Realtime Texture Upsampling on Graphics Hardware Using Fractal Coding

Yuto Kominami
Graduate School of Systems Design
Tokyo Metropolitan University
Tokyo, Japan
kominami-yuto@ed.tmu.ac.jp

Tomohiko Mukai
Graduate School of Systems Design
Tokyo Metropolitan University
Tokyo, Japan
tmki@acm.org

***Abstract*—The texture upsampling technique reduces the memory requirement and manual labor to create a high-definition texture by procedurally increasing the image resolution on the fly. We propose a realtime texture upsampling technique utilizing an underlying fractal structure of surface texture found in nature, such as in stone surfaces, tree bark, and animal skin. Our method encodes the source texture by a fractal compression technique in the authoring process. The cross-scale correspondences between two mipmap layers are encoded as compact fractal codes and packed in three types of GPU-friendly formats. The runtime computation increases the resolution of the encoded textures by recursively applying fractal decoding. Thanks to the concurrent data structures, the texture upsampling by any power of 2 are efficiently executed on GPUs. We also propose an iterative algorithm to refine the fractal code to improve the upsampling quality. We demonstrate the effectiveness of our proposed algorithm for synthesizing a high-resolution texture with semi-random patterns.**

***Index Terms*—texture, upsampling, fractal coding, graphics hardware**

I. Introduction

Realtime rendering of high-definition 3D graphics demands high-resolution (or high-res) content, including high-poly models, texture maps, and normal and displacement maps. A digital content creation tool provides many functions to create high-res assets efficiently and intuitively, whereas manual creation is still labor-intensive. Especially, texture creation requires hard manual labor as the image resolution increases, even though many sophisticated texture synthesis tools are available. For example, procedural techniques have been developed to generate various textures by simply tweaking intuitive parameters of expressions and scripts. Example-based techniques are also widely used to create a rich-detailed texture from a photograph or a stocked image in the authoring process. However, such a high-res texture requires a large memory capacity and bandwidth of graphics processing units (GPUs) in runtime computation.

Image upsampling or super-resolution technique is a promising approach to alleviate the problems. The upsampling technique synthesizes a higher resolution image from a low-res image on the fly. This approach significantly reduces memory usage and data transfer time, compensating for the additional computation time. For example, spline interpolation, such as bilinear and bicubic interpolation, is a classical technique to resize a texture size dynamically. The numerical approach guarantees a continuous and smooth result, but the high-frequency details are not produced. Deep learning techniques are widely used to reconstruct a higher resolution image with rich details from a low-res image. For example, commercially available technologies, such as AMD FSR[1], NVIDIA DLSS[2], and NVIDIA NIS [3], increases the resolution of a rendered scene as a post-processing at an interactive rate. However, the computational cost of the machine learning-based functions is still non-negligible for hard realtime applications.

Our goal is to develop a realtime method that generates a higher resolution texture from a low-res source image. Our motivation is to synthesize a *plausible* texture with less computational cost, which can be applied to background 3D objects in an interactive application. In other words, an accurate reconstruction of an original image from a degraded lower resolution image is not necessarily our goal. The problem relaxation is reasonable under the assumption that the application user pays more attention to the granularity of the rendered image than the accuracy. Furthermore, we focus on semi-regular or semi-random surface textures found in nature, such as tree bark, stone surface, and cloth surface. The texture frequently had a fractal structure where similar texel blocks appear multiple times in identical textures and across multiple scales.

Moreover, our research aims to develop a GPU-friendly realtime method. There are several ideal conditions for realtime rendering on a GPU. The first is that the computational procedure should be optimized for the highly parallel architecture. Each texel block must be processed independently with a well-designed concurrent data structure. The second is the deterministic and unbranching computation: since texture upsampling with conditional branching increases the computational cost on GPUs, the runtime module should be implemented using simple instructions. The third is memory efficiency; the cache-friendly memory layout and smaller footprint are crucial for maximizing the computational speed.

Our basic idea is to utilize the fractal structure of semi-random images for texture upsampling. Figure 1 illustrates the

[1]https://www.amd.com/en/technologies/fidelityfx-super-resolution
[2]https://www.nvidia.com/en-us/geforce/technologies/dlss/
[3]https://github.com/NVIDIAGameWorks/NVIDIAImageScaling

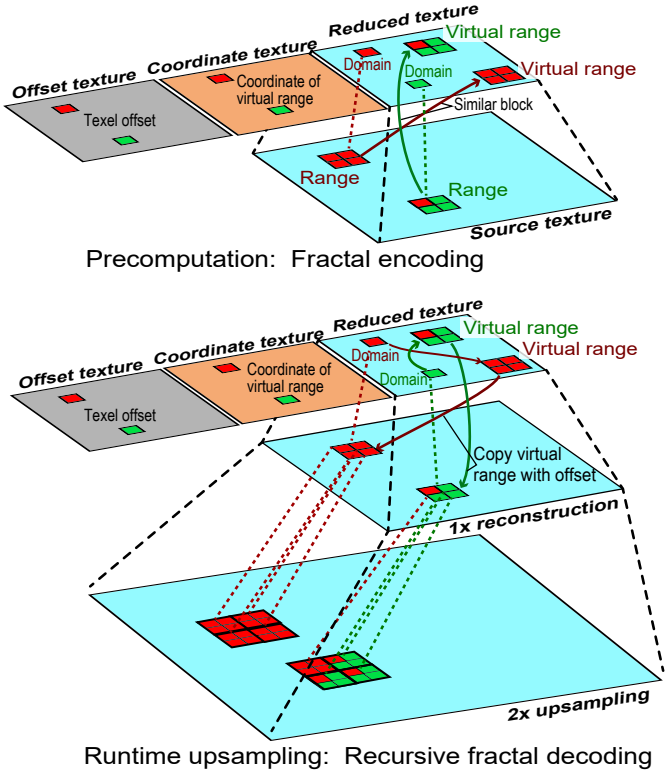


Fig. 1: Overview of the fractal coding-based texture upsampling. The domain-range correspondences are extracted in the authoring process. The texture resolution is efficiently increased by recursively applying the fractal decoding in the runtime process using our novel texture formats.

overview of our method. Our method encodes correspondences between a texel of a lower resolution texture and a small texel block of the original texture into fractal codes. The encoded information is compactly stored in three GPU-friendly data formats: reduce texture, coordinate texture, and offset texture. Thanks to the concurrent data structure, a visually plausible higher resolution texture can be synthesized in realtime by texel-wise parallel computation on a GPU. Our method enables texture upsampling by the power of 2 by recursively applying the fractal decoding with less computational cost. Moreover, we propose an iterative algorithm to refine the fractal code to improve upsampling quality. The parallel upsampling algorithm causes a block-noise-like artifact as the texel-wise upsampling neglects the continuity with surrounding texels. The refinement algorithm optimizes the fractal codes to minimize statistical differences among adjacent texel blocks of the upsampled image.

The strength of the proposed algorithm includes efficient and simple runtime computation. Our method is applicable to the upsampling of various semi-random textures. However, block-noise-like artifact becomes more noticeable as the scaling ratio increases since the parallel runtime algorithm does not consider the spatial coherence over the synthesized texture. Moreover, our method is built on strong assumptions about the fractal nature of the source texture. The upsampling quality deteriorates when the source texture has few similar texel blocks. These advantages and disadvantages of our method will be discussed through several experiments.

II. Related Work

Mipmapping is a standard approach to efficiently render the optimal resolution texture using a pre-created multi-resolution image. This technique constructs an image pyramid from the highest resolution image in the authoring process. A level-of-detail controller automatically selects an adequate mipmap level at the runtime process. For example, the highest resolution layer is used for a 3D object closest to the camera. The lower resolution layers are adaptively selected based on the distance from the camera. A problem with this approach includes the redundant data size to store all mipmap layers. Furthermore, the designer should create a high-definition texture. The manual creation is still tedious even though many procedural texture synthesis tools are available.

Upsampling of a digital image has been well studied in the fields of image processing and computer vision. There are many methods, such as rule-based upsampling methods, spline interpolation-based techniques, and the optimization-based approach [1]. An example-based upsampling is a promising approach that learns pixel block-wise correspondences between low and high-res images. The learned correspondence is then applied to a new low-res image to recover its most likely high-res version [2]. Deep neural networks have also been widely used for complex upsampling tasks [3], [4] and the on-the-fly upsampling of a rendered image. They provide high-quality results through a complex computation of deep neural networks. In contrast, our purpose is to develop a lightweight algorithm for synthesizing a plausible higher resolution texture by upsampling a source texture rather than accurately reconstructing a high-res image from the lower-res degraded image.

Our method utilizes a fractal structure in a natural image. Fractal coding is a method of compressing the information in an image by replacing it with parameters in a system of local iterative functions [5]. This method first divides the entire image into multiple blocks, called domains, without overlaps. Next, the larger blocks similar to the domain, called ranges, are searched for each domain. If an adequate range is found, the domain is replaced by the reference to the range block with a simple affine transformation matrix. This approximation dramatically reduces the data size when many domains are replaced by a small number of ranges [6]. Our work is inspired by fractal interpolation [7] that uses fractal coding for image upsampling. The fractal interpolation is also used for image upsampling in gradient domain [8]. These techniques provide quality results but are not suited for realtime computation.

Fractal coding usually requires a large amount of sequential computation. A parallel implementation method [9] executes a fractal encoding process on GPUs, whereas the decoding is sequentially processed that is not suitable for GPU implementation. Mip-pyramid texture compression method [10] uses

a local fractal coding strategy that enables random memory access and a parallel implementation by reducing the data size of fractal code. Our method introduces the basic idea of mip-pyramid texture for realtime upsampling.

Our method is also closely related to single image super-resolution techniques [11], [12], which can be regarded as a kind of fractal interpolation. These techniques are an example-based technique that generates a low-res/high-res pair of image patches from a source image using an image pyramid technique. For example, the single image super-resolution method [11] searches for patches in the reduced image that are similar to patches in the source image. The patch correspondences can be used to increase the resolution of the lower resolution patch. This approach was later extended to an example-based regression model that estimates a high-res patch from a low-res patch by learning multiple pairs of examples [13].

We employ a similar approach to find low-res/high-res correspondences across multiple scales. We assume the low-res/high-res correspondence is given in the mipmap; 2×2 texel block of the source texture to a corresponding texel in the reduced texture. Our method further approximates the cross-scale correspondence by replacing the high-res texel block with a similar texel block in the reduced texture, assuming the source image has a fractal structure. The approximation enables an efficient packing of the fractal code into a GPU-friendly texture format.

III. Algorithm

The proposed algorithm consists of the precomputation stage and the runtime computation stage. The fractal encoding of the source texture and the iterative refinement are executed on CPUs in the precomputation stage. The runtime upsampling is implemented as a pixel or compute shader program composed of simple arithmetic operations and texture fetch instructions on GPUs. This section explains the overview of fractal codes and the relationship with the texture upsampling. Next, we will explain how to encode a source texture into fractal codes as three types of smaller textures in §III-B. Then, the runtime upsampling procedure will be detailed in §III-C, and the iterative refinement will be explained in §III-D.

A. Fractal Codes For Upsampling

Our method first encodes a source texture into fractal codes. We introduce three types of half-size textures: reduced texture, offset texture, and coordinate texture for representing the encoded data in a GPU-friendly format. Let $T_1^*(\mathbf{p})$ be a texel value of the source texture at a 2D coordinate $\mathbf{p} = [p_x, p_y]$, and $R(\mathbf{c})$, $C(\mathbf{c})$, and $O(\mathbf{c})$ be a texel value at $\mathbf{c} = [c_x, c_y]$ in the reduced texture, the coordinate texture, and the offset texture, respectively. We here use a calligraphic symbol $\mathcal{T}(\mathbf{p})$ that denotes a 2×2 texel block composed of $T(p_x, p_y)$, $T(p_x + 1, p_y)$, $T(p_x, p_y + 1)$, and $T(p_x + 1, p_y + 1)$.

A typical fractal compression technique approximates a small texel block, called *domain*, by a similar texel block of different sizes, called *range*, with an affine transformation matrix. The traditional methods find a domain-range relation in the same source image. In contrast, our method utilizes a cross-scale correspondence [10], [11] between adjacent mipmap layers. We assume that the low-res/high-res correspondence is established between a domain texel in the reduced texture $R(\mathbf{c})$ and the 2×2 range block $\mathcal{T}_1^*(2\mathbf{c})$ in the source texture [10]. Furthermore, the range block is approximated by a *virtual range* in the reduced texture utilizing the fractal nature of the semi-random texture. Concretely, our method approximates a range block in the source texture using the encoded textures as follows.

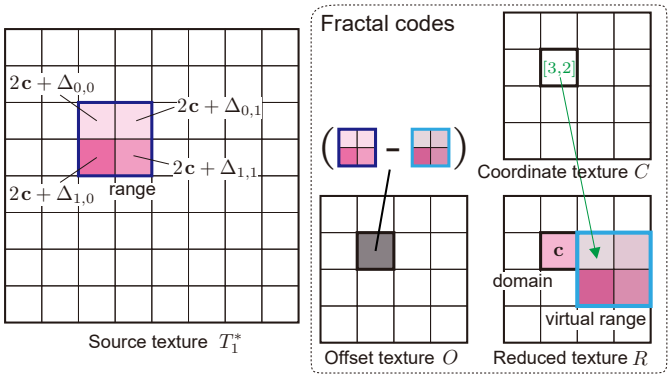


Fig. 2: Three types of half-size textures for fractal codes.

$$\begin{cases} T_1^*(2\mathbf{c}+\Delta_{0,0}) & \approx R(C(\mathbf{c})+\Delta_{0,0})+O(\mathbf{c}) \\ T_1^*(2\mathbf{c}+\Delta_{1,0}) & \approx R(C(\mathbf{c})+\Delta_{1,0})+O(\mathbf{c}) \\ T_1^*(2\mathbf{c}+\Delta_{0,1}) & \approx R(C(\mathbf{c})+\Delta_{1,0})+O(\mathbf{c}) \\ T_1^*(2\mathbf{c}+\Delta_{1,1}) & \approx R(C(\mathbf{c})+\Delta_{1,1})+O(\mathbf{c}) \end{cases}, \quad (1)$$

where $\Delta_{a,b}$ is an integer vector $\Delta_{a,b} = [a,\ b]$ representing a texel offset. The reduced texture R is generated from the source texture T_1^* using an arbitrary downsampling algorithm. The coordinate texture $C(\mathbf{c})$ represents a mapping from the 2D coordinate of the domain texel $R(\mathbf{c})$ to the 2D coordinate of the virtual range $\mathcal{R}(C(\mathbf{c}))$ that best approximates $\mathcal{T}_1^*(2\mathbf{c})$. A texel value of the offset texture $O(\mathbf{c})$ contains the mean texel error between the range and virtual range.

Figure 2 illustrates an example of the domain-range relation and the fractal codes. In this example, 2×2 block in the source texture $\mathcal{T}_1^*(2\mathbf{c})$ is the range block, and the corresponding texel $R(\mathbf{c})$ in the reduced texture is the domain texel. The coordinate of the left top texel of the virtual range closely similar to the range block $\mathcal{T}_1^*(2\mathbf{c})$ is stored in the coordinate texture $C(\mathbf{c})$. The mean difference of texel value between the range and the virtual range is stored in the offset texture $O(\mathbf{c})$. The original range block is approximated by applying the offset $O(\mathbf{c})$ to each texel of the virtual range $\mathcal{R}(C(\mathbf{c}))$.

The fractal decoding process can be regarded as a texture upsampling where the source texture is reconstructed from the half-size reduced texture. We apply the domain-to-range upsampling rule of Equation 1 recursively to generate a higher resolution texture by assuming the decoded image to be the reduced texture of the next upsampling level. This approach achieves an efficient data compression and runtime synthesis thanks to the packed texture format. However, since our

fractal coding is a lossy algorithm, the reconstructed texture causes an error as the reduced texture loses the high-frequency component of the source texture. We think that a certain amount of reconstruction error can be tolerated to generate a high-res texture with plausible details.

B. Fractal Encoding

The encoded textures are generated in the precomputation stage using a fractal encoding technique. First, the source texture is divided into multiple ranges where each range is 2×2 texel block. The domain-range relation is then established between a domain texel in the reduced texture $R(\mathbf{c})$ and the range block in the source texture $\mathcal{T}_1^*(2\mathbf{c})$. Next, virtual range $\mathcal{R}(C(\mathbf{c}))$ that is similar to the range block $\mathcal{T}_1^*(2\mathbf{c})$ is searched in the reduced texture using an off-the-shelf template matching algorithm. We use the sum of squared error of texel value as similarity measure for the template matching as follows.

$$x^*, y^* = \arg\min_{x,y} E^{\text{texel}}(x, y) \ , \tag{2}$$

$$E^{\text{texel}}(x, y) = \sum_{\Delta x=0}^{1} \sum_{\Delta y=0}^{1} |T_1(2c_x + \Delta x, 2c_y + \Delta y) - R(x + \Delta x, y + \Delta y)|_2^2 \ , \tag{3}$$

where $|\cdot|_\alpha$ represents L^α norm. The optimized coordinate value is stored to the coordinate texture as $C(\mathbf{c}) = [x^*,\ y^*]$.

The offset texture value $O(\mathbf{c})$ is finally determined as the mean of approximation error of the virtual range as follows:

$$O(\mathbf{c}) = \frac{1}{4} \sum_{\Delta x=0}^{1} \sum_{\Delta y=0}^{1} T_1^*(2c_x + \Delta x, 2c_y + \Delta y) - R(x^* + \Delta x, y^* + \Delta y) \ . \tag{4}$$

The reconstruction quality is degraded by averaging the approximation error of four texels, especially when the gradient in the range block is large and there is a significant dissimilarity between the range and virtual range. We could improve the upsampling quality by storing the texel-wise error as the offset texture, but it increases computation time and data size.

C. Runtime Upsampling By Fractal Decoding

The reduced texture can be upsampled by powers of two. First, the texture T_1 whose size is equivalent to the source T_1^* is approximated using the encoded textures based on Equation 1 as follows.

$$T_1(2\mathbf{p}_1 + \Delta_1) = R(\mathbf{c}_1) + O(\mathbf{c}_0) \ , \tag{5}$$

$$\mathbf{c}_1 = C(\mathbf{c}_0) + \Delta_1 \ , \tag{6}$$

where $\mathbf{c}_0 = [\text{floor}(p_{1,x}/2^1), \text{floor}(p_{1,y}/2^1)]$. Δ_n is an integer offset at n-th level which is any one of $\Delta_{0,0}$, $\Delta_{0,1}$, $\Delta_{1,0}$, or $\Delta_{1,1}$, as shown in Figure 2.

Similarly, the twice larger texture T_2 is synthesized using the encoded textures. Figure 3 illustrates the upsampling procedure. A domain texel $R(\mathbf{c}_0)$ is first replaced by the sum of virtual range $\mathcal{R}(C(\mathbf{c}_0))$ and the offset value $O(\mathbf{c}_0)$. The

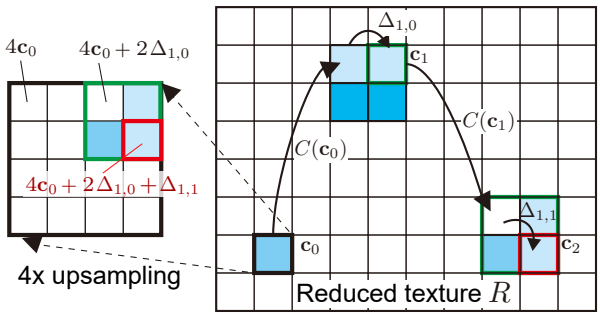


Fig. 3: Recursive upsampling using the reduced texture.

right top texel of the virtual range $R(C(\mathbf{c_0}+\Delta_{1,0}))$ is next upsampled using the virtual range at $C(\mathbf{c}_1) = C(C(\mathbf{c}_0 + \Delta_{1,0}))$. The virtual range $R(C(\mathbf{c}_1)) + O(\mathbf{c}_1) + O(\mathbf{c}_0)$ is used as the right top 2×2 block as framed in green. The synthesis of the twice larger texture T_2 is formulated as Equation 7 and 8.

$$T_2(\mathbf{p}_2 + \Delta_2) = R(\mathbf{c}_2) + O(\mathbf{c}_1) + O(\mathbf{c}_0) \ , \tag{7}$$

$$\mathbf{c}_2 = C(\mathbf{c}_1) + \Delta_2 \ , \tag{8}$$

where $\mathbf{c}_0 = [\text{floor}(p_{2,x}/2^2), \text{floor}(p_{2,y}/2^2)]$. Inductively, the 2^n fold upsampling procedure is formulated as a recursive equation as follows.

$$T_n(\mathbf{p}_n + \Delta_n) = R(\mathbf{c}_n) + \sum_{i=0}^{n-1} O(\mathbf{c}_i) \ , \tag{9}$$

$$\mathbf{c}_n = C(\mathbf{c}_{n-1}) + \Delta_n \ . \tag{10}$$

Our recursive upsampling algorithm takes linear time with respect to the integer scaling factor n as Equation 9 and 10 indicate.

D. Fractal Code Refinement

Any stage in the runtime upsampling does not consider the spatial coherence with the neighboring texels and blocks. Our method causes block noise-like artifacts in the upsampled image, especially when the scaling ratio is large. To improve the upsampling quality, we extend the precomputation of fractal coding using a spatial proximity condition and a synthesis-analysis-refinement approach.

First, the dissimilarity measure in the template matching for the queried domain texel $R(\mathbf{c})$ and corresponding range block $\mathcal{T}_1^*(2\mathbf{c})$ is weighted to consider the spatial proximity in the reduced texture as follows.

$$E(x, y) = E^{\text{texel}}(x, y) + E^{\text{coord}}(x, y) \ . \tag{11}$$

The definition of texel dissimilarity term E^{texel} is same as in Equation 3. The coordinate dissimilarity measure E^{coord} is defined as a Manhattan distance:

$$E^{\text{coord}}(x, y) = \beta \left(|x - 2c_x|_1 + |y - 2c_y|_1 \right) \ , \tag{12}$$

where β is a weighting coefficient, which was set to $\beta = 1.0$ in our experiments. This weighting strategy preferentially finds neighboring blocks.

The weighted dissimilarity term is used to find multiple candidates that satisfy the following equation:

$$E(x, y) < E(x^*, y^*) + \Delta E_{\max} \ , \tag{13}$$

where (x^*, y^*) denotes the texel coodinate that has the smallest dissimilarity, and $\Delta E_{\max}$ is the tolerance limit from the minimum dissimilarity.

Second, the virtual range is iteratively replaced by the best candidate, minimizing the difference with the neighboring texel blocks. We use variance of Laplacian value for the minimization criterion as

$$\sum_{\mathbf{c}} \left(L(\mathbf{c}) - \bar{L}\right)^2 \ , \tag{14}$$

where $L(\mathbf{c})$ is a 4-neighborhood Laplacian at $\mathbf{c}$ and $\bar{L}$ is the mean of the all texels. The refinement of coordinate texture $C(\mathbf{c})$ is consequently formulated as Equation 15.

$$C(\mathbf{c}) = \underset{(x,y)\in\Omega}{\arg\min} \sum_{(c_x, c_y)} \left(L(\mathbf{c}) - \bar{L}\right)^2 \ , \tag{15}$$
$$\Omega := \{(x, y) | E(x, y) < E(x^*, y^*) + \Delta E_{\max}\} \ .$$

The offset texture $O(\mathbf{c})$ is also updated by Equation 4 using the refined coordinate.

We use a synthesis-and-analysis approach to update the domain-range relation iteratively. $C(\mathbf{c})$ and $O(\mathbf{c})$ are updated by selecting an optimal candidate to minimize the variance of Laplacian (Equation 14) while fixing the other range blocks corresponding the domain $\mathbf{c}' \neq \mathbf{c}$. This block coordinate descent refinement repeats for all domain texels. This approach achieves a better quality of the synthesized texture, although the convergence to the global optimum is not guaranteed.

IV. Implementation

We implemented fractal encoding using Python with the OpenCV library. The runtime upsampling module was implemented as a pixel shader of Unity. The reduced texture R is created by a standard downsampling algorithm. Both the offset texture O and coordinate texture C are stored in two-dimensional RGB texture format, in which an eight-bit represents each color channel. To compactly pack $C(\mathbf{c})$ into eight bit format, the exploration area of Equation 2 is limited to even-index texel blocks within 511×511 area centered at the domain as $x^* \in [c_x - 255, c_x + 255]$ and $y^* \in [c_y - 255, c_y + 255]$. As a result, $C(\mathbf{c})$ stores the half of coordinate value relative to $[c_x, c_y]$, and the integer value within $[-125, 126]$ is stored in two channels of each texel of the coordinate texture.

V. Results

We validated our proposed method through several experiments. In all our experiments, we set the tolerance limit $\Delta E = 100$. The computational performance is measured on a laptop PC with AMD Ryzen5 3600, 24 GB RAM, and NVIDIA GeForce RTX 2070.

Figure 4 shows the 8x upsampling results of the sisalfloor texture using the different three methods. The source texture has typical semi-regular patterns where combinations of random and regular patterns appear cyclically. The bilinear provided a smooth result but lost the high-frequency details. A deep learning-based method, called LapSRN [14], [15], preserved the high-frequency details better than bilinear, but the anisotropic fiber texture was not obtained. In contrast, our proposed method well preserved fibrous features. Moreover, our method is significantly faster than LapSRN owing to embarrassingly parallel computation with simple instructions. Note that we obtained trained models for LapSRN from the OpenCV website [4]. We could further improve the upsampling quality of LapSRN using another dataset appropriate for our experimental dataset.

Figure 5 summarizes the upsampling result of the sisalfloor texture by our method at different scales. The fibrous features were well preserved even when increasing the resolution by eight times, and there are few artifacts thanks to the fractal code refinement.

Figure 7 shows the results of the microscopic leaf texture. The granularity of the texture was well enhanced without fundamental artifacts. Sharper boundaries were produced between the green and yellow regions and between the darker and light green regions. Our method successfully worked for a source texture with a mixture of slight and significant color changes.

Figure 8 shows the upsampling results of the manufactured brickwall texture. These results indicate that the texture features were well preserved up to 4x upsampling. However, noticeable block noise and impulsive black texels were caused at 8x upsampling regardless of the fractal code refinement. This result demonstrates a drawback of the texel-wise parallel algorithm without considering the spatial coherence.

Figure 9 demonstrates the other limitation of our method. The upsampled results of the rotatetiles texture lost the sharp boundaries and became rough and jaggy. The low-quality result was produced because of the low similarity between the range block and the virtual range. Our method assumes that the source texture has a cross-scale fractal structure, i.e., there are many similar blocks among multiple mipmap levels. We think that the search for the virtual range failed since there were a few similar texel blocks in the source texture, as shown in Figure 9(a).

Table I summarizes the quality evaluation of 1x reconstruction of the four textures. We used the peak signal-to-noise ratio (PSNR) to quantitatively evaluate the accuracy of the reconstructed texture T_1 from the reduced texture R. These results demonstrate the inaccuracy of image reconstruction by our method, as the bilinear interpolation and LapSRN method indicated better accuracy for the sisalfloor texture, with PSNR=28.46 and 30.29, respectively. However, we think a certain amount of error is acceptable to synthesize plausible high-res textures with less visible artifacts.

[4]https://github.com/opencv/opencv_contrib/tree/master/modules/dnn_superres

(a) Source texture

(b) Region of interest

(c) Bilinear interpolation

(d) LapSRN method

(e) The proposed method

Fig. 4: Comparison with two conventional methods

TABLE I: Quantitative quality evaluation using PSNR

sisalfloor	leaf	brickwall	rotatetile
23.3032	29.1015	23.0404	34.4637

Finally, runtime performance was evaluated using a simple scene as shown in Figure 10. The leaf and brickwall textures were mapped onto the ground floor and the cube, respectively. The baseline method is the standard bilinear interpolation technique and mipmaps. We measured the turnaround time of the pixel shader assigned to the two models. The bilinear interpolation and our method spent 28 usec and 184 usec, respectively. Our method took a longer computation time due to the recursive upsampling. We think it is worth spending the additional computation to synthesize high-res texture on the fly with less memory requirement.

VI. Discussion

In this study, we have developed a realtime, GPU-friendly upsampling algorithm. The technical contribution of our method is twofold. First, the cross-scale correspondences between two mipmaps are extracted as fractal codes. Second, the fractal codes are compactly packed into three types of concurrent data formats thanks to the virtual range representation. These fundamental ideas enable the simple implementation of the runtime computation, the small number of texture samplings, and a realtime computation of recursive fractal decoding.

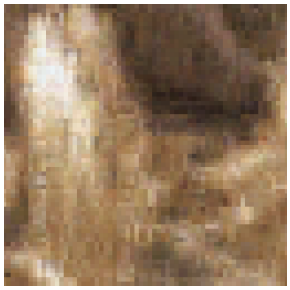
(a) Region of interest

(b) 2x upsampling

(c) 4x upsampling

(d) 8x upsampling

Fig. 5: Upsampling of sisalfloor texture at difference scales.



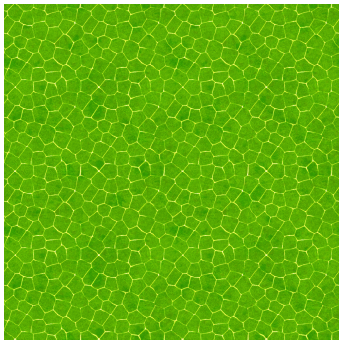
(a) None

(b) Once

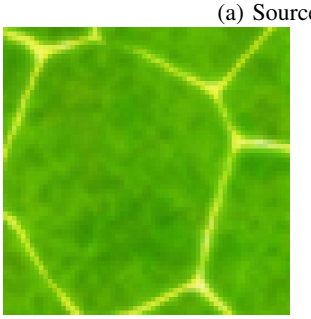
(c) Twice

(d) Three times

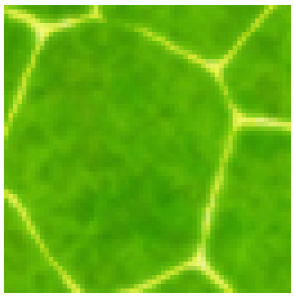
Fig. 6: Effect of the number of refinement iterations.



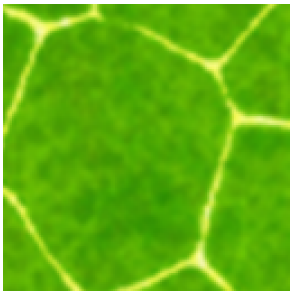
(a) Source texture



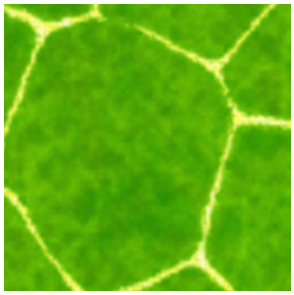
(b) Region of interest



(c) 2x upsampling



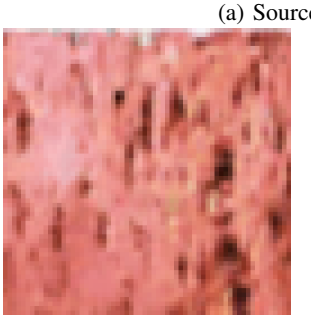
(d) 4x upsampling



(e) 8x upsampling

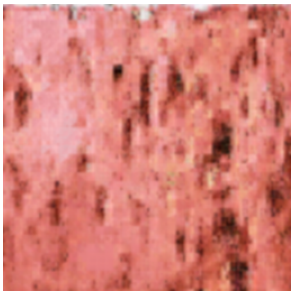
Fig. 7: Upsampling results of leaf texture

(a) Source texture

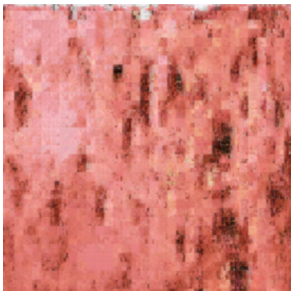


(b) Region of interest

(c) 2x upsampling



(d) 4x upsampling



(e) 8x upsampling

Fig. 8: Upsampling results of brickwall texture

Our method assumes that the source texture has a fractal structure, under which the plausible upsampling is possible for semi-regular or semi-random textures. Our experiments demonstrated that this assumption is valid for natural images such as tree bark, stone, and cloth surface. However, the upsampling quality deteriorates when the source texture includes a few similar texels. For example, our method is unsuitable for a geometrically-designed texture with many colors, regularly-shaped elements, and texts. We should investigate another approach suitable for such geometric patterns. The integration of a lightweight denoiser will be helpful in reducing artifacts.

Our upsampling algorithm is customized for color maps. Realtime upsampling of normal, displacement and the other maps should be explored for high-quality rendering with fewer data and computation. Our future work also includes the use of more mipmap levels. The current algorithm only uses two mipmaps to establish correspondences between domain texel and virtual range blocks. The upsampling quality could be improved by utilizing the fractal structure among three texture scales. Moreover, instead of texel-to-block correspondence, it is possible to use block-to-block correspondence. These extensions would contribute to quality improvement in compensation for additional computational costs.

Acknowledgments

This work was supported by PlatinumGames Inc.

References

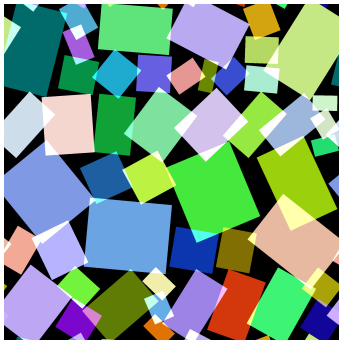
[1] J.D. van Ouwerkerk. Image super-resolution survey. *Image and Vision Computing*, 24(10):1039–1052, 2006.

[2] W.T. Freeman, T.R. Jones, and E.C. Pasztor. Example-based super-resolution. *IEEE Computer Graphics and Applications*, 22(2):56–65, 2002.

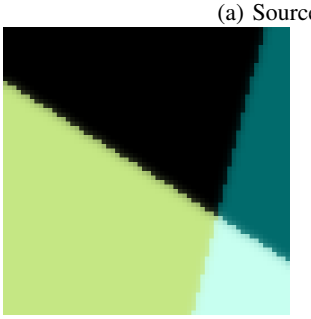
[3] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *European Conference on Computer Vision*, pages 184–199, 2014.

[4] Z. Wang, J. Chen, and S. H. Hoi. Deep learning for image super-resolution: A survey. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 43(10):3365–3387, 2021.

[5] M.F.Barnsley. *Fractals Everywhere*. Academic Press, 2005.



(a) Source texture



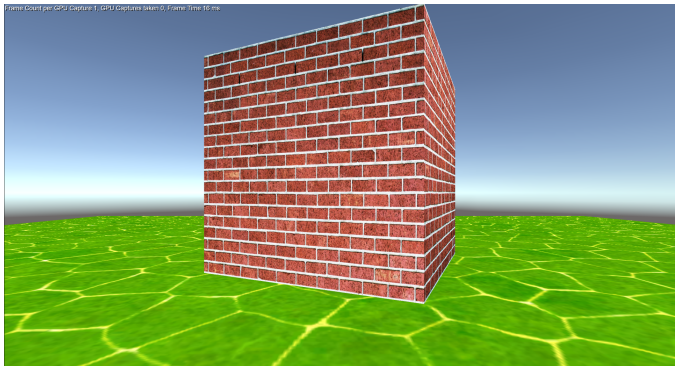
(b) Region of interest

(c) 2x upsampling

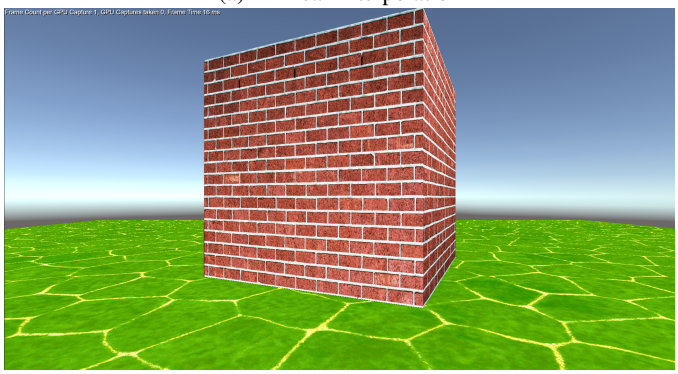
(d) 4x upsampling

(e) 8x upsampling

Fig. 9: Upsampling results of rotatetile texture



(a) Bilinear interpolation



(b) Our method

Fig. 10: Performance comparison using a simple scene.

[6] A.E. Jacquin. A novel fractal block-coding technique for digital images. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2225–2228, 1990.

[7] H. Honda, M. Haseyama, and H. Kitajima. Fractal interpolation for natural images. In *International Conference on Image Processing*, volume 3, pages 657–661, 1999.

[8] Licheng Yu, Yi Xu, Hongteng Xu, and Xiaokang Yang. Self-example based super-resolution with fractal-based gradient enhancement. In *IEEE International Conference on Multimedia and Expo Workshops*, pages 1–6, 2013.

[9] Abir Al Sideiri, Nasser Alzeidi, Mayyada Al Hammoshi, Munesh Singh Chauhan, and Ghaliya AlFarsi. Cuda implementation of fractal image compression. *Journal of Real-Time Image Processing*, 17(5):1375–1387, 2020.

[10] J. Stachera and P. Rokita. Fractal-based hierarchical mip-pyramid texture compression. *Machine Graphics & Vision International Journal*, 15(3):607–619, 2006.

[11] Daniel Glasner, Shai Bagon, and Michal Irani. Super-resolution from a single image. In *IEEE 12th International Conference on Computer Vision*, pages 349–356, 2009.

[12] Gilad Freedman and Raanan Fattal. Image and video upscaling from local self-examples. *ACM Trans. Graph.*, 30(2), 2011.

[13] Jianchao Yang, Zhe Lin, and Scott Cohen. Fast image super-resolution based on in-place example regression. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1059–1066, 2013.

[14] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[15] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Fast and accurate image super-resolution with deep laplacian pyramid networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.