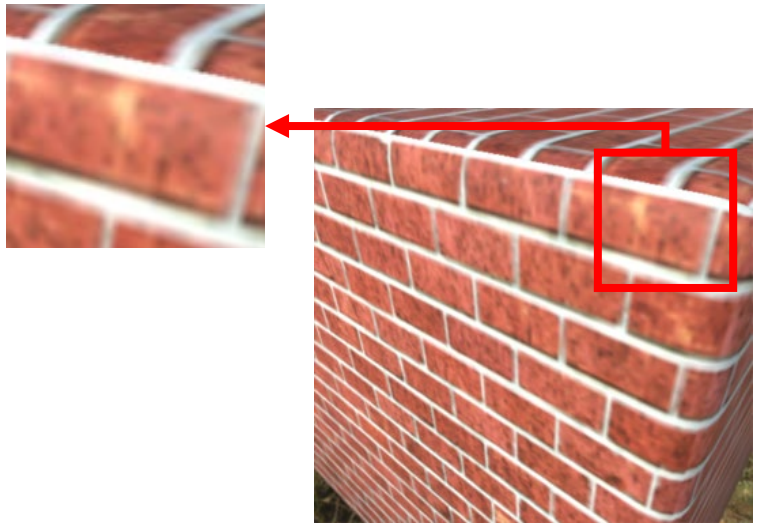
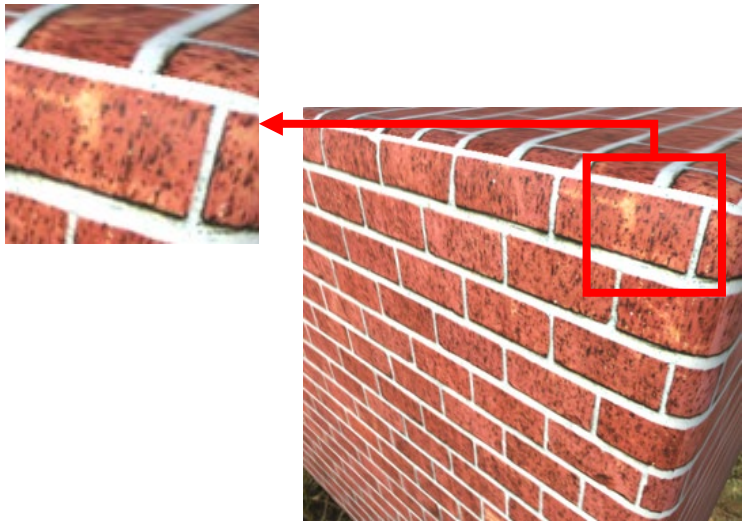
Realtime Texture Upsampling on Graphics Hardware Using Fractal Coding

Yuto Kominami　Tomohiko Mukai
Tokyo Metropolitan University

Background

- The use of high-resolution textures requires memory capacity and bandwidth
- Bilinear interpolation does not produce high-frequency components





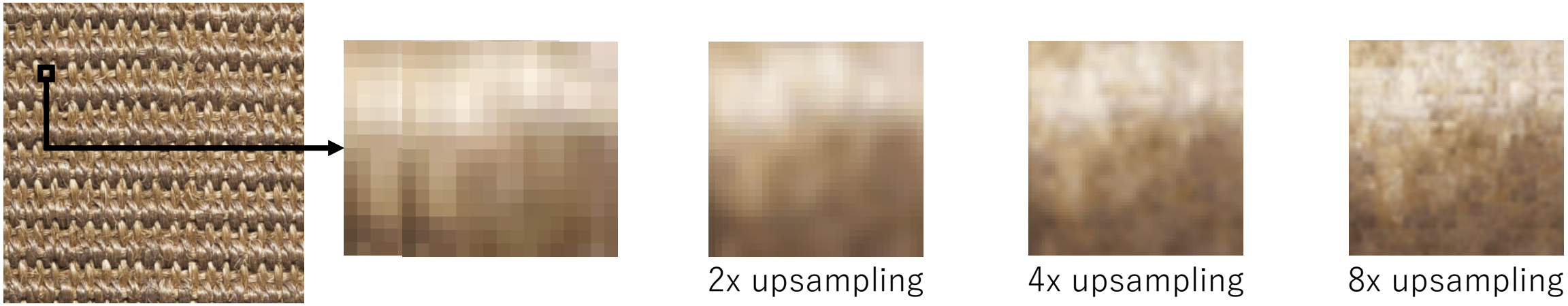
Related Work

- Example-based super-resolution （2002）
- Super-resolution from a single image （2009）
- Fractal-based hierarchial mip-pyramid texture compression （2006）
- Learning a deep convolutional network for image super-resolution （2014）

Not intended to be used in realtime 3DCG

Contribution

- GPU-friendly realtime method
- Using Fractal Compression to synthesize a higher resolution image
- Iterative algorithm to refine the fractal code



source

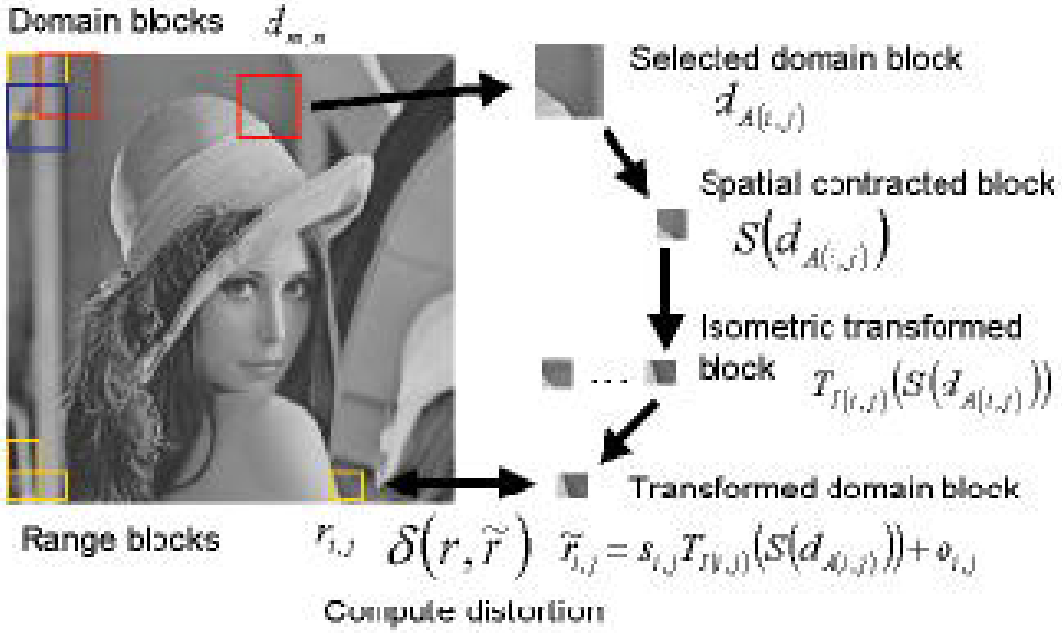
2x upsampling

4x upsampling

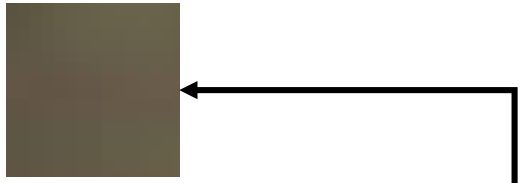
8x upsampling

Fractal Compression [M. Barnsley et al. 1987]

- Transform coding that uses the self-similarity of an image to generate an approximate image of itself
- Known for its high compression ratio and fast decoding speed

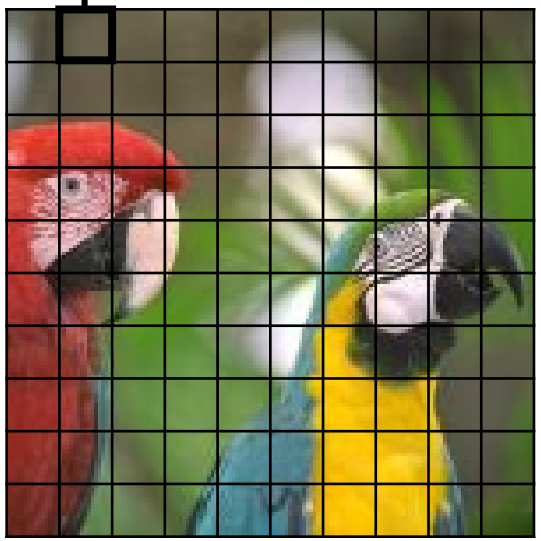


Principle of Fractal Compression



Domain

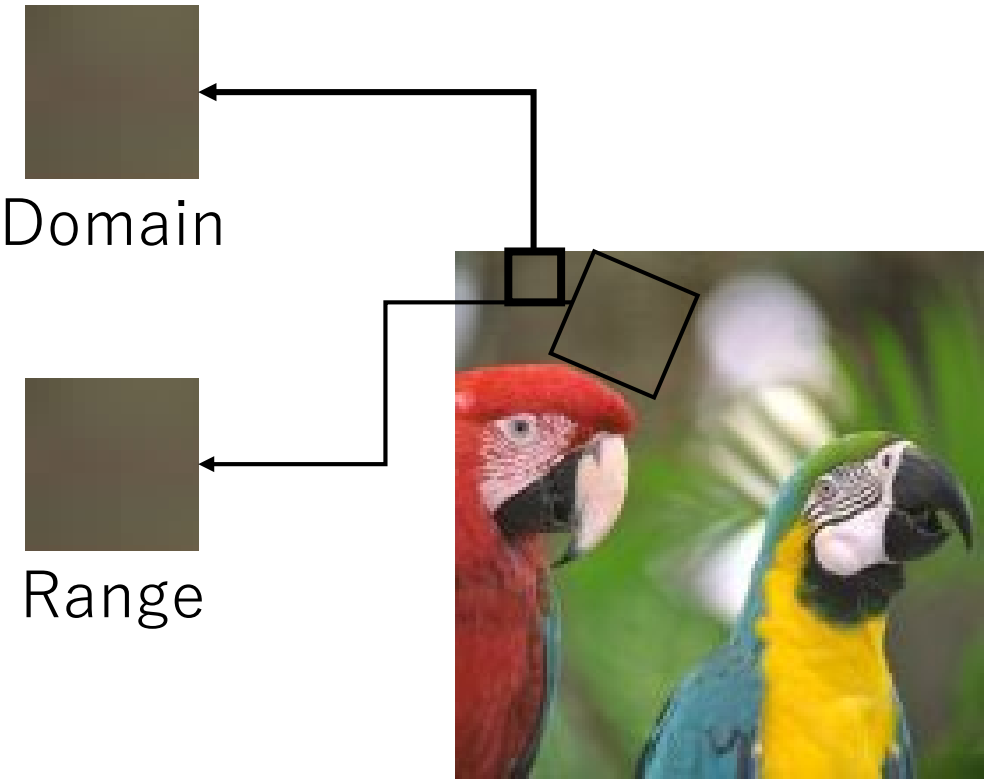
Source image



Divide the entire image into multiple blocks

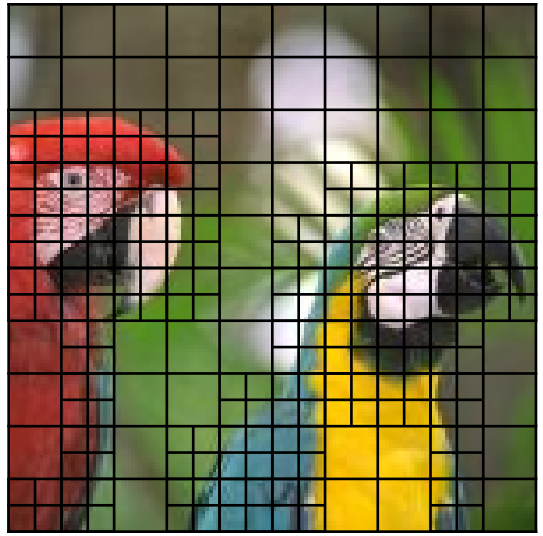
Principle of Fractal Compression

Source Image



Principle of Fractal Compression

Source Image

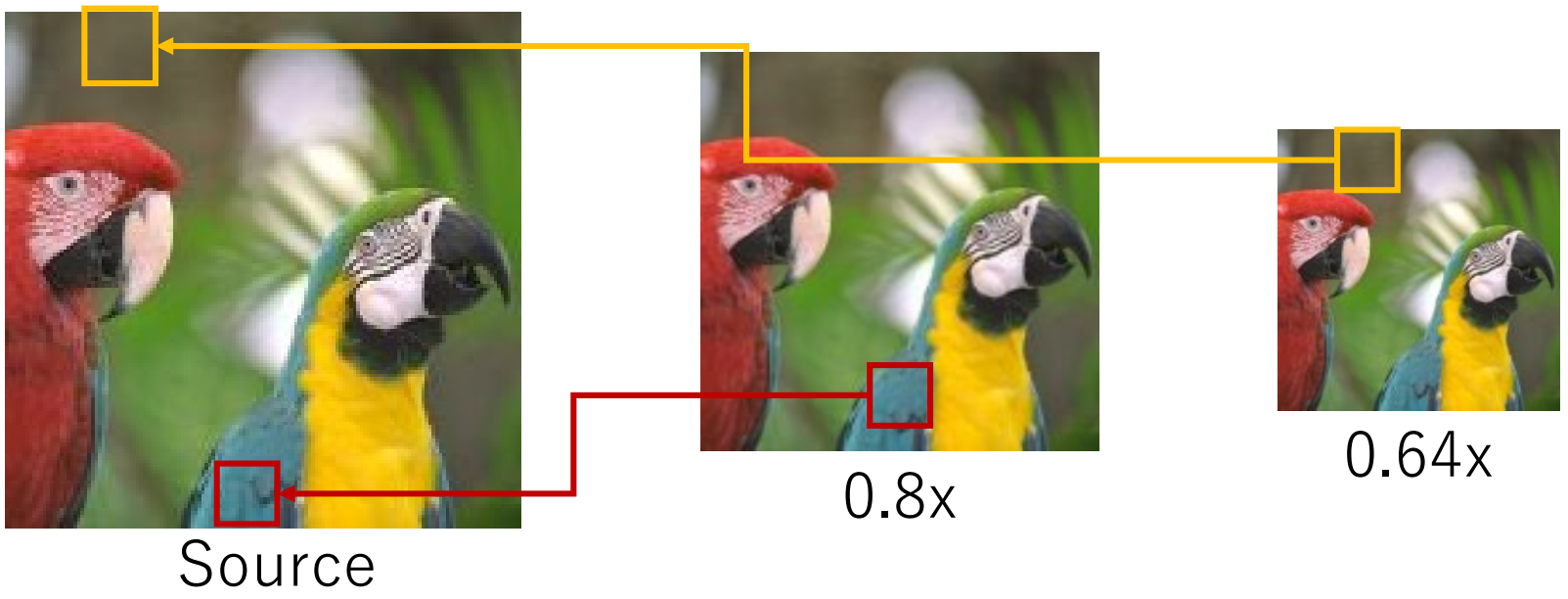


Make more division

Closely-Related Work:

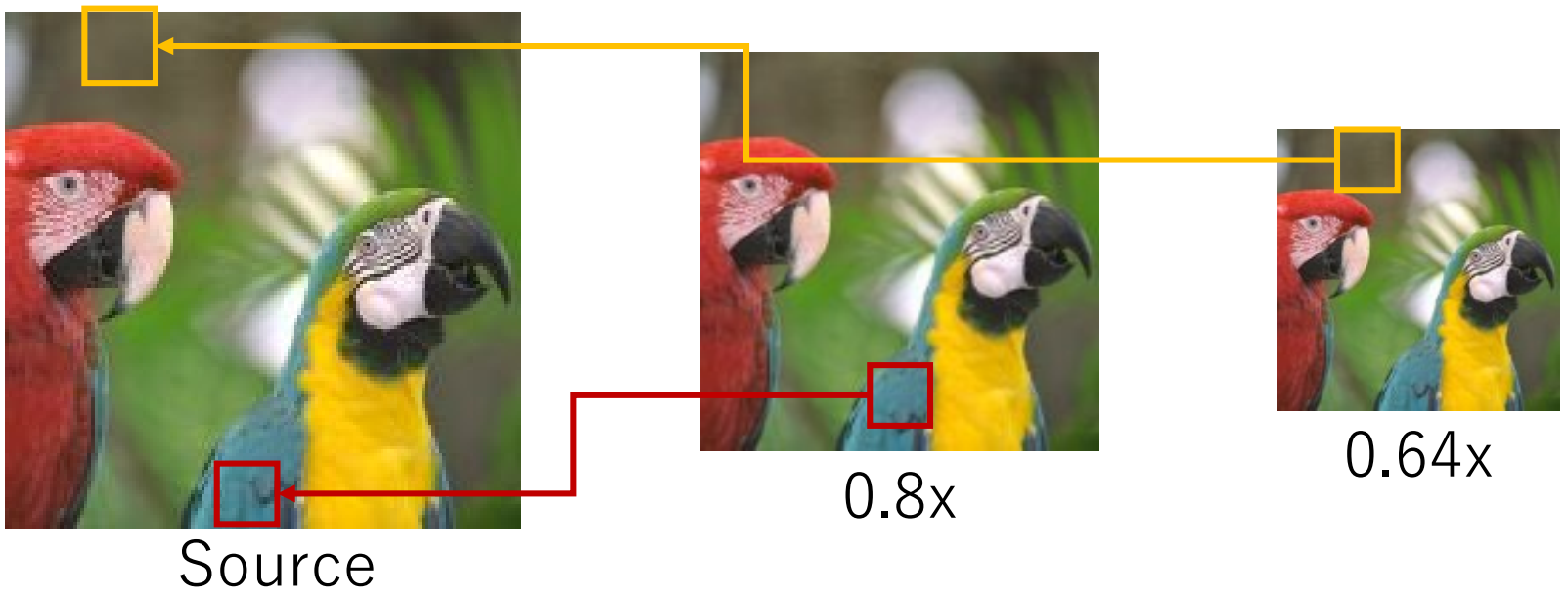
Fractal-based hierarchical mip-pyramid texture compression [Stachera 2006]

Super-resolution from a Single Image [Glasner 2009]



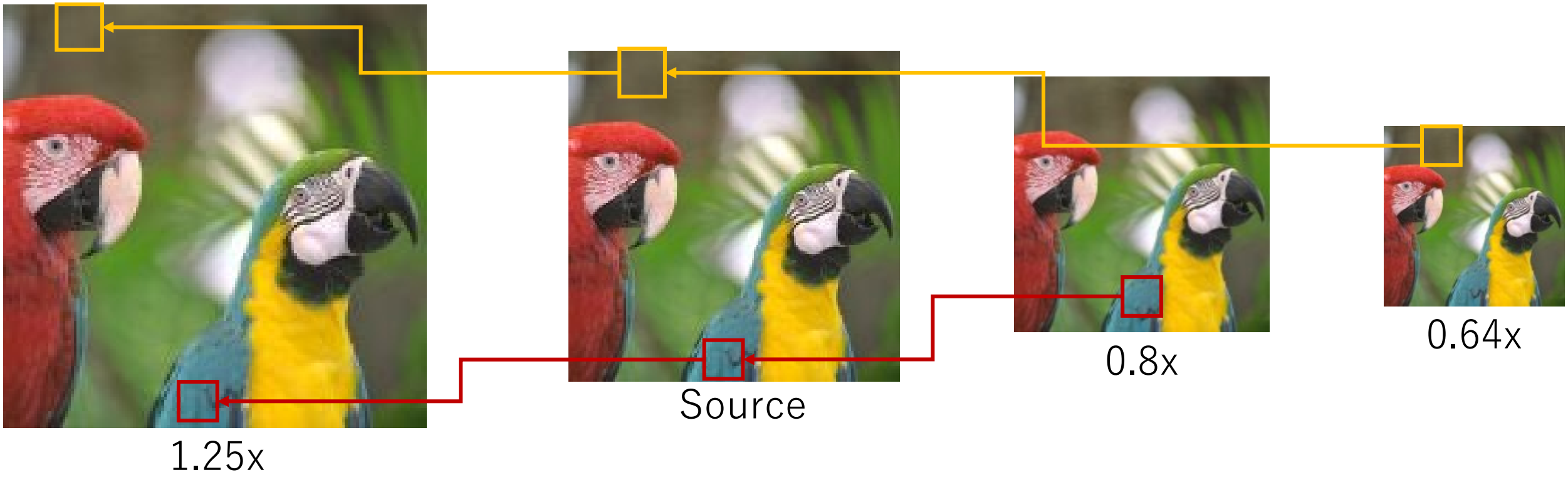
- It uses only one image to predict high resolution
- Predict high-res image by finding corresponding area

Super-resolution from a Single Image [Glasner 2009]



These correspondences are used as hints for magnification.

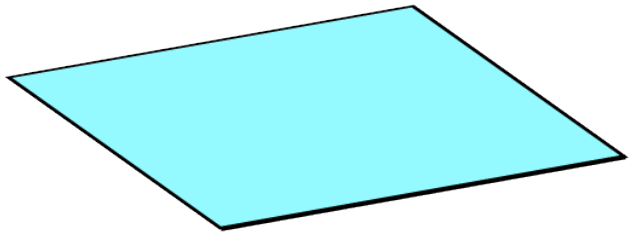
Super-resolution from a Single Image [Glasner 2009]



Algorithm

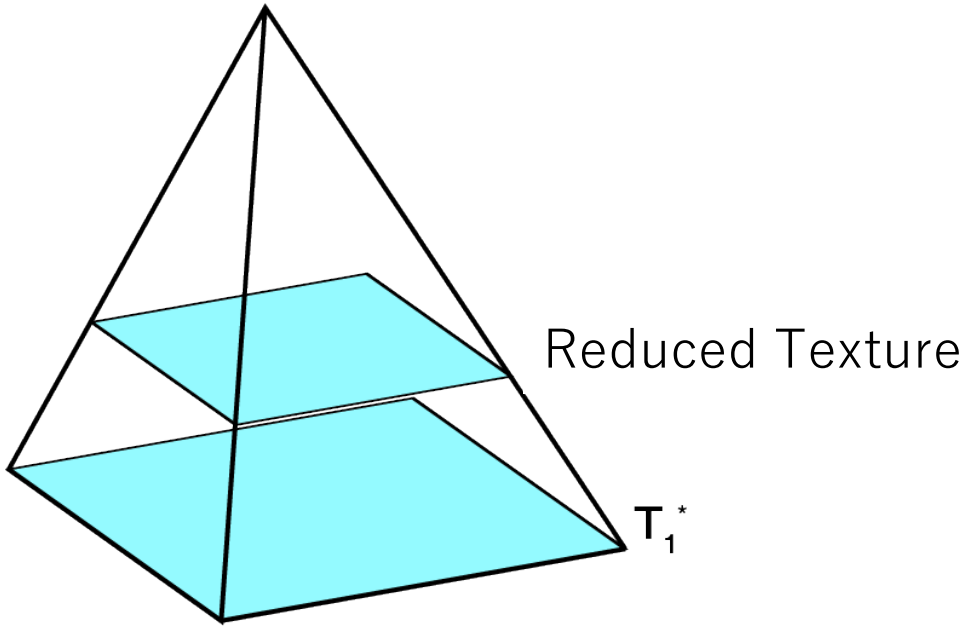
- Finding correspondence using mipmap textures
- Simple procedure and structure to handle on GPU
- Iterative refinement

Algorithm



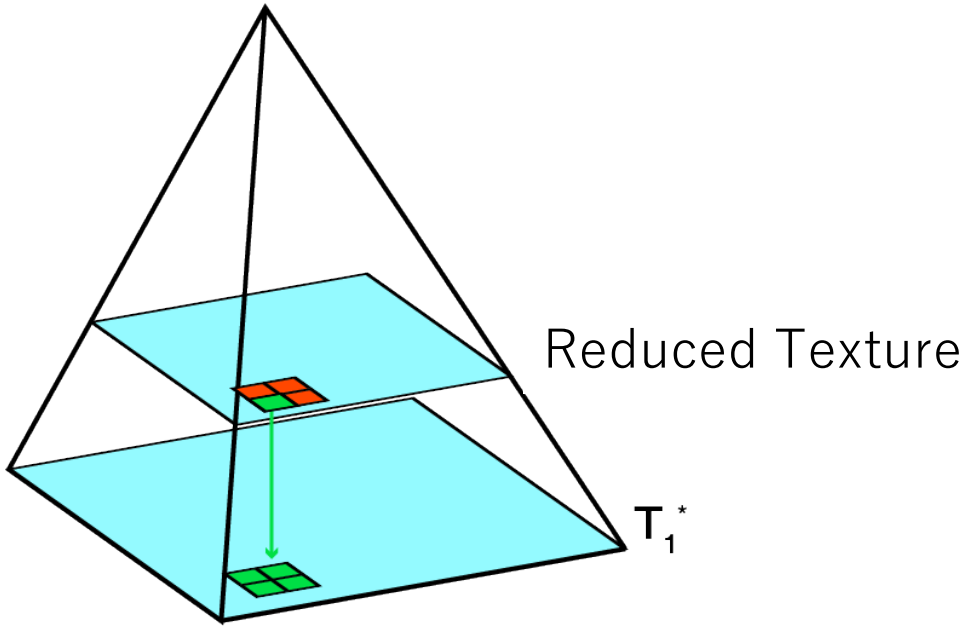
Algorithm

Create reduced texture from source texture



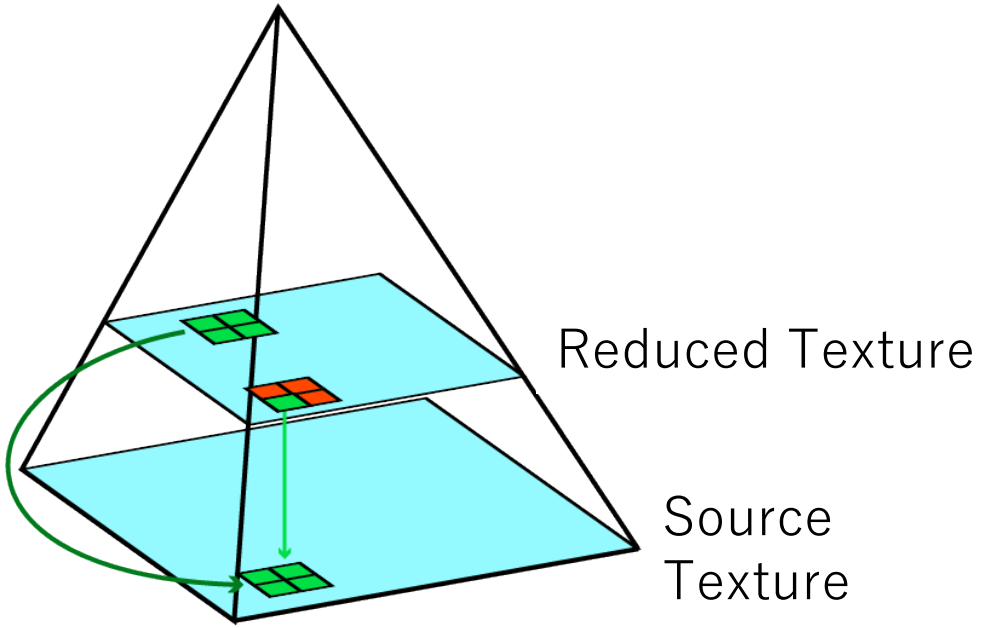
Algorithm

Sampling 2x2 texel block from source texture



Algorithm

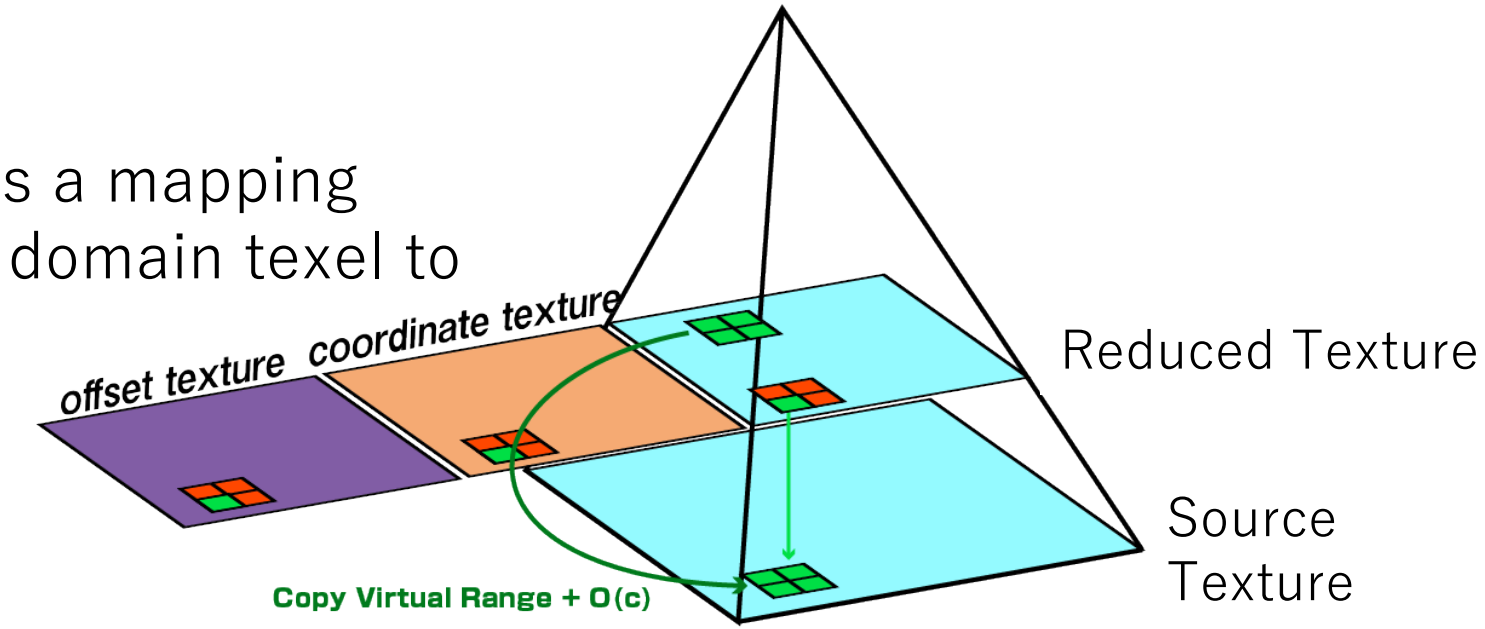
Search virtual range that can be substituted for source 2x2 texel block



Algorithm

Coordinate texture

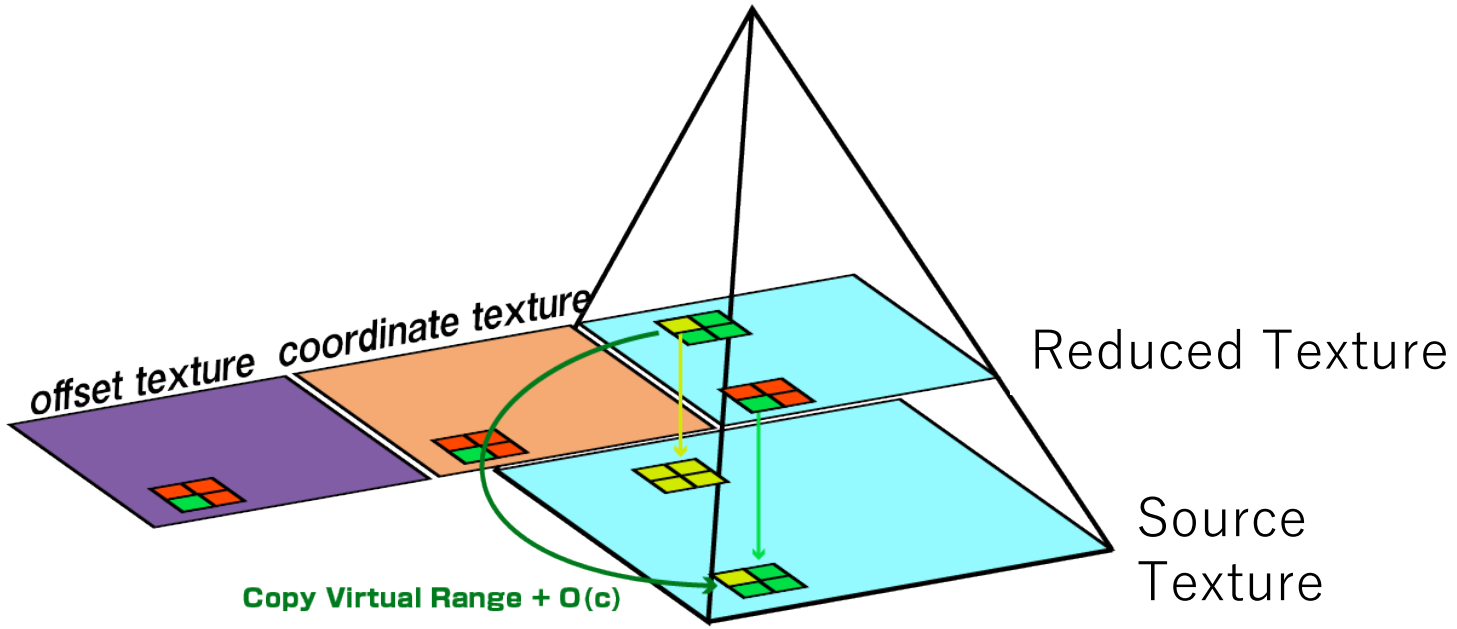
- Store value that represents a mapping from 2D coordinate of the domain texel to the virtual range



Offset texture

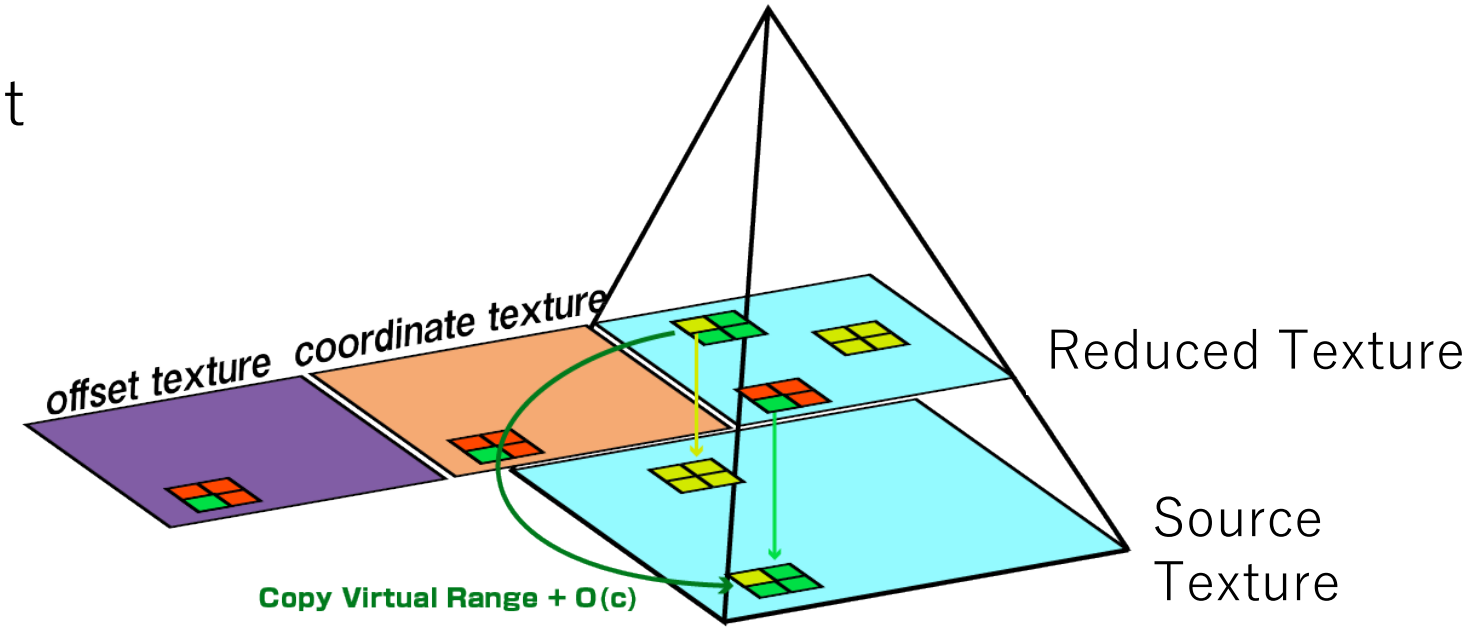
- Store value that represents mean texel error between the range and virtual range

Algorithm


offset texture
coordinate texture
Reduced Texture
Source Texture
Copy Virtual Range + O(c)

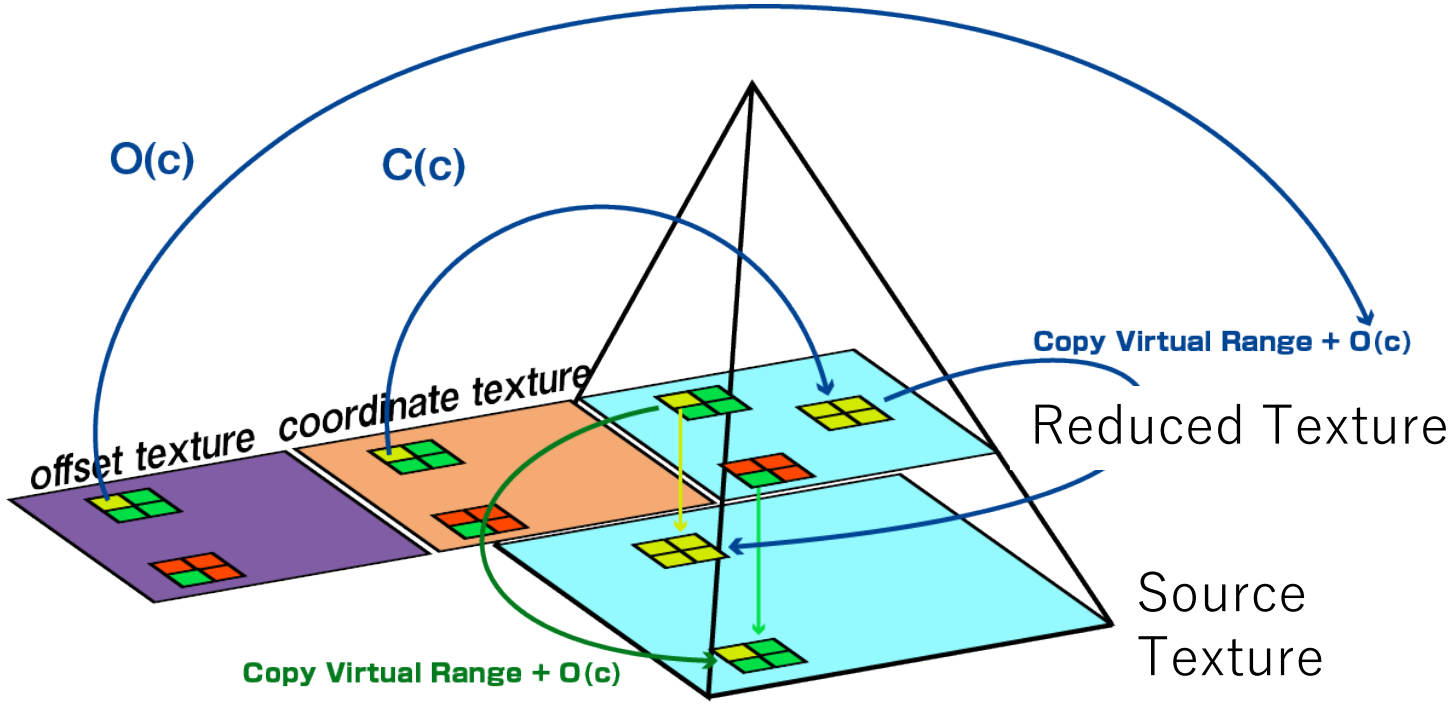
Algorithm

Likewise, there is a block that approximates a yellow block



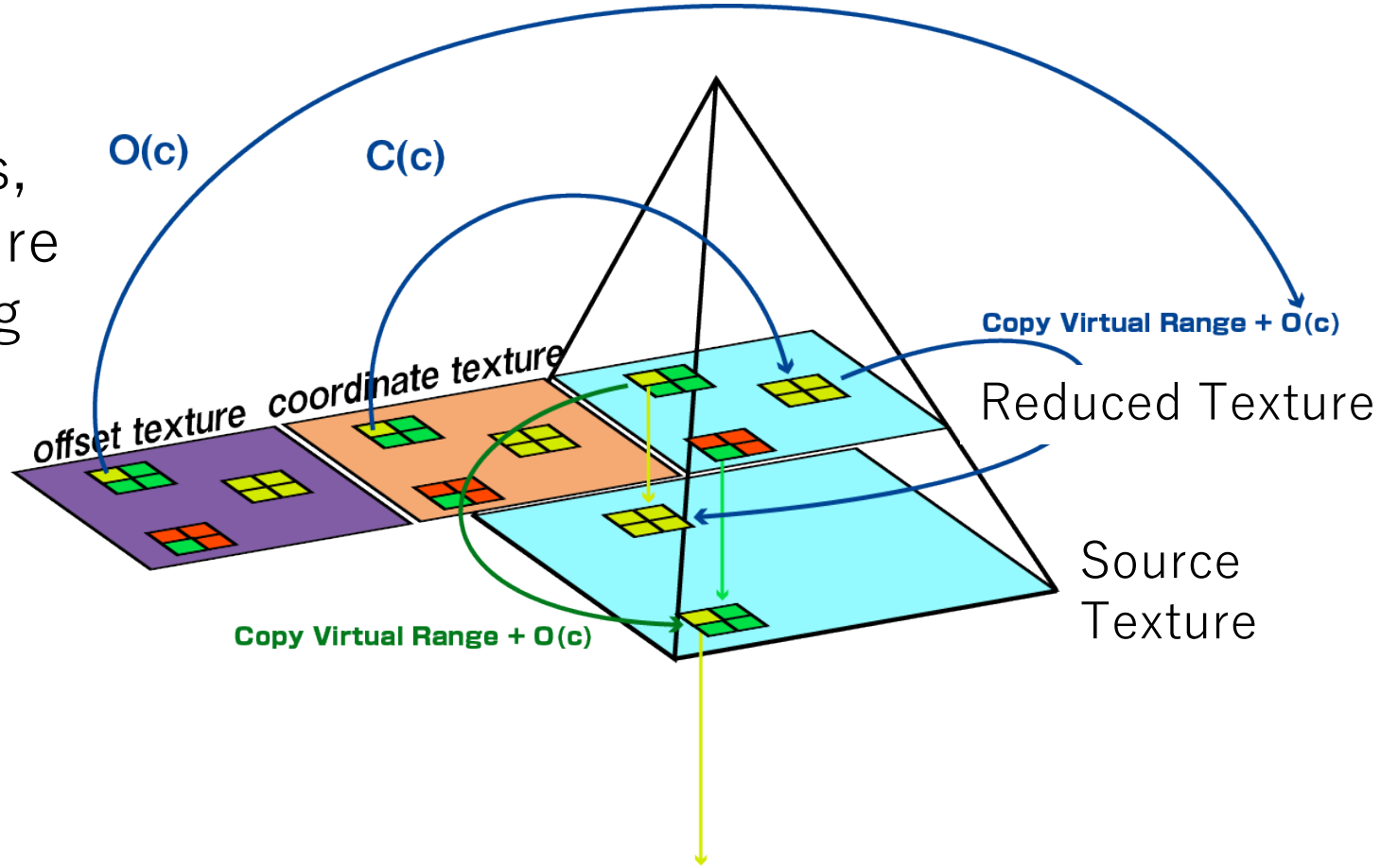
Algorithm

Variables are stored in the fractal codes.



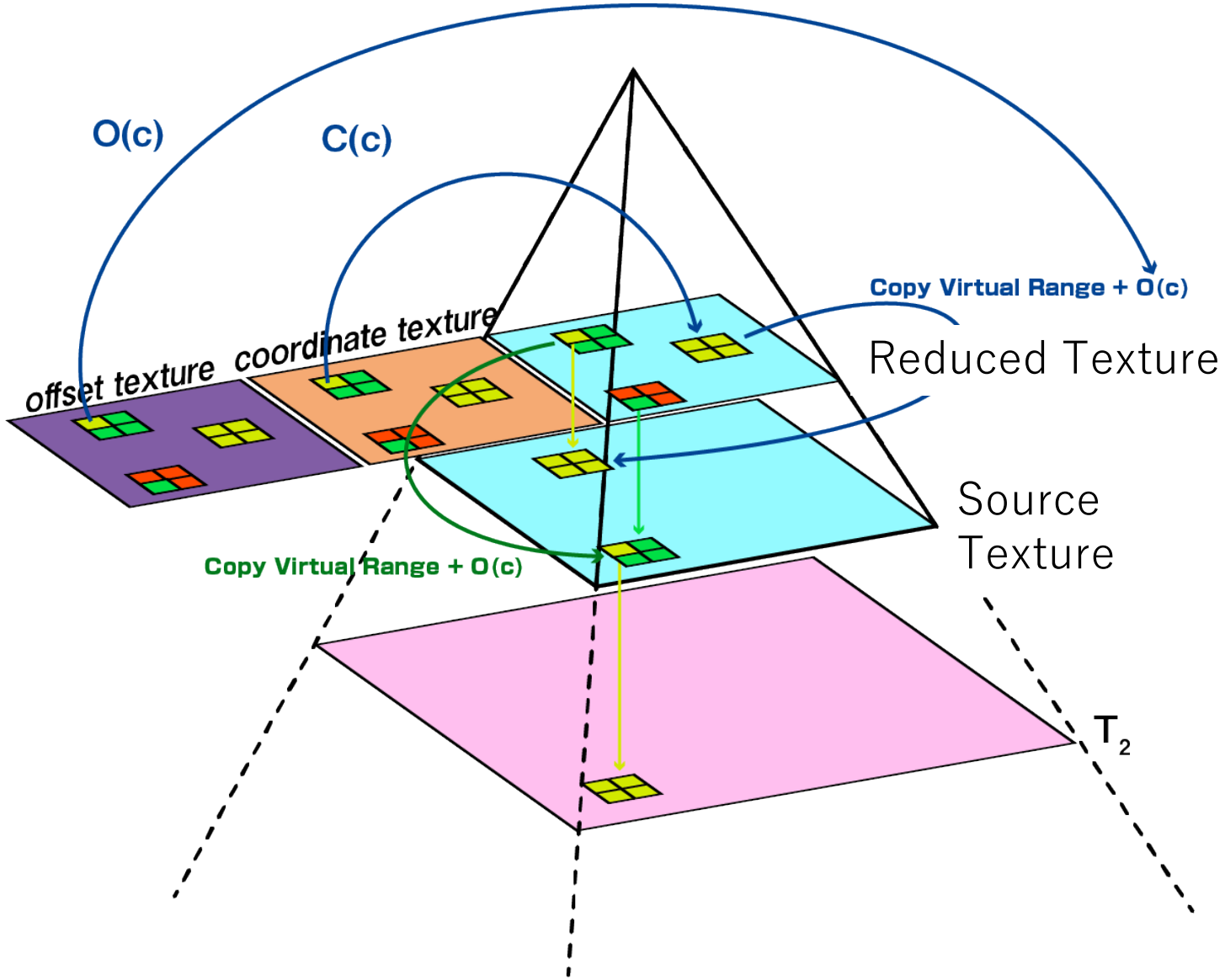
Algorithm

By applying the same process, all texels in the reduced texture are assigned to corresponding texel-block in the same reduced texture



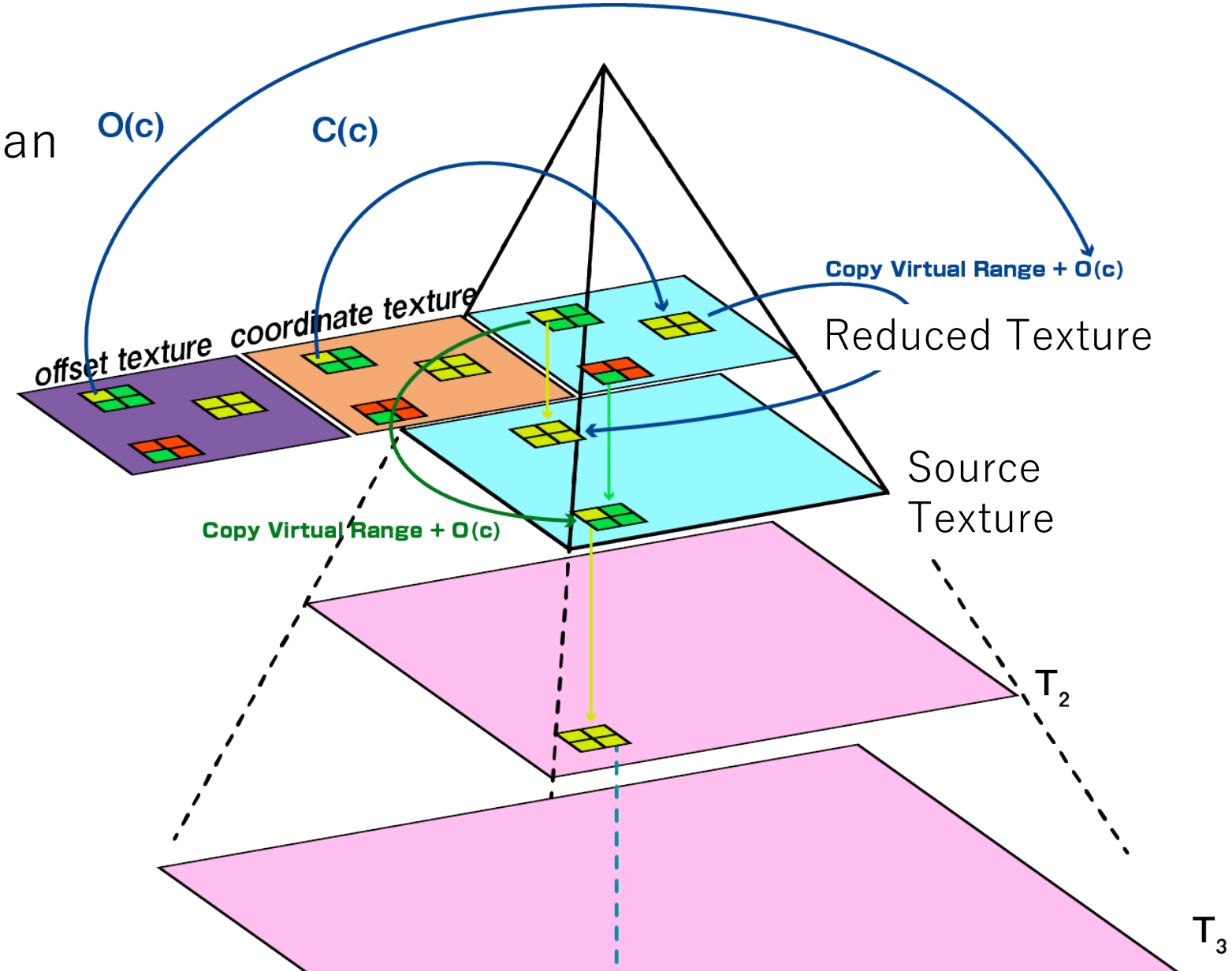
Algorithm

We can get
2x upsampled texture



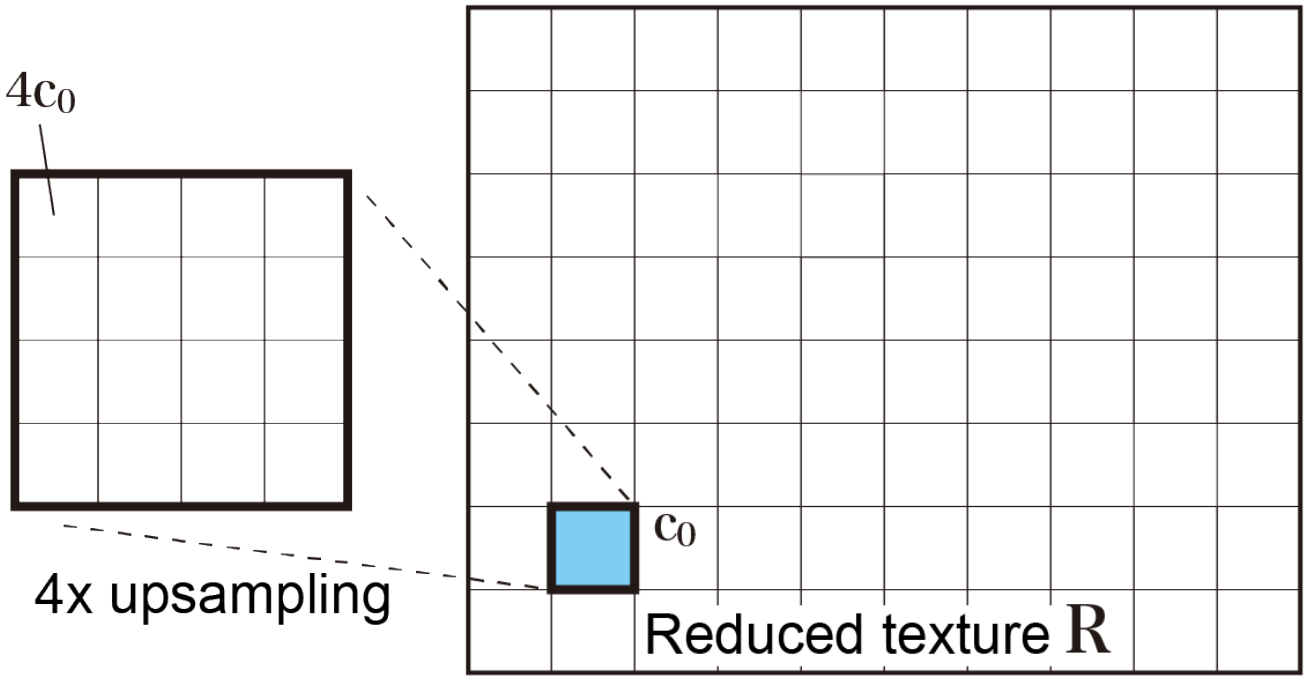
Algorithm

Furthermore, this algorithm can be computed recursively



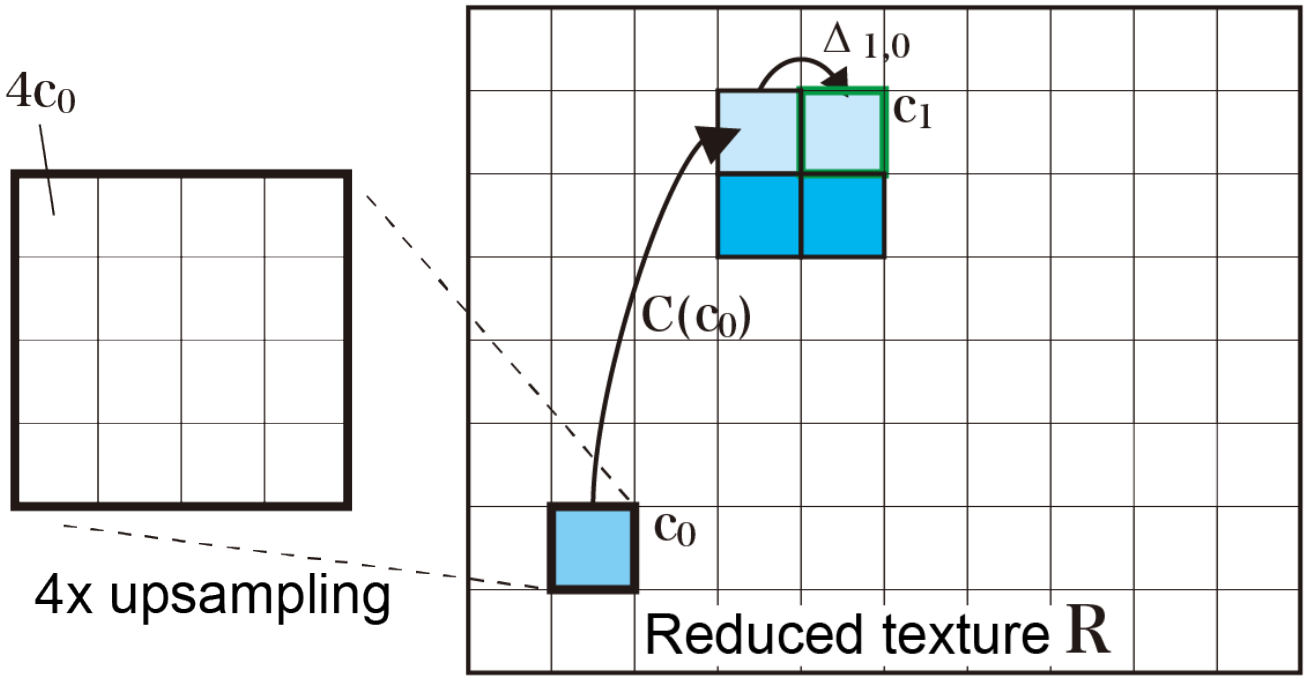
Example: 4x upsampling

Decoding is executed by reading fractal codes



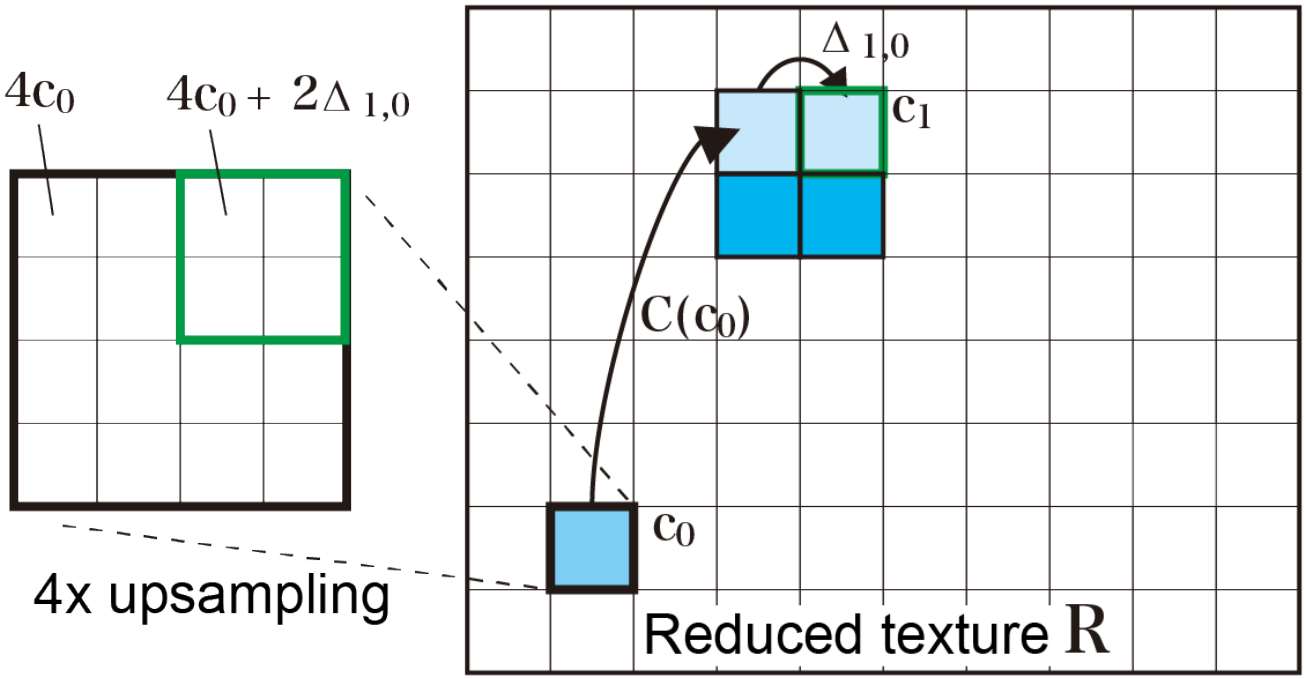
Example: 4x upsampling

- Read the coordinate from coordinate texture ($C(C0)$)
- Fetch texture color of $C(C0)$



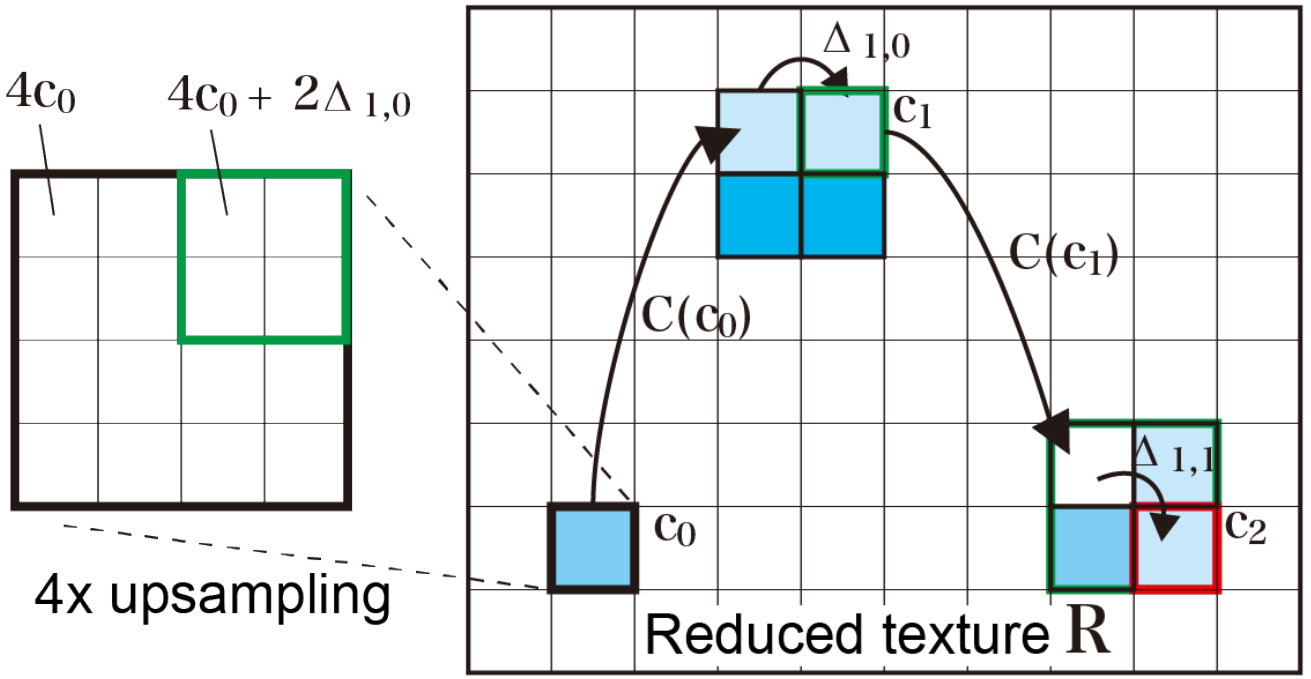
Example: 4x upsampling

- The areas highlighted in green correspond to each other
- Read the color from this coordinate of offset texture
- Offset is used to correct the color of the upsampling texture



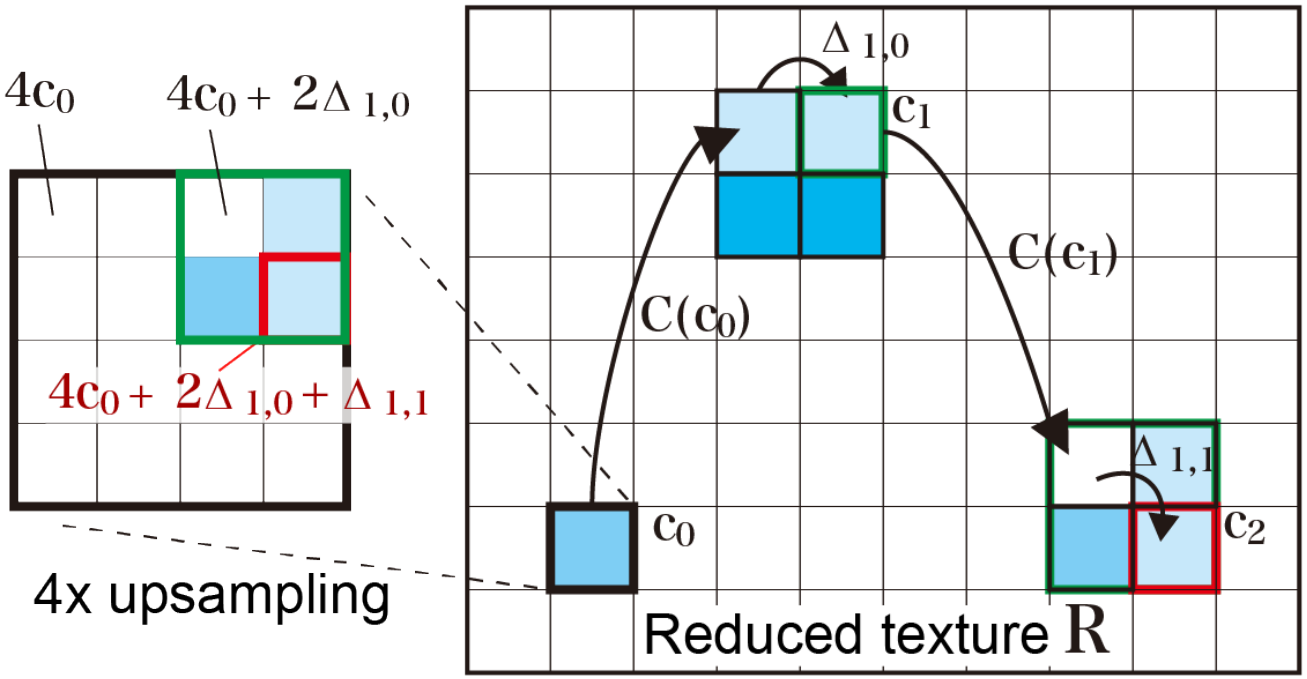
Example: 4x upsampling

- The coordinates are also recorded for **c1.**
- fetch texture color of C(c1).

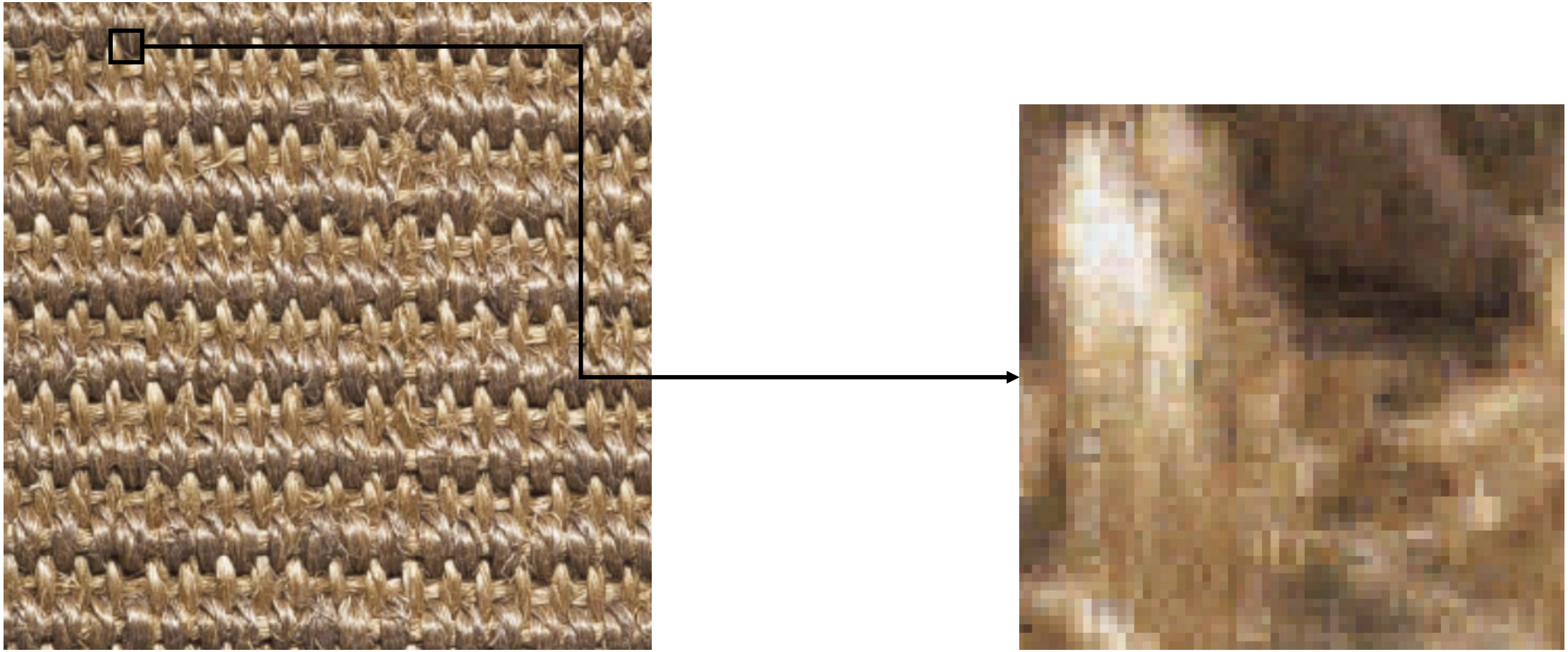


Example: 4x upsampling

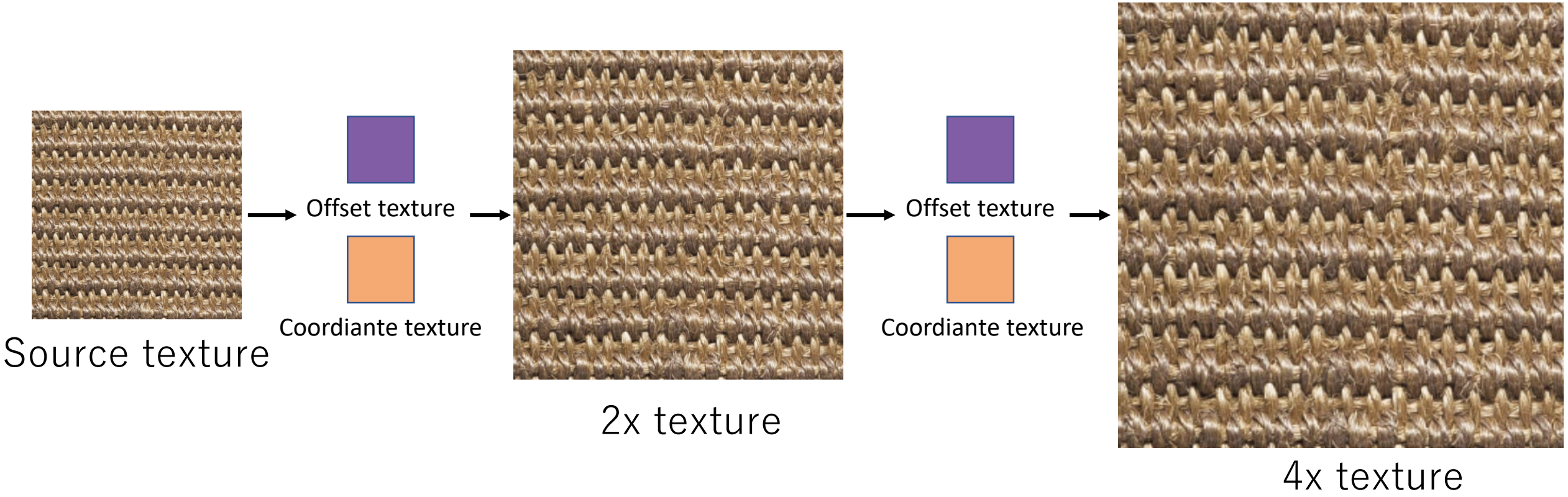
- The areas highlighted in red correspond to each other
- Resolution is increased without using any source texture



Problem of block-wise upsampling



Iterative refinement

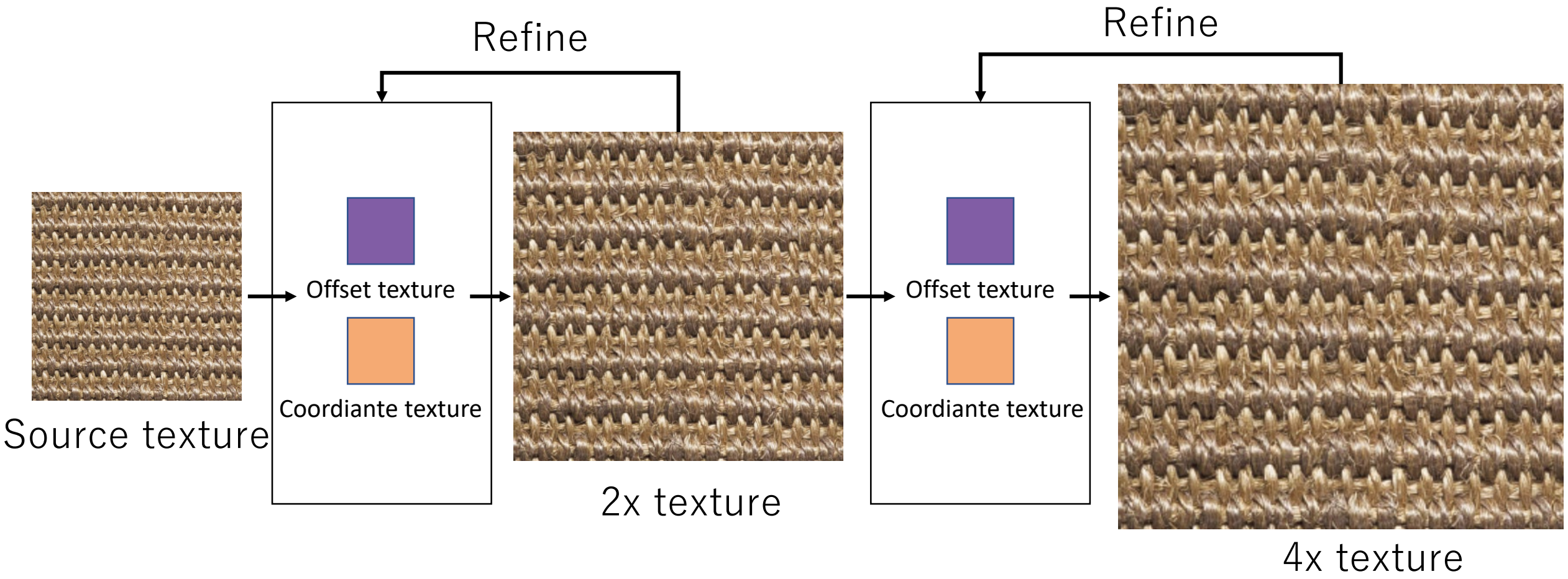


Source texture

2x texture

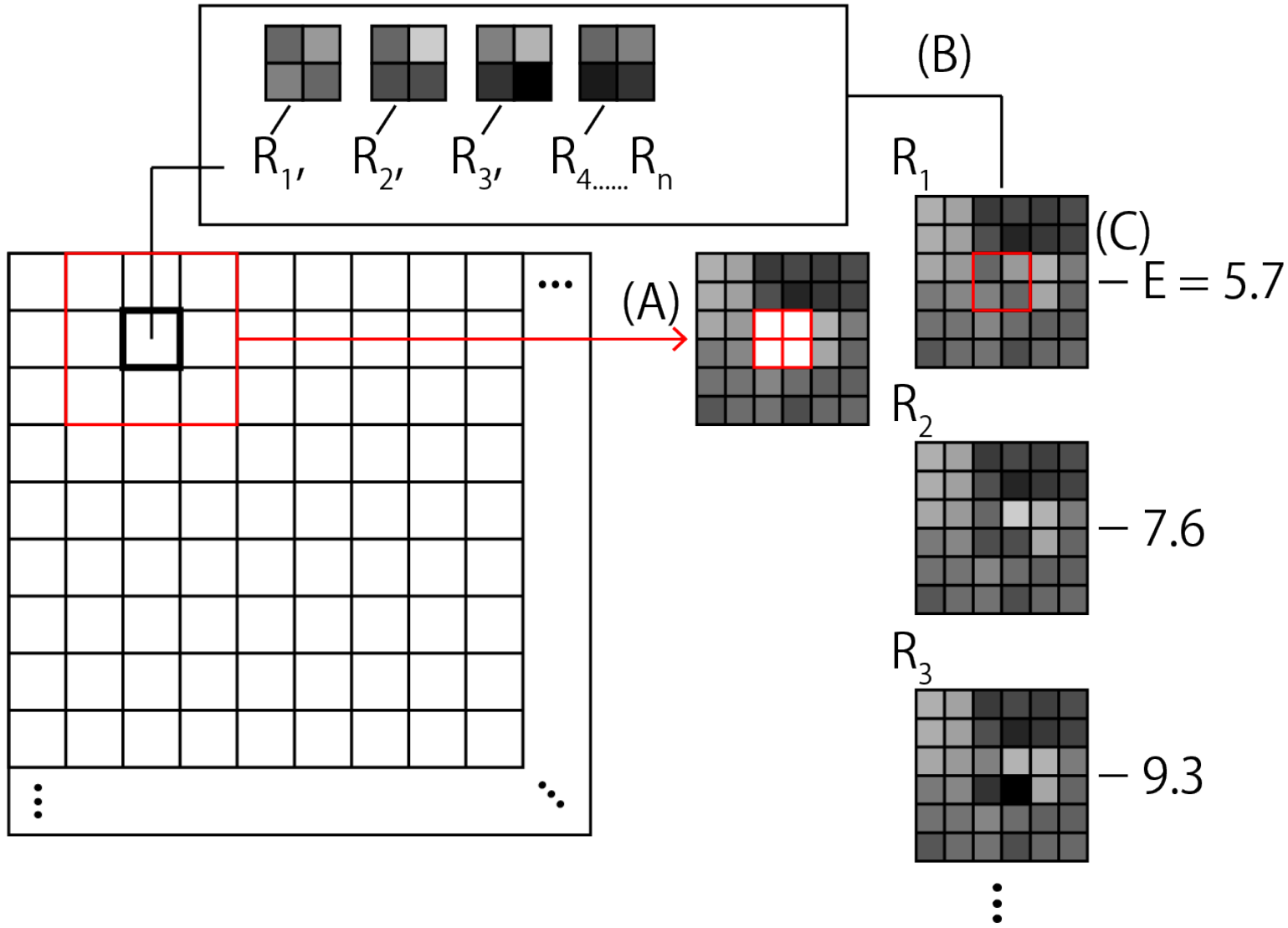
4x texture

Iterative refinement



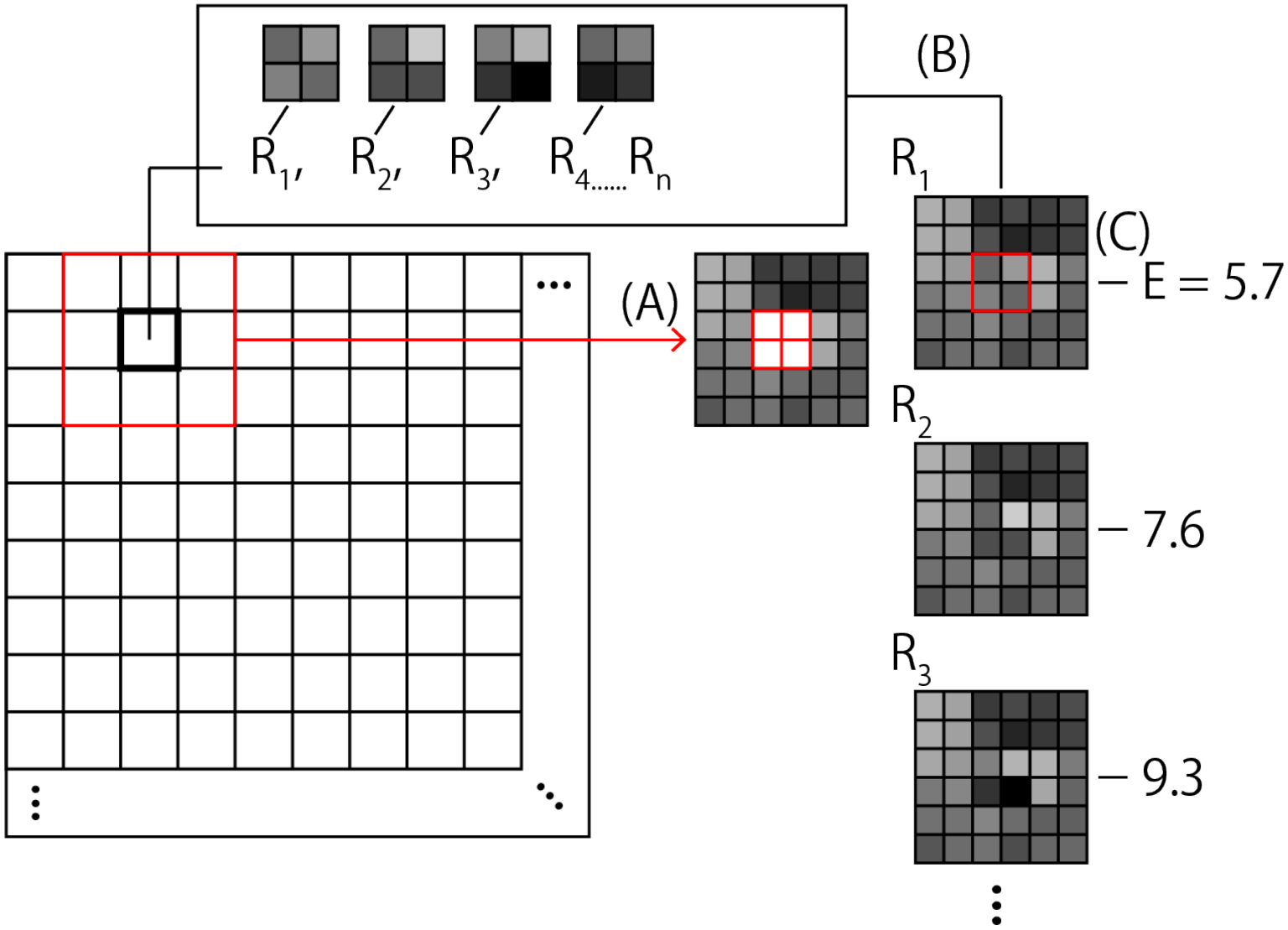
Laplacian minimization

The objective function evaluates the pixels predicted by the neighborhood, which is minimized when the target pixel is well-smoothed relative to the neighborhood.



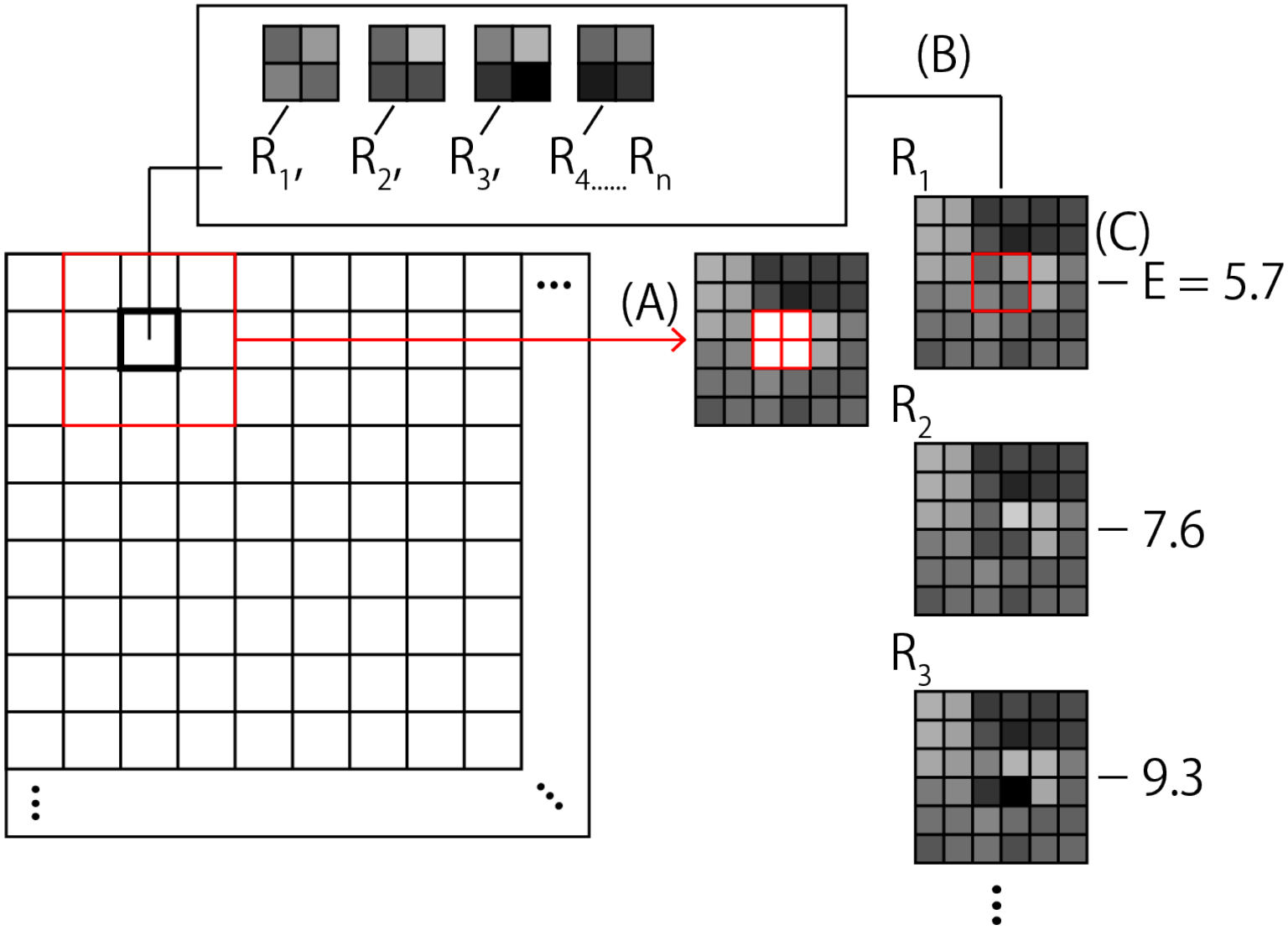
Laplacian minimization

(A) Synthesizes the neighboring pixels.



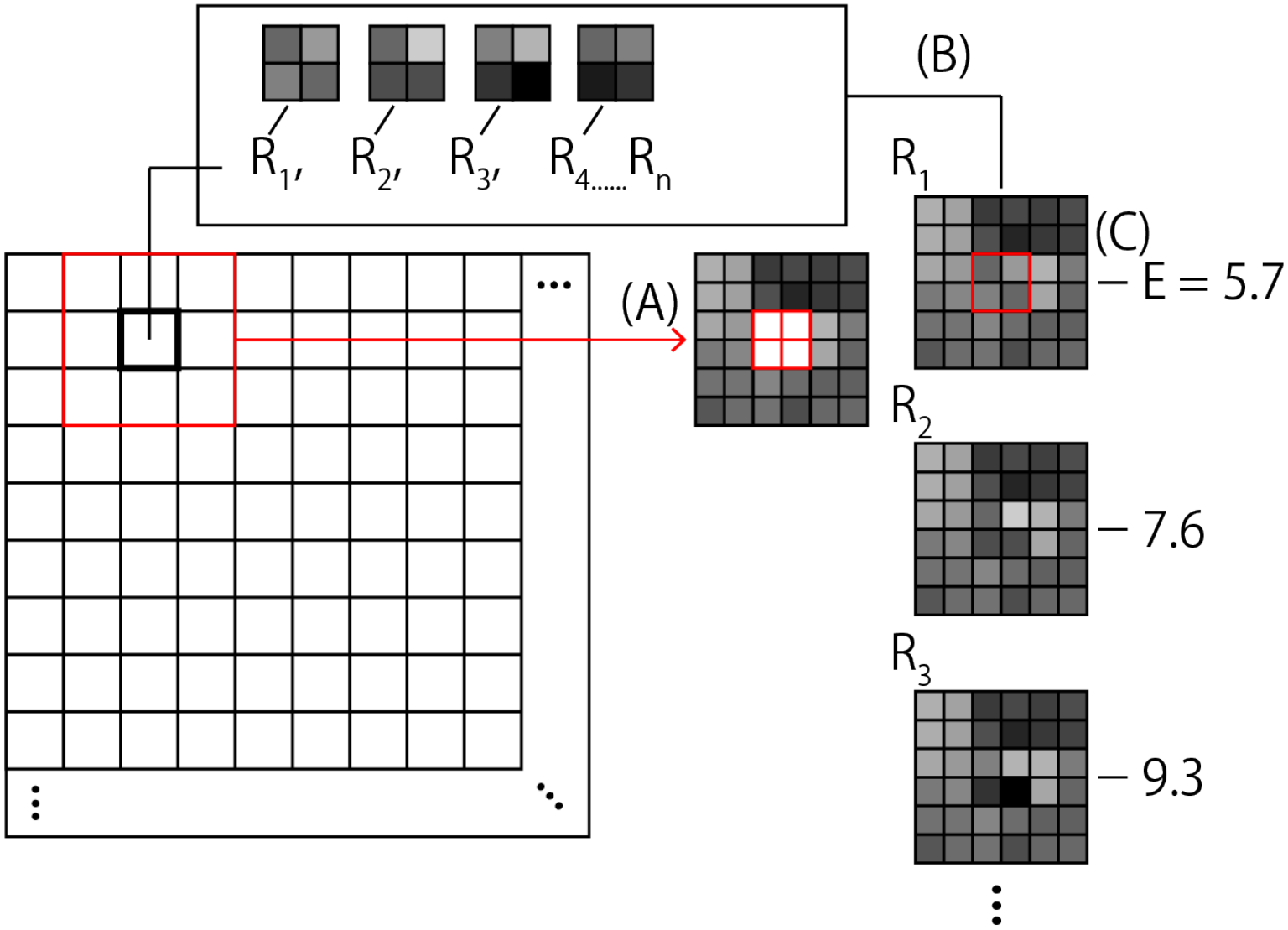
Laplacian minimization

(B) Put each candidate texel in the center.



Laplacian minimization

(C) Calculate Laplacian value for the minimize objective function.

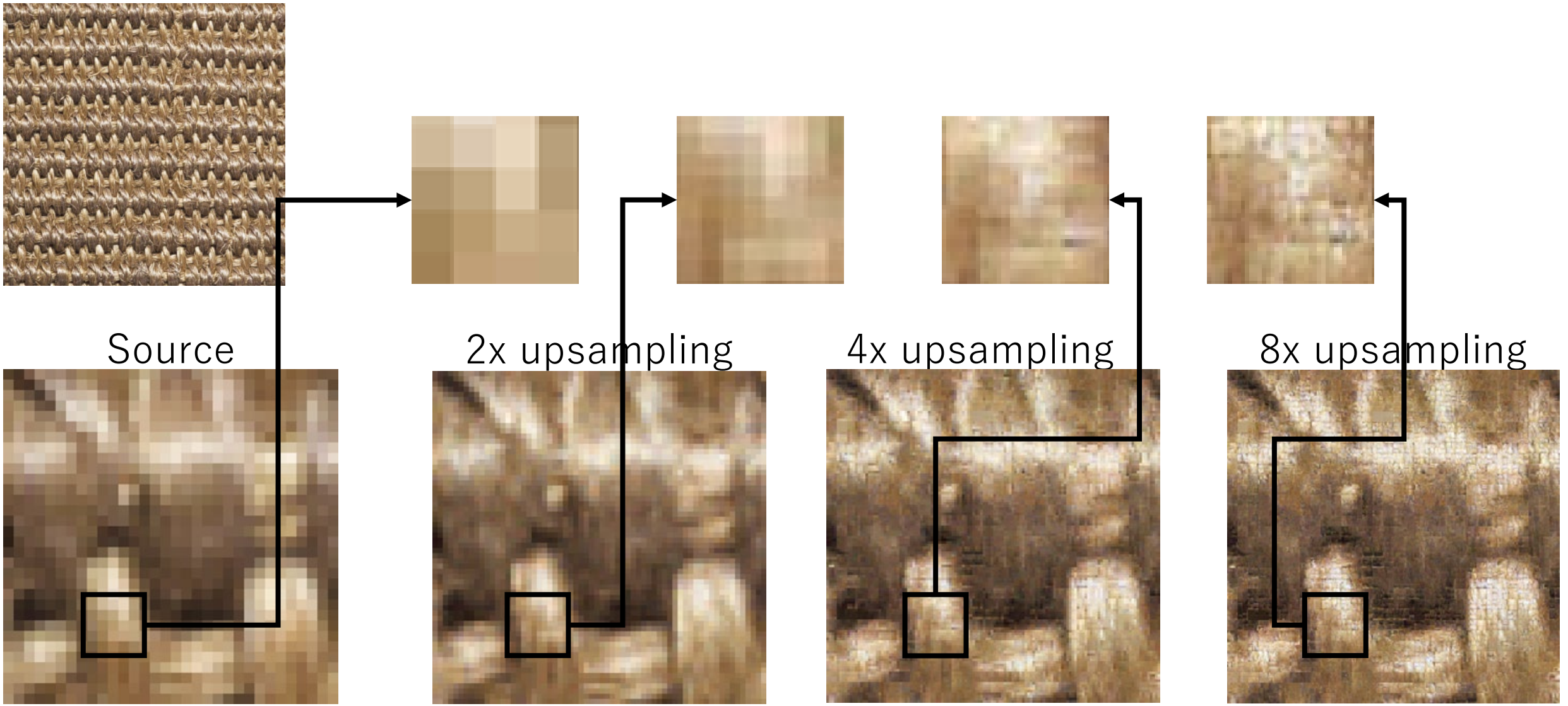


Iterative refinement result

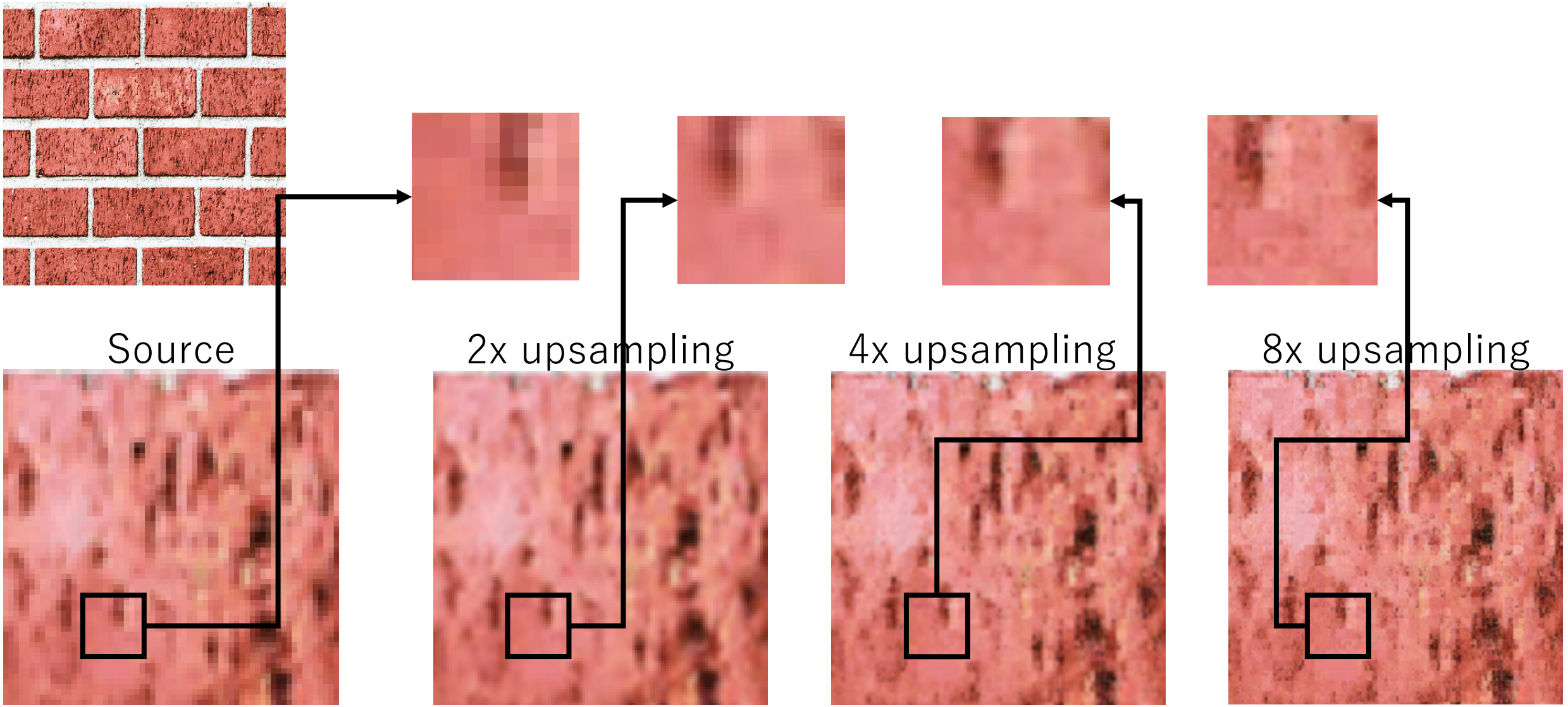
No iterative refinement

2 times Iterative refinement

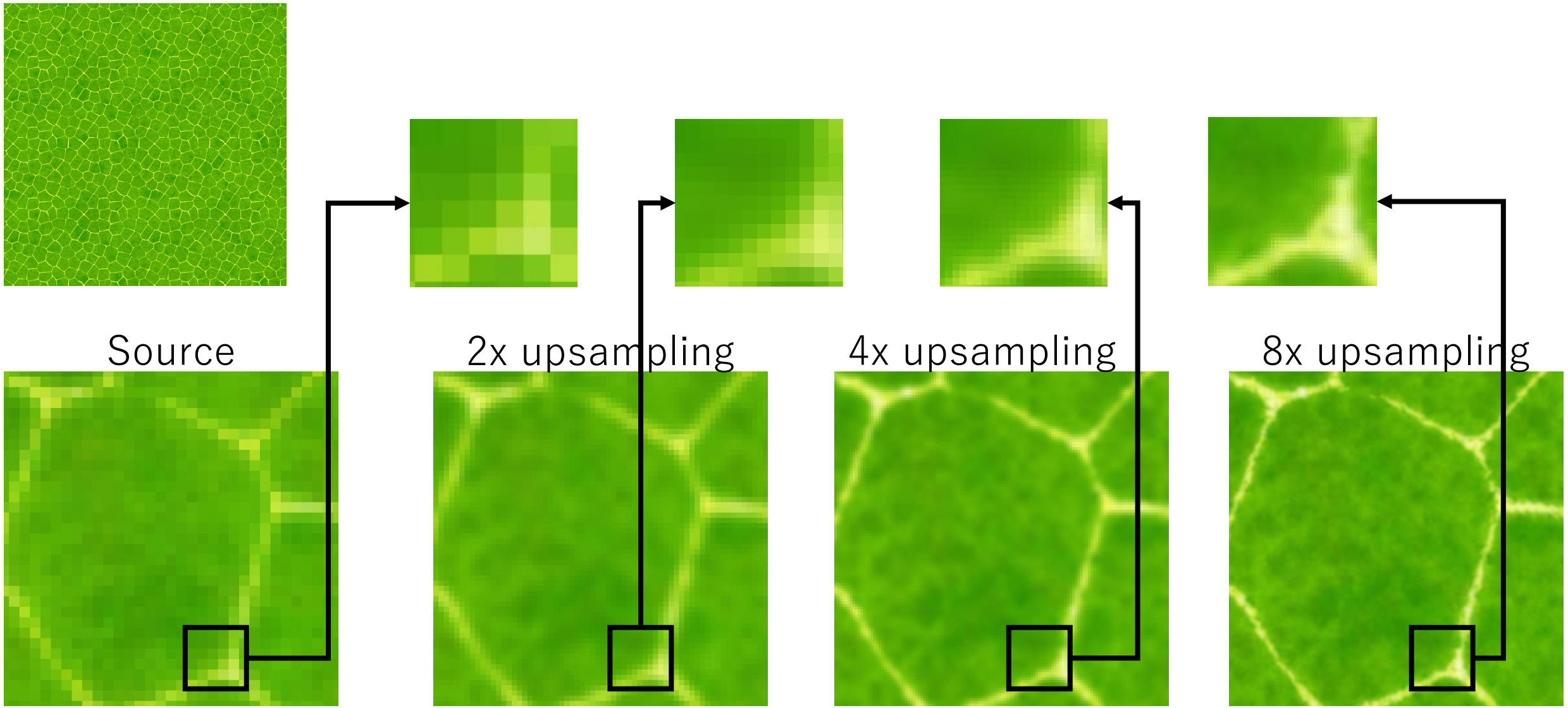
Results



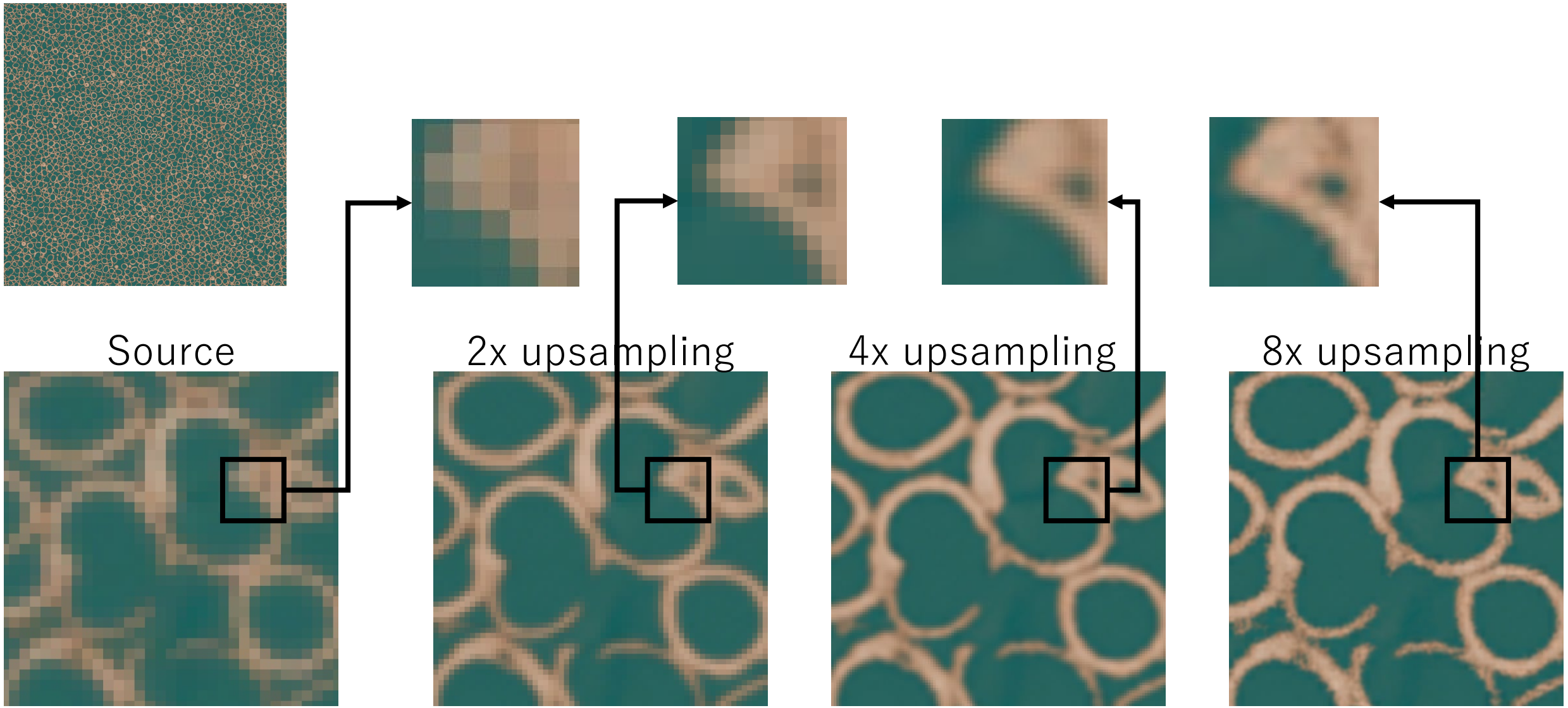
Results



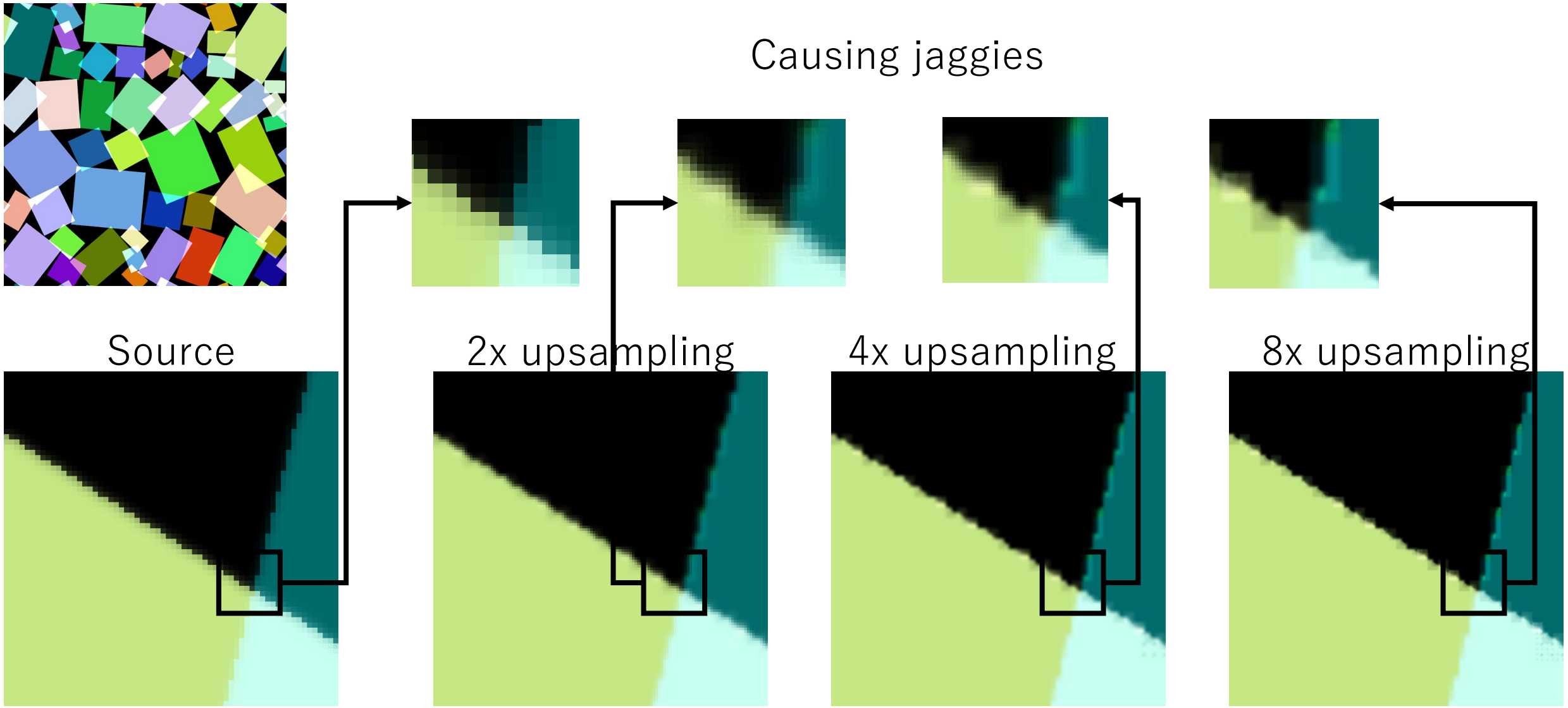
Results



Results

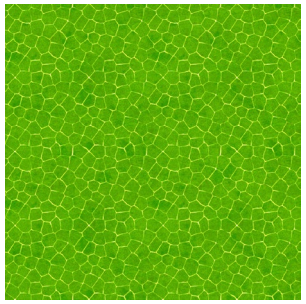


Results


Causing jaggies
Source
2x upsampling
4x upsampling
8x upsampling

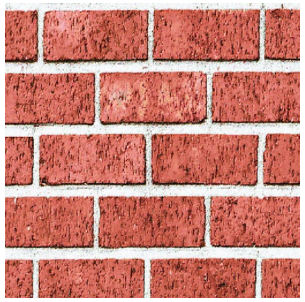
Evaluation

Grass:1024×1024



PSNR 29.1015

Brickwall:512×512

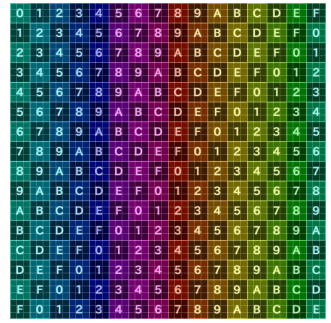


PSNR 23.0404

Mandrill: 512×512

PSNR 19.58179

UVTexture:512×512

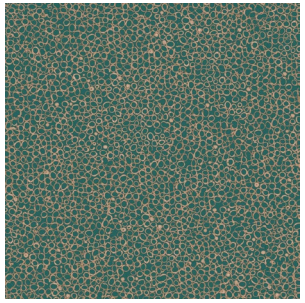


PSNR 22.3722

SisalFloor:1024×1024

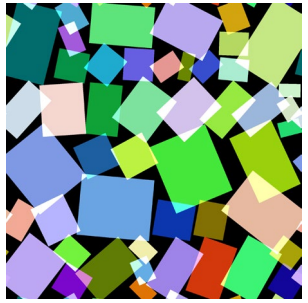
PSNR 23.3032

Alienskin: 1024×1024



PSNR 28.3720

RotateTile:1024×1024



PSNR 34.4637

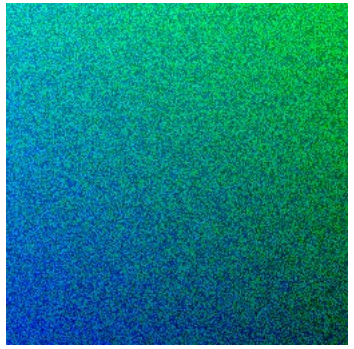
Evaluation

Time of upsampling(512x512px)

- Bilinear interpolation: 28 μ sec
- Deep convolutional network: 24.0 sec
- Our method: 184 μ sec

Half size texture

Offset texture



Coordiante texture

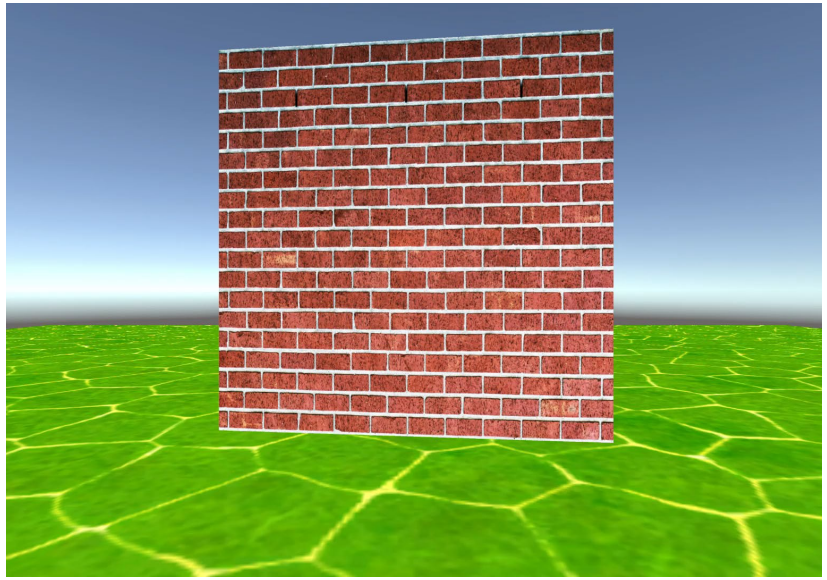
4K = 4096 × 4096
= 16777216(pixel)

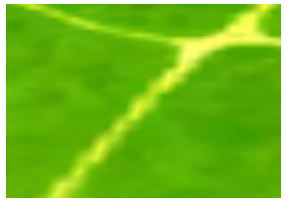
2K × 3 = 2048 × 2048 × 3
= 12582912(pixel)

Rendering

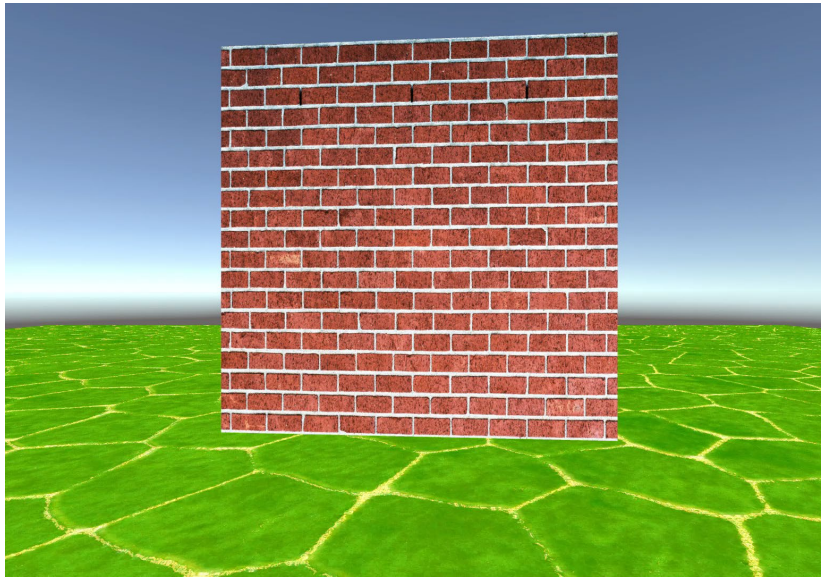
Magnified 64 times.

Bilinear





Our method



Summary and future work

- Recursive upsampling
  - Produce a plausible result with only one layer
  - Superior to **Super-resolution from a single image**
  - Reduced memory load
  - Faster decoding process
- Future work
  - More mipmap levels
  - Domain-block-to-block correspondence
  - Transferred to high-speed image compression methods.

Acknowledgment: PlatinumGames Inc.