Repeatable Texture Sampling with Interchangeable Patches

Martin Kolář · Alan Chalmers · Kurt Debattista

Abstract Rendering textures in real-time environments is a key task in computer graphics. This paper presents a new parallel patch-based method which allows repeatable sampling without cache, and doesn't create visual repetitions. Interchangeable patches of arbitrary shape are prepared in a preprocessing step, such that patches may lie over the boundary of other patches in a repeating tile. This compresses the example texture into an infinite texture map with small memory requirements, suitable for GPU and ray tracing applications. The quality of textures rendered with this method can be tuned in the offline preprocessing step, and they can then be rendered in times comparable to Wang tiles. Experimental results demonstrate combined benefits in speed, memory requirements, and quality of randomization when compared to previous methods.

Keywords Texture Synthesis · Texture Mapping · Parallel Rendering · Ray Tracing

1 Introduction

Texture synthesis is a core process in Computer Graphics and design. It is used extensively in a wide range of applications, including computer games, virtual environments, manufacturing, and rendering. Crucial points on which current methods compete are perceived texture quality, rendering speed, scale considerations, and memory requirements.

In order to allow rendering with ray-tracing, and to render textures in real time, current methods need to

University of Warwick
WMG
Tel.: +44-024-7652-2397
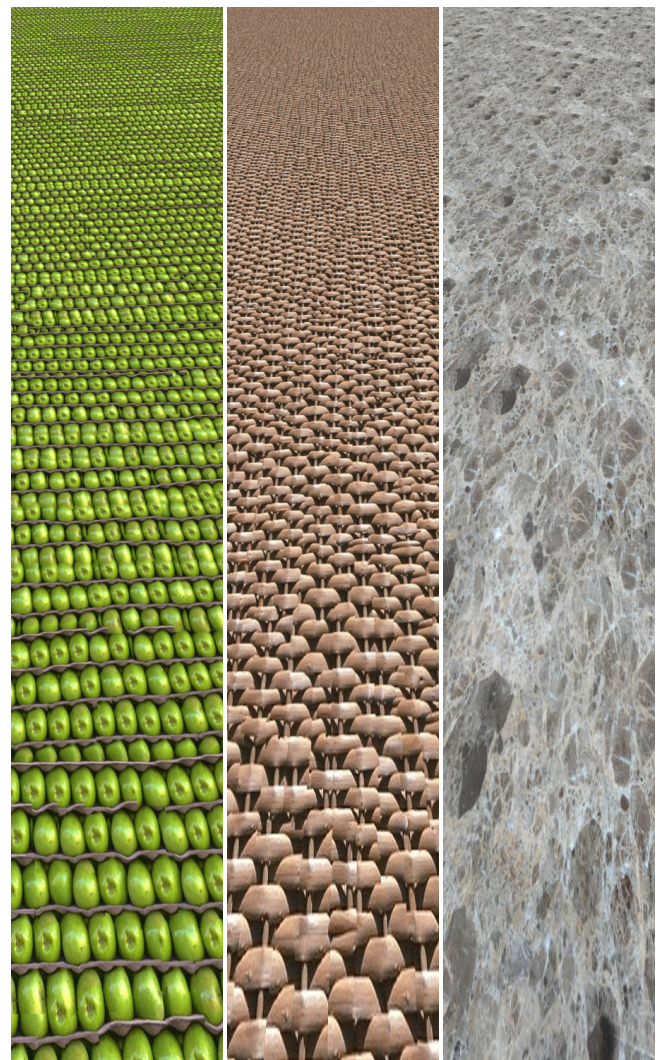E-mail: {m.kolar,alan.chalmers,k.debattista}@warwick.ac.uk

Fig. 1: Output textures from our algorithm display white-noise properties, without using cache (images are linearly transformed without interpolation to vanish at the horizon, results without offline manual tuning)
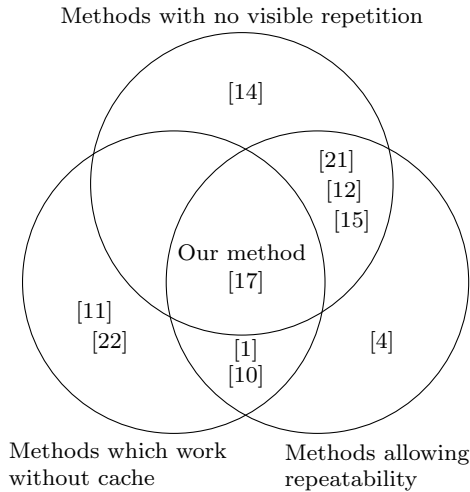
Fig. 2: Venn diagram of the trade-offs between current example-based parallel texture synthesis algorithms

run in parallel and on a GPU, without hindering the perceived quality.

As shown in Figure 2, our method addresses the main requirements for example-based parallel texture synthesis algorithms. This paper introduces a method which allows parallel texture synthesis with patches of arbitrary shape, without the necessity of a fixed repeating patch boundary. In previous work [17], selection of interchangeable patches at runtime was also possible, but required a repeating fixed patch boundary. As discussed later, the method presented here has similar pre-processing complexity, memory consumption, and rendering speed, but allows a wider class of interchange types with higher variability, resulting in a higher quality texture.

Our method allows the sampling of any pixel in the output texture with a deterministic algorithm, without requiring any information from the pixels that have already been synthesised. Therefore, adjacent pixels can be synthesised in parallel in separate threads, which do not communicate.

Large textures are rendered by randomly selecting subsets of prepared patches in a parallel and repeatable manner. These precomputed patches are stored efficiently, allowing seamless integration in GPU, and the texture is rendered independently for each pixel on-the-fly, allowing repeatable parallel access to an infinite, non-periodic texture, appropriate for ray tracing applications.

The paper is structured as follows: Section 2 discusses previous work, and how our method improves on the state-of-art in parallel texture synthesis. Section 3

outlines the high-level concept behind the method, and section 4 goes through how these points are implemented. Method outputs, their comparison to other work, and other results are presented in section 5, and future work and the conclusion are in section 6.

2 Previous Work

Sequential exemplar-based texture synthesis falls into one of three classes [19]: pixel-based methods, patch-based methods, and texture-optimisation methods. Parallel texture synthesis can be divided into an additional three: dependency-tree methods, constant-time tiling methods, and non-constant-time tiling methods.

Sequential pixel-based methods [3] consider each output pixel in sequence, while sequential patch-based methods [2] replicate entire patches, optimising seams using an algorithm such as Graph Cut [9]. Instead of performing the process once, patches can be placed iteratively over the output until desired quality is achieved [8,21]. However, sequential algorithms are not suitable for simultaneous synthesis of disjoint regions, because the space between them needs to be synthesised as well.

Where the entire texture cannot be held in memory, but needs to be generated on-the-fly, parallel texture synthesis methods can be used. The naïve approach to reduce rendering time is to create a repeating tile from an input exemplar, such that the edges fit [20]. This causes visibly noticeable "tiling" effects. Tile-based runtime synthesis relies on offline-preparation contents of a texture map, which are then placed on a rendered surface. Such placement schemes can be done in a number of different ways using a rectangular grid: Ammann tiles [6], Wang tiles [1], stochastic tiles [18], s-tiles [22], and colored corners [10]. Triangular [13] grids have also been used. These approaches make the output pixel retrieval a constant-time operation for output textures of arbitrary size. However, they create visible repeated edges and grid patterns when zoomed out, as shown in Figure 3.

In turn, this visible aberration is addressed by non-rectangular region copying, such as megatexture [14], virtual texturing [4,16], and patch-based methods [15]. However, these methods rely on cached information, which can cause temporal artefacts when a scene is re-rendered in a different order. To allow a different part of the scene to be rendered elsewhere, in the next video frame, or to be able to revisit a texture in a virtual environment, it is desirable to guarantee that a texture rendered again from the same compact seed will be identical to one rendered previously. This quality is referred to as "repeatability". (Not to be confused with
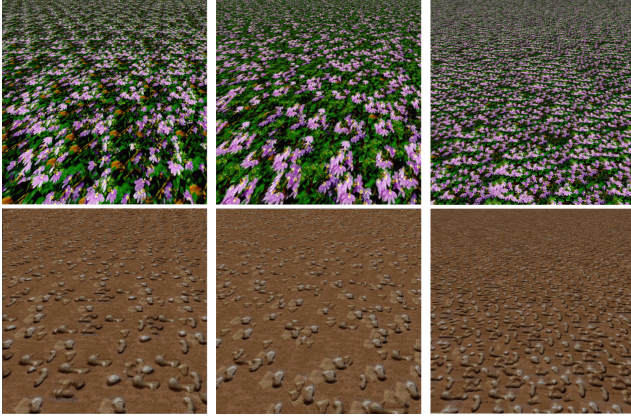
Fig. 3: Comparison with Wang tiles using 2 different borders (16 tiles) [1] (left), [17] (middle) and our results (right). Sampled linearly. Top exemplar is 268x230, bottom 512x512

"repetition", which is generally undesirable in synthesised textures)

Parallel non-constant-time patch replacement methods [5, 15, 17] perform a non-constant overhead operation while rendering, to address grid artefacts. These methods place patches of precomputed shape on the texture at run-time, according to a run-time computation, but require a fixed patch map whose boundaries cannot be overlaid with a patch. For example, in the offline step of [17], a fixed repeating patch map is created, along with various interchangeable patches which fit the patch map boundaries (Figure 5). The texture can then be sampled independently online with a pseudo-random number generator at each repeating patch map. However, none of these resolve the local adjacency constraint for patches overlapping repeatable tile boundaries.

Methods which use a statistical shape model for the texture [5] are able to outperform rendering speeds and quality of exemplar-based parallel synthesis, at the expense of relying on additional user input to model the texture. As this is no longer automated example-based texture synthesis, such methods are not included in Figure 2.

A visual comparison of methods which precompute tiles and select placement during rendering is shown for: Wang tiles (Figure 4), fixed map patches of [17] (Figure 5), and patches without map boundaries presented here (Figure 6). In each figure, the precomputed set of patches or shapes is on the left, with colors corresponding to places of interchangeability.

This paper describes how interchangeable patches can be applied online to repeating tiles without a fixed patch map, and without posing constraints on boundaries and adjacencies. A discussion of the benefits of this approach is in the results section.
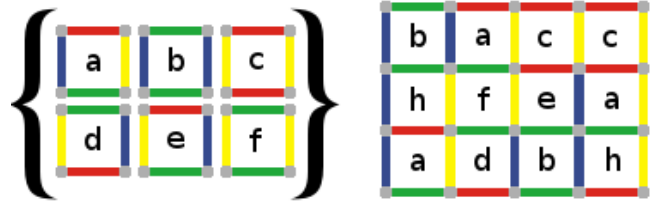

Fig. 4: Precomputed Wang tiles a to f are represented on the left, and a synthesised image is on the right
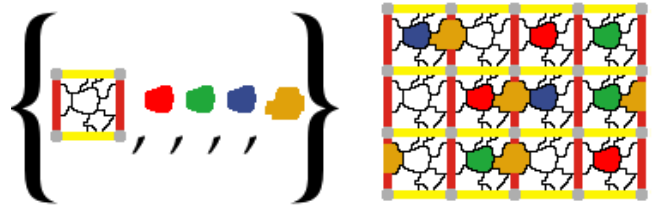

Fig. 5: Method of Vanhoey et al. [17]. Precomputed fixed patch map and patches for content exchange on the left, and a synthesised image on the right
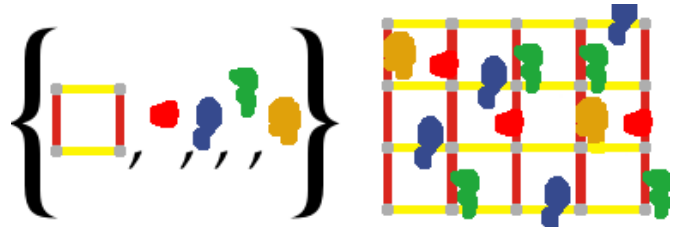

Fig. 6: Our method. The precomputed tile and interchangeable patches are on the left, and a synthesised image is on the right

While offering this enhancement over the state-of-art, our method maintains the benefits of parallel non-constant-time tiling: texture quality depends only on the quality of a pre-processing step and available GPU space. Memory and load on a GPU are addressed to show that even complex textures can fit into limited GPU memory, and the non-constant runtime overhead is only a light logic operation.

3 Patch-based Texture Synthesis without Spatial Dependency

The algorithm described in this paper is divided into two steps: preprocessing and rendering, see Figure 8.
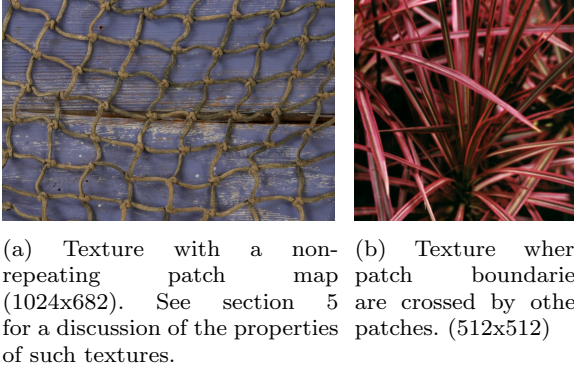
(a) Texture with a non-repeating patch map (1024x682). See section 5 for a discussion of the properties of such textures.

(b) Texture where patch boundaries are crossed by other patches. (512x512)
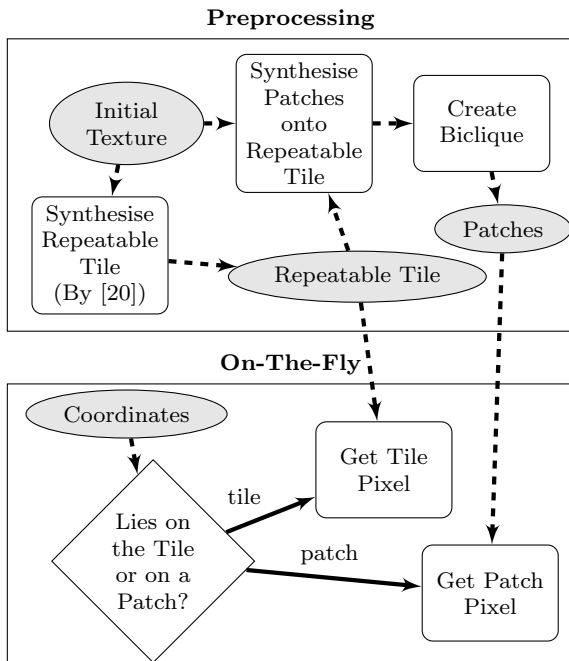
Fig. 7: Irregular textures

Fig. 8: Flowchart overview of our method

The method starts with a simple repeatable tile created from the exemplar texture. Next, interchangeable patches of varying size are precomputed on the repeatable tile. These can reach over the repeatable tile boundary, so they are created such that they form two sets, which mutually do not overlap. During rendering, this non-overlapping criterion permits parallel rendering, while guaranteeing local adjacency because "active" patches cannot overlap.

The tile is made larger than the largest visual repeating element of the texture in the exemplar. For example, in the left image of figure 1, this corresponds to the number of pixels spanned by one apple. By using interchangeable patches of various sizes (from a few pixels to a large portion of the tile), the synthesised texture will contain elements on multiple scales.

The patches are saved, and at runtime are chosen in each tile in a random, repeatable process, without any cached information. A binary map of the patch allows constant-time retrieval of pixel values.

Preprocessing guarantees that patches of adjacent tiles do not overlap, and the online selection guarantees that patches chosen within each specific tile are selected so that they do not overlap. Because of these constraints, every sampled pixel is copied from one of two regions: the repeatable tile, or a selected patch of this tile or the neighbouring tile. At runtime, pixel lookup is performed based on a simple logical operation which makes this decision, and the selected pixel is retrieved from the input texture.

4 Algorithm Implementation

The algorithm is composed of an offline preprocessing step which creates the texture map representation, and an online on-the-fly algorithm called for each requested pixel coordinate (see Figure 8). The texture map consists of a repeatable tile, and patches divided into two sets with offset vectors on their specific tile (but not locations in the final output texture). (See Figure 6)

4.1 Preprocessing

From an input texture, we create

- a repeatable tile
- difference vectors for each patch, denoting its location in the base texture, and in the repeatable tile
- a 2D binary array containing the shape of each interchangeable patch
- a binary matrix for each of two sets of patches with the information whether any given pair overlaps

First, using Image Quilting [2], we synthesise the "repeatable tile" from the exemplar (Figure 9).

(a) repeatable tile (300x300px)

(b) a large patch (91x112px)
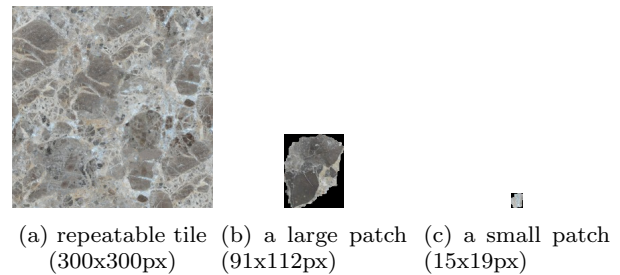
(c) a small patch (15x19px)

Fig. 9: Sample repeatable tile and patches for a given texture

The input texture is used as the patch source. If this input texture does not fit into GPU memory, it may be desirable to render a smaller base texture from which patches will be copied, by [20]. Patches are not stored explicitly, but are indexed from this base texture using the difference vector. Therefore, each pixel of a patch takes 1 bit of memory, instead of a minimum 3 bytes for a naïve RGB pixel representation.

Next, we generate candidate patches by associating random pixel locations between the repeatable tile and the base texture, and by executing GridCut [7] to find the optimal cut. Patches of various sizes are generated by weighting the cuts by a Gaussian bell curve of varying width. The cut is allowed to overflow over borders of the repeatable tile, but not the source tile.

As in previous work, the cut cost is Euclidian distance in CIELab colour space [23] of pixels in the original texture ("base"), and the repeatable tile ("tile"). For each potential patch, we find the maximum pixel cut cost along the boundary, and choose a predefined number of patches ($P = 100$ to 1000) with smallest maximum cut cost. This removes poorly matched patches.

The following step, involving a rhombus and pseudo-biclique, ensures that when patches are selected in adjacent tiles, they will not interact by potentially overlapping.

The output space is divided into tiles A and B, and there are two sets of precomputed interchangeable patches, one for each. The same repeatable tile is used for A and B, to make the texture map compact, but the patch sets differ, to allow greater variability. These patches can lie on the boundary between A and B, but must be entirely within the rhombus around the region they lie in (Figure 10). It is important that patches do not overlap in the output texture, because the region simultaneously covered by multiple patches is not guaranteed to fit. The patches which lie over the tile boundaries ensure that there is no straight repeating boundary in the output texture, and the bounding rhombus assures independence between patches "active" in adjacent tiles.

The patches are divided into a pseudo-biclique such that all patches in set A never overlap with any patch in set B (Figure 11). Using the P patches of varying size (Figure 9), a graph is constructed where each patch is a node and each edge is a "does-not-overlap" relationship (Figure 11). This division is done heuristically, using algorithm 1. Note: if edges are made to represent an "overlaps" relationship instead, this pseudo-biclique becomes the union of two disjoint graphs.

The selected patches are then saved in a binary array, along with the following variables: width, height, top left corner location in the base texture, and top left
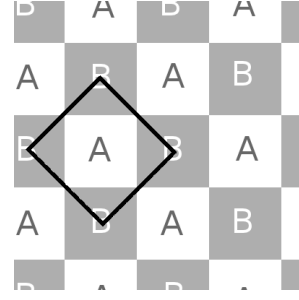

Fig. 10: The output space alignment. The black rhombus represents the boundary that patches in A cannot overlap
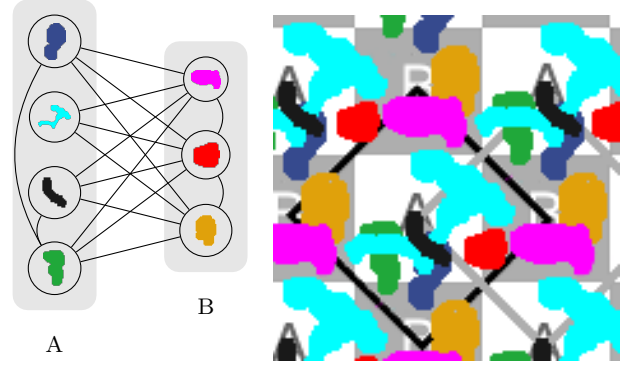

Fig. 11: A pseudo-biclique. Every edge between patches represents a "does not overlap" relationship, and edges between patches in either group are allowed. On the right, corresponding patch positions are shown.

corner location in the repeatable tile. Square subsets of the matrix of overlaps are saved as well, one for the overlaps among patches in A and a second for B.

4.2 On-the-fly Sampling

Given a single (x, y) coordinate, determine which tile it lies in (t_x, t_y) by rounding to the nearest tile, and its location in the tile (p_x, p_y) by modulo. It is then determined whether the desired pixel is on an A tile or a B tile, by whether $t_x + t_y$ is odd or even.

For each tile type, there is a set of precomputed patches P_A and P_B. For each patch ρ in each set, there is a pseudo-random function $r(a, b)$ which lies in the binary domain. For example, our implementation uses the following function:

$$r_\rho(a, b) = mod((\alpha_\rho + a + cos(b))^2 + (\beta_\rho + b + sin(a))^2, 1) < \eta \tag{1}$$

where α_ρ and β_ρ are initialization parameters of the function, specific for each patch ρ. This binary Per-

input : square binary matrix M, where true at (a, b)
               represents that patches a and b do not overlap
output: subset of patches, divided into a
               pseudo-biclique
lists A and B are initialized with the row and column
indexes of a random true point in M
while *sum of lengths of A and B < P* **do**
 a := index of randomly selected row of M, such
 that all intersections with elements in B are true
 if *a is not empty* **then**
  add a to A
 end
 b := index of randomly selected column of M, such
 that all intersections with elements in A are true
 if *b is not empty* **then**
  add b to B
 end
end
Algorithm 1: pseudo-biclique graph division algorithm

lin noise function was chosen because it can be executed efficiently on a CPU and GPU, and it passes the Diehard battery of randomness tests[1]. The parameter $\eta \in [0, 1]$ varies incidence (in our implementation, we set $\eta = 10/|\text{patches}|$.

input : square binary matrix, where true at (a, b)
               represents that patches a and b overlap
output: subset of patches, such that there is no
               overlap
while *the matrix contains at least one true* **do**
 find first row with most true values;
 remove this row, and the same column;
end
Algorithm 2: Deterministic creation of non-overlapping patch subset

$r_\rho(t_x, t_y)$ is evaluated for each patch in the appropriate set. For "active" patches, those where $r_\rho(t_x, t_y)$ is true, the precomputed binary overlap matrix is used to find overlaps between them. Algorithm 2, which is deterministic, is then used to eliminate overlaps. This creates a small non-constant overhead, which is at most linear, but always terminates in very few iterations. Because patches have been divided during offline preprocessing, this operation does not need to consider more than a few potential overlaps, each requiring one clock cycle.

This creates a subset of patches on this tile, called the "active subset". For each, the 2D precomputed binary map of the patch is used to find whether (t_x, t_y) is inside. If the point (t_x, t_y) is inside the patch, the associated pixel is retrieved from the synthesised base texture. This operation is a trivial array lookup in the

appropriate saved patch, since patches are not saved as polygons.

If the point is not inside the patch, the nearest edge is found, and the procedure repeated for the adjacent tile. Note that, thanks to the rhombus-shaped boundary for patches overlaying the boundary between A and B, a pixel can only be affected by interchangeable patches from the adjacent tile which is nearest (Figure 10). If the point is found to be in one of the non-overlapping patches in the adjacent tile, the associated pixel is retrieved from the synthesised base texture.

If the point does not lie in a patch chosen in this tile, and does not lie in a patch chosen in the adjacent tile, we retrieve the pixel from the tile itself.

4.3 Complexity, Memory, and Quality

The computational complexity of the live sampling is near-constant, thanks to the structure in which precomputed information is stored. The process for each pixel is to find which patches are active, then to find the active subset, and retrieve the pixel. Deciding which patches are active is a constant time operation, choosing the active patch subset is at most quadratic in the number of patches, and retrieving the relevant pixel is also constant. As with other tiling methods, memory consumption is completely independent on the number of sampled pixels and the size of the output texture.

Memory consumption is determined by parameters of the preprocessing step, allowing fine tuning to best balance the tradeoff between quality and use of available memory. This was determined to be in the range of tens of kilobytes (for simple textures such as Figure 12a), to 1MB (for complex textures such as Figure 12b). If the first texture uses 600 patches of 5x5 to 86x86 (2400 bytes for difference vectors and 47kB of binary maps) pixels from a texture map of 96x96 (27kB), and a repeatable tile of 96x96 (27kB), the texture map totals 103kB. Note that memory consumption is a factor of four of the texture map, which has been approximately true for all included textures. For larger textures, the total memory footprint will always be determined by these four factors, and each of them can be tuned for the specific application.

The computational complexity of the preprocessing step is comparable to sequential patch-based texture synthesis methods, but the contribution of this work is the texture map compression and on-the-fly synthesis. In practice, the preprocessing can even be done semi-automatically, allowing the user to manually choose patches which are visually satisfactory. The quality of the synthesised texture can be made arbitrarily tuned

---

[1] http://stat.fsu.edu/~geo/diehard.html
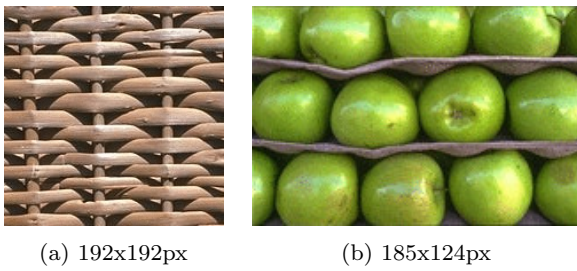
(a) 192x192px   (b) 185x124px

Fig. 12: Input textures

at the scale of patches, and patches can be chosen to have any size. Therefore, by definition, the runtime algorithm of our method can theoretically synthesise a texture of the same quality as any offline patch-based algorithm, repeatably, in parallel, and as fast as other tiling methods. This only depends on the quality of the preprocessing selection. Note that our implementation and results are of a fully automated algorithm, to allow a fair comparison to results published elsewhere.

On the GPU, pixel values are stored in DRAM and cached in texture memory, and all other variables can be optimized to fit into limited L1 processor shared memory. The SIMD model of the GPU allows each multiprocessor to evaluate equation 1 for multiple pixels, and the quadratic selection operation can be performed to the earliest stopping among pixels sharing a multiprocessor. The pixel coordinates in the repeating tile and base texture are returned. Since both images can fit into the texture cache, non-local pixel value retrieval will happen quickly, without reading DRAM memory.

5 Results

Our method produces textures whose quality is not dependent on the runtime computational complexity, but on the quality of the preprocessing step. Therefore, at equal memory footprint, our runtime performance is comparable to simple tiling methods (repeating precomputed tiles, as in [6], [1], [18], and [22]), but the texture quality is comparable with patch-based iterative approaches.

In our experiments, precomputing was set to chose the 1000 best patch interchanges found over a 4 hour period, comparing tens of millions patch interchanges at different scales. For the textures used here, these settings proved satisfactory. Out of these, 300 patches were used in each tile type, and 400 patches were discarded, as discussed in Section 4.1. These amounts have been selected because of memory constraints, because a long search improves the patch quality, and because this many patches provide ample variation in the rendered

texture. The upper bound on possible distinct rendered tiles is $300!^2$, but because there are up to 75% overlaps within each group, this is reduced by a few orders of magnitude.

The speed of the sampling process itself compares favourably with current approaches, despite the overhead to be calculated at every pixel. Speeds are reported on a single core of a 3GHz Xeon with 667 MHz DDR2 RAM. This overhead is tuned by changing η, which was set to $\eta = 0.1$. Because the time required to calculate a single pixel is constant, the algorithm scales linearly in the number of sampled pixels, and is parallelizable in a straightforward manner with speedup proportional to the number of cores.

Our method has multiple applications: ray tracing, rendering from bundled texture maps, or creating non-repeating patches, such as for ceramic tile-printing.

For irregular textures, synthesis requires handling complex properties, such as layering and overlapping, which are not handled automatically by optimal seam selection algorithms. For these, our method allows manual selection of an appropriate repeating tile and patches. Multiple repeating tiles can be synthesised, and the best one is chosen by an expert. Patches are prepared offline, so they can be shown to an expert user, who determines if they make a believable substitution, and selects the best. Synthesis results in figure 13 show how human intervention can improve synthesis quality, while maintaining the storage and run-time speed advantages of our method. Interestingly, by allowing patches to form assume locally optimal shape at numerous scales, interchanged patches often contain visual or semantic features of the example texture.

While [17] works well for regular and stochastic textures, repeating a fixed patch map across an image cannot capture certain irregular textures. Certain irregular textures cannot be faithfully replicated by simply repeating patches of a given shape, no matter what the shape is (Figure 7). Our method does not restrict patches to replace contents only within precomputed boundaries, instead allowing the boundaries themselves to be replaced by other patches, thanks to a patch biclique division. Section 5 contains a deeper discussion of the limitations for certain irregular textures.

The fishing net in figure 13a changes orientation, so patches replacing strings won't align with the wood texture. However, If patch boundaries lie on strings, the underlying wood texture will not align. In figure 13b, if patches are chosen to contain parts of leaves rather than leaves, faithful reproduction won't be guaranteed.

Note that all other textures in this paper have been created without manual intervention.
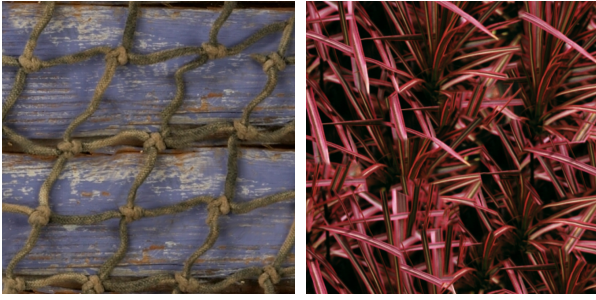
(a) High-level properties of textures can be handled by selecting a repeating tile and patches which satisfy the properties.

(b) Overlapping can be interactively modelled with our technique by manually appropriately selecting patches during preprocessing.
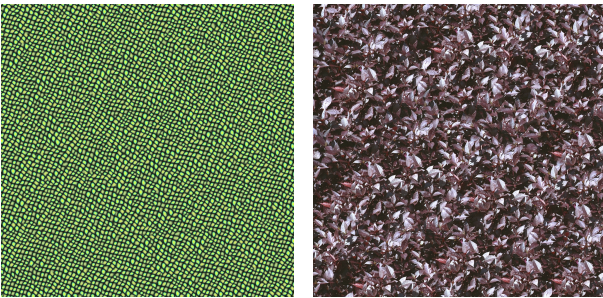
(c) exemplar size 64x64

(d) exemplar size 512x512

Fig. 13: Synthesis results for irregular textures

5.1 Independent Pixel sampling for Ray Tracing

In various applications, a crucial property of texture synthesis algorithms is that the error produced when sampling distant pixels will not be constructive, but will have the same properties as randomly selecting pixels. This is referred to as the white noise property, and is particularly desirable with ray tracing. See results in figure 3. We benchmarked the algorithm's speed by randomly sampling pixels. Across the different textures we have tested, the time required to sample one pixel did not significantly vary, because the key parameters (number of patches, tile size, patch size) were set similarly for all textures. On the tested platform, the time required to sample one pixel averages 60 microseconds, with little variation. Out of this, 45 microseconds are required for pseudo-random binary sampling. Testing various random access scenarios for different textures has no effect on the retrieval time.

Unlike [21], the algorithm does not place a spatial dependency on sampled pixels, so their performance cannot be compared in practice. Any algorithm which places dependency on sampled pixels would rely on cache, making it slower for larger output textures, so that sampling the texture millions of pixels apart would

give our method an advantage with predictable results. Therefore, such a test was not performed in practice.

5.2 Patch sampling

If pixels forming a rectangle are calculated independently, the pseudo-random binary sampling and patch selection is performed unnecessarily. As the largest part of time is spend precomputing which patches are used within a specific tile, performing sampling for adjacent pixels is significantly sped up when the tile properties are already known. Here, simply retrieving the information which patch the pixel is in and returning the appropriate pixel is even faster. This allows us to synthesise a continuous texture of 512×512 pixels in the order of tens of milliseconds, while our naïve pixel-based approach takes over a second.

Such an implementation can be beneficial when entire neighboughoods are required for further processing, such as in filtering.

Figure 3 compares the quality of a texture synthesised with Wang tiles [1], with our method. Note the diamond shaped artefacts, which our method inherently avoids.

Figures 1, 14, and 15 show textures rendered from the inputs in Figures 9a and 12. Figure 1 demonstrates that our method doesn't create constructive repetitions at scales far larger than the input texture, but instead displays properties of white noise necessary for certain applications. Flaws in images generated by our method are seams, visible when the texel is large, and discontinuous objects (Figure 14). This problem is inherent to patch-based methods, as can be seen in the results of the baseline method (Figure 15). Both issues can be mitigated by manual selection of the tile and patches when precomputing.

If adjacent pixels are rendered naïvely, the speedup is be linear. However, if congruous sections are retrieved simultaneously in each thread, the algorithm can be sped up further, because the decision process selecting patches need only be executed once.

6 Conclusion and Future work

We have presented a method with the benefits of current parallel texture synthesis algorithms, allowing texture synthesis in real-time environments from a minimal texture map. By sampling every pixel independently, parallel processing can be exploited for a synthesised texture of arbitrary size, while avoiding repetition along lines or rectangles to avoid visible seams. Our method makes it possible to perform exemplar-based texture
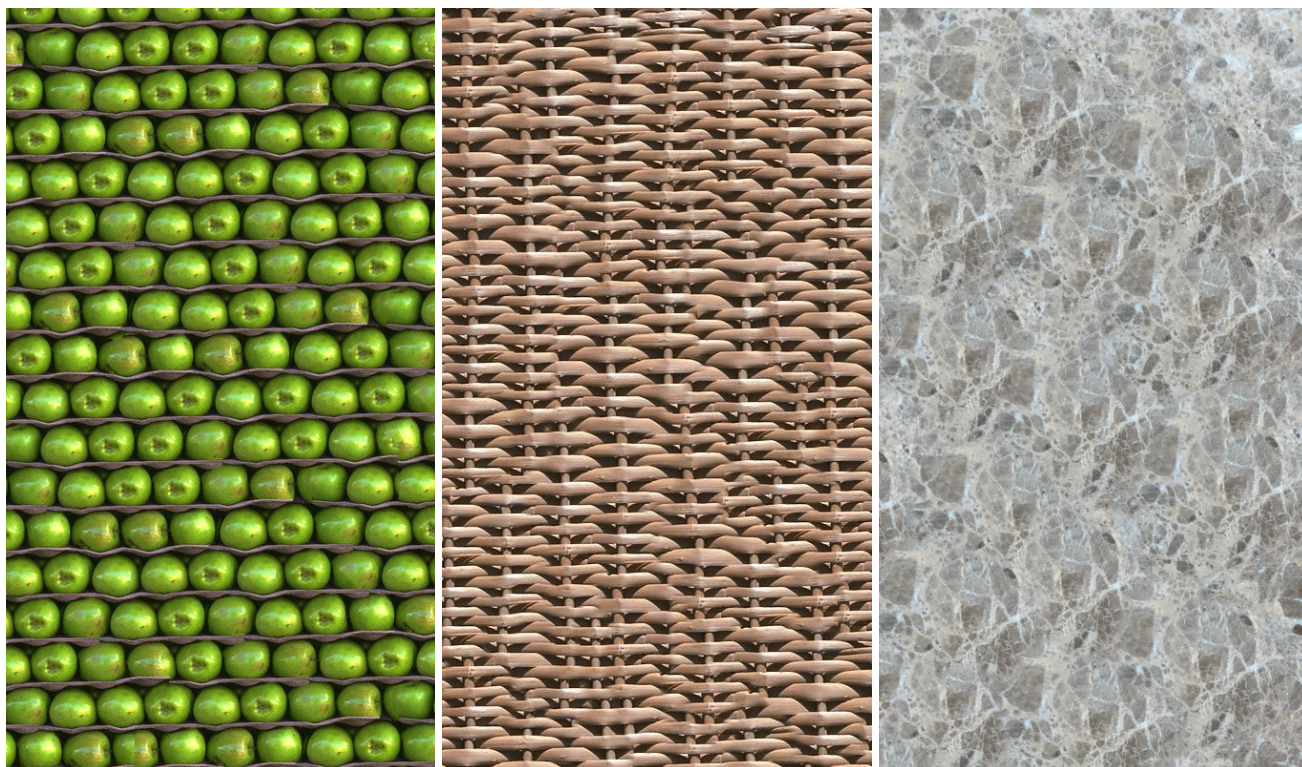
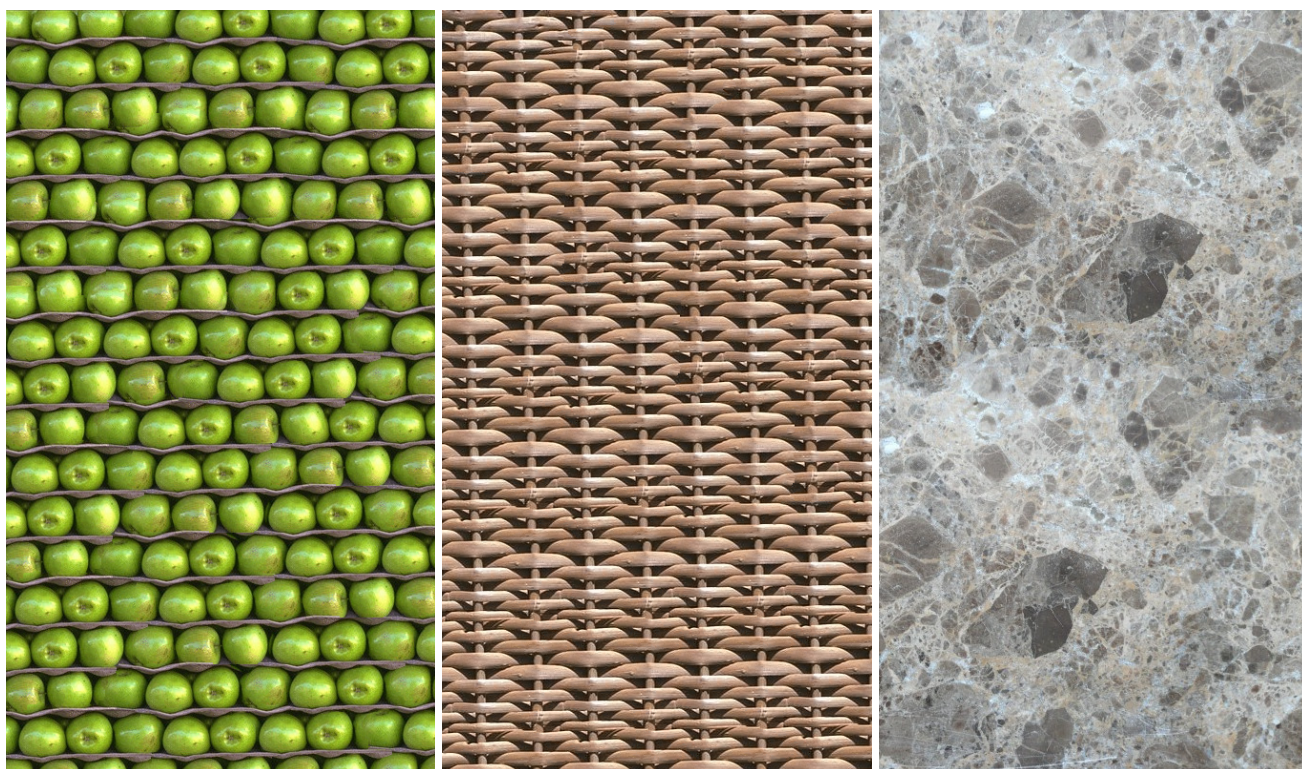Fig. 14: Output textures from our algorithm (450x800px)

Fig. 15: Output textures using Image Quilting [2] (450x800px)

synthesis of arbitrary size in times comparable with much simpler image retrieval operations. Our results are of the same quality as sequential patch-based synthesis, while significantly reducing retrieval time.

The benefits of this method over previous parallel patch-replacement [17] is two-fold: preomputed patches are not limited to lie on a precomputed fixed patch map, and rendered patches can lie over the boundaries of other precomputed patches. This makes the method suitable even to complex irregular textures.

A major drawback of the method is the unavoided repetition of corners of the repeatable tile. Careful selection of patches would allow these corners to be overlapped by patches as well, and future work could investigate this. Thanks to the read-only access of precomputed texture information, our method will make it possible to efficiently utilise GPU hardware to improve rendering times, which is also future work.

In future work, high-level ideas from this work will make it possible to create replaceable patches which do not follow any repeating grid. Patch sets are not inherently limited to two groups. The non-overlapping biclique introduced here is not limited to patches which follow the checkerboard pattern of precomputed interchange locations 10, but can follow a random Wang tile pattern. This will further combine benefits of the two techniques.

It is clear that with a single underlying tile, repetitions in areas not covered by patches will be visible. Future work could combine our technique with Wang tiles to deal with both this issue, and the issue of repeating edges in Wang tiles.

Thanks to the inherent parallelism, many further applications can be explored. This method will great benefit for texture compression, bundling of preprocessed textures with graphical design packages, virtual environment platforms, video games, rendering engines, and mobile applications.

References

1. Cohen, M.F., Shade, J., Hiller, S., Deussen, O.: Wang Tiles for image and texture generation. ACM Transactions on Graphics **22**(3), 287 (2003)
2. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 341–346. ACM (2001)
3. Efros, A.A., Leung, T.K.: Texture synthesis by non-parametric sampling. In: Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, vol. 2, pp. 1033–1038. IEEE (1999)
4. Ephanov, A., Coleman, C.: Virtual texture: A large area raster resource for the gpu. In: The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC), vol. 2006. NTSA (2006)
5. Gilet, G., Dischler, J.M., Ghazanfarpour, D.: Multi-scale assemblage for procedural texturing. Computer Graphics Forum **31**(7 PART1), 2117–2126 (2012). DOI 10.1111/j.1467-8659.2012.03204.x
6. Grunbaum, B., Shephard, G.C.: Tilings and patterns (book). W.H. Freeman & Company (1986)
7. Jamriska, O., Sykora, D., Hornung, A.: Cache-efficient graph cuts on structured grids. In: Computer Vision and Pattern Recognition (CVPR), 2012, pp. 3673–3680. IEEE (2012)
8. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. In: ACM Transactions on Graphics (TOG), vol. 24, pp. 795–802. ACM (2005)
9. Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: image and video synthesis using graph cuts. In: ACM Transactions on Graphics (TOG), vol. 22, pp. 277–286. ACM (2003)
10. Lagae, A., Dutré, P.: An alternative for wang tiles: colored edges versus colored corners. ACM Transactions on Graphics (TOG) **25**(4), 1442–1459 (2006)
11. Lasram, A., Lefebvre, S.: Parallel patchbased texture synthesis. High Performance Graphics (2012)
12. Lefebvre, S., Hoppe, H.: Parallel controllable texture synthesis. In: ACM Transactions on Graphics (TOG), vol. 24, pp. 777–786. ACM (2005)
13. Neyret, F., Cani, M.P.: Pattern-based texturing revisited. In: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pp. 235–242. ACM Press/Addison-Wesley Publishing Co. (1999)
14. Obert, J., van Waveren, J., Sellers, G.: Virtual texturing in software and hardware. In: ACM SIGGRAPH 2012 Posters, p. 5. ACM (2012)
15. Praun, E., Finkelstein, A., Hoppe, H.: Lapped textures. Proceedings of the 27th annual conference on Computer graphics and interactive techniques SIGGRAPH 00 (1), 465–470 (2000)
16. Taibo, J., Seoane, A., Hernández, L.: Dynamic virtual textures. Journal of WSCG **17**(1-3), 25–32 (2009)
17. Vanhoey, K., Sauvage, B., Larue, F., Dischler, J.M.: On-the-fly multi-scale infinite texturing from example. ACM Transactions on Graphics (TOG) **32**(6), 208 (2013)
18. Wei, L.Y.: Tile-based texture mapping on graphics hardware. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, pp. 55–63. ACM (2004)
19. Wei, L.Y., Lefebvre, S., Kwatra, V., Turk, G., et al.: State of the art in example-based texture synthesis. In: Eurographics 2009, State of the Art Report, EG-STAR, pp. 93–117 (2009)
20. Wei, L.Y., Levoy, M.: Fast texture synthesis using tree-structured vector quantization. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 479–488 (2000)
21. Wei, L.Y., Levoy, M.: Order-independent texture synthesis. Tech. rep., TR 2002 (2002)
22. Xue, F., Zhang, Y.S., Jiang, J.L., Hu, M., Wu, X.D., Wang, R.G.: Real-time texture synthesis using s-tile set. Journal of Computer Science and Technology **22**(4), 590–596 (2007)
23. Zhang, X., Wandell, B.A., et al.: A spatial extension of cielab for digital color image reproduction. In: SID international symposium digest of technical papers, vol. 27, pp. 731–734 (1996)