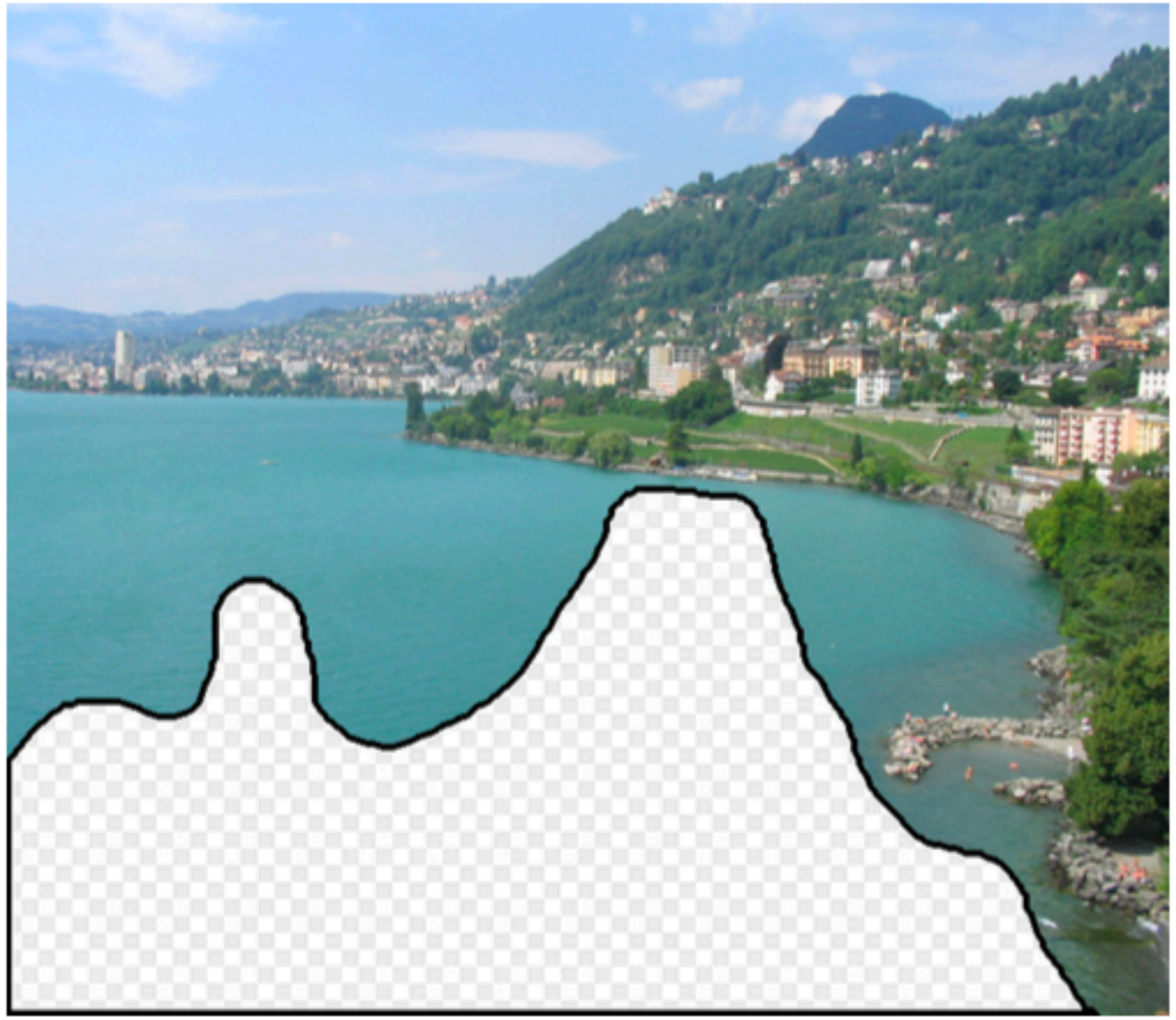
“**Big** Data” Meets Inpainting

“**Big** Data" enables surprisingly simple non-parametric, matching-based techniques to solve complex problems in computer graphics and vision.

Suppose instead of a single image, you had a massive database of a million images. What could you do?

"**Big** Data" Meets Inpainting

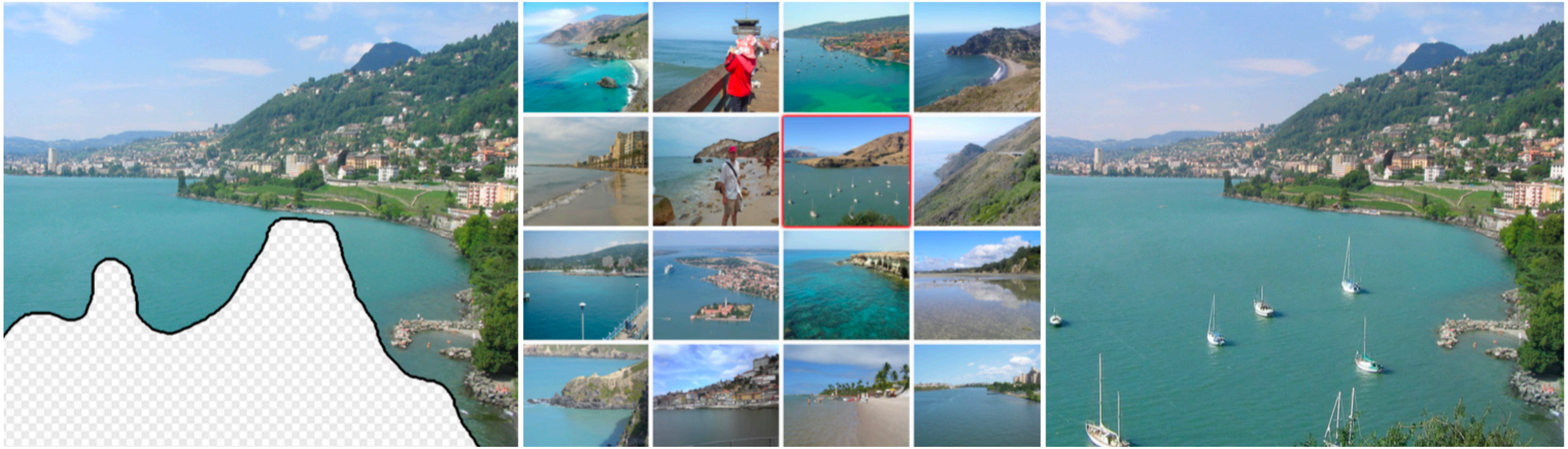
Original Image



Input

Figure Credit: Hays and Efros 2007

“**Big** Data” Meets Inpainting



Input Scene Matches Output

Figure Credit: Hays and Efros 2007

Effectiveness of “Big Data”

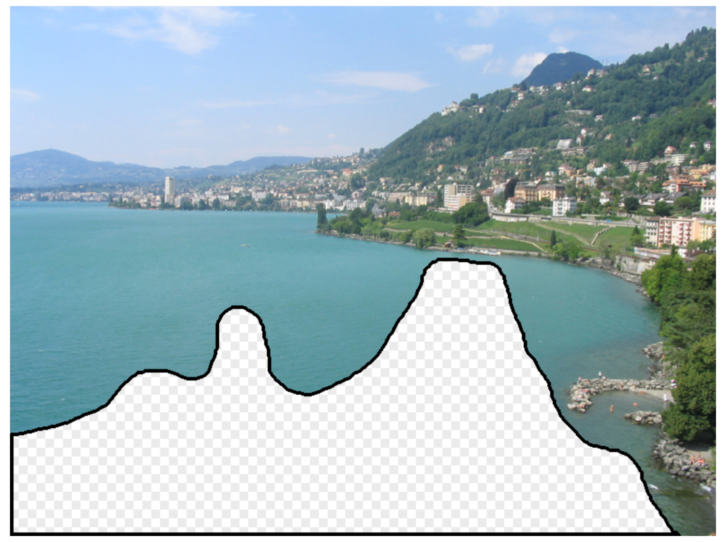
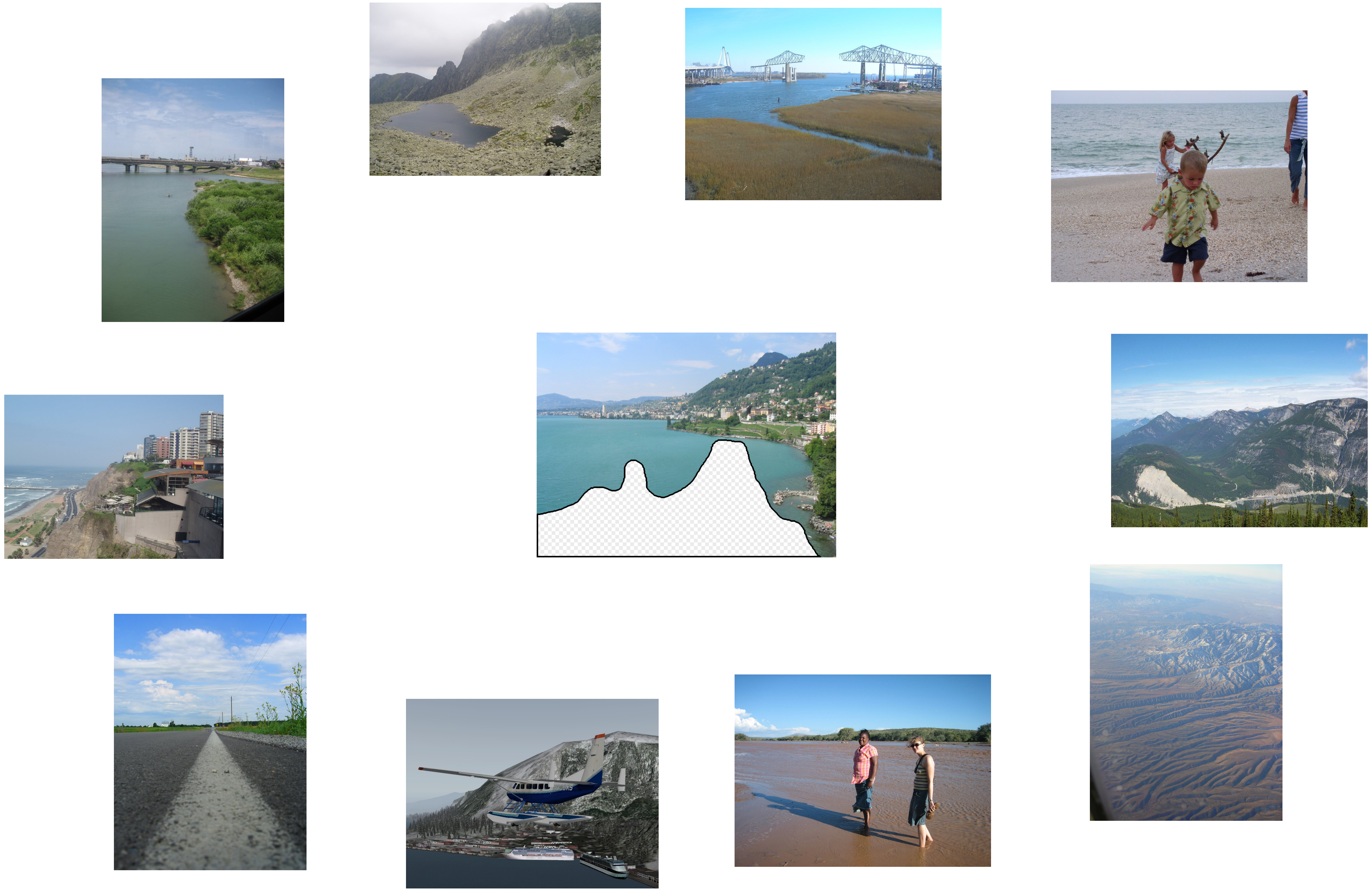


Figure Credit: Hays and Efros 2007

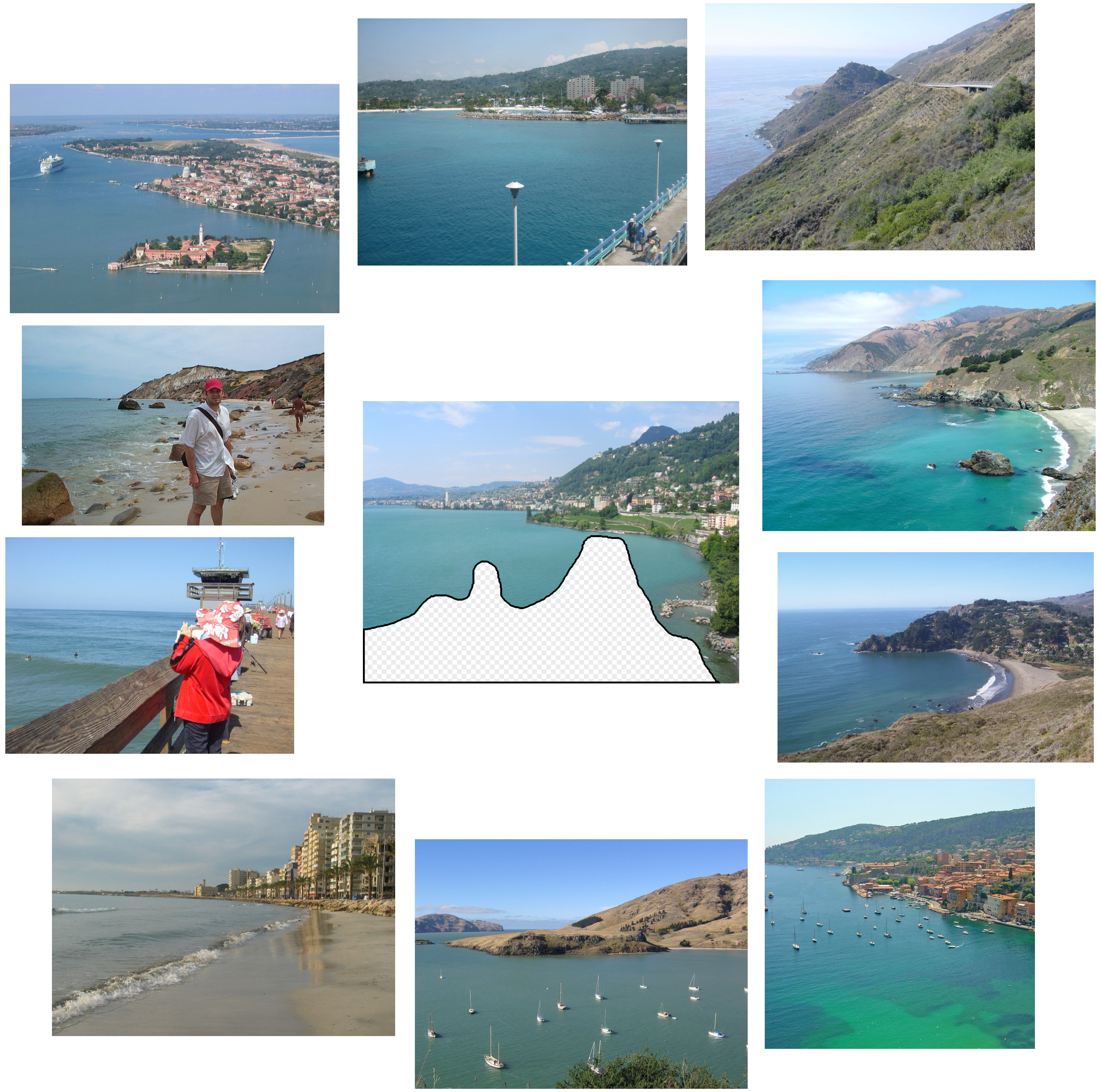
Effectiveness of "Big Data"



10 nearest neighbors from a collection of 20,000 images

Figure Credit: Hays and Efros 2007

Effectiveness of "Big Data"



10 nearest neighbors from a collection of 2 million images

Figure Credit: Hays and Efros 2007

“**Big** Data” Meets Inpainting

Figure Credit: Hays and Efros 2007

“**Big** Data” Meets Inpainting

Algorithm sketch (Hays and Efros 2007):

1. Create a short list of a few hundred “best matching" images based on global image statistics

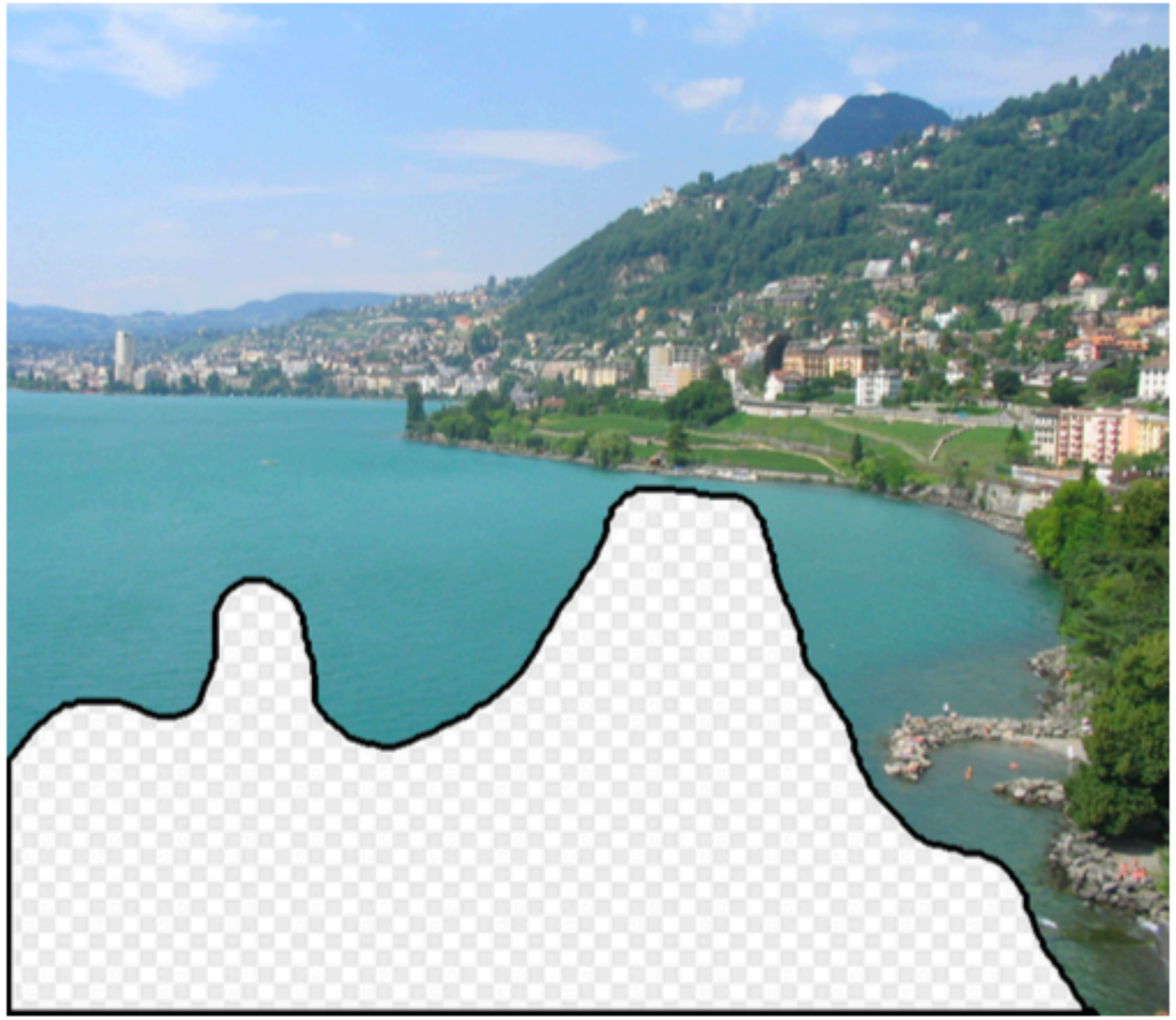
2. Find patches in the short list that match the context surrounding the image region we want to fill

3. Blend the match into the original image

Purely **data-driven**, requires no manual labeling of images

"**Big** Data" Meets Inpainting

Original Image



Input

Figure Credit: Hays and Efros 2007

“**Big** Data” Meets Inpainting

Figure Credit: Hays and Efros 2007

“**Big** Data” Meets Inpainting

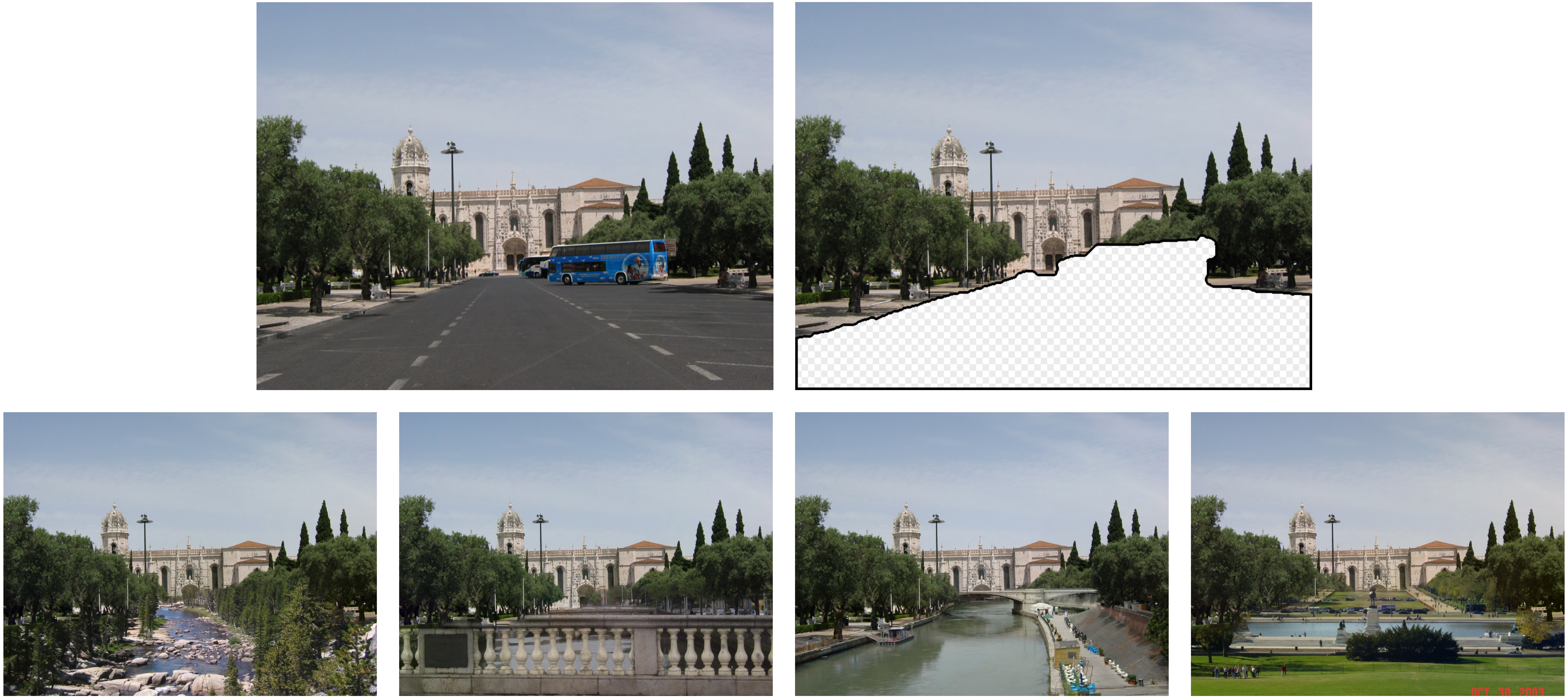


Figure Credit: Hays and Efros 2007