# Cut-primed smart copying

Alexander Neubeck, Alexey Zalesny and Luc van Gool
Swiss Federal Institute of Technology
Zurich, Switzerland
Email: {aneubeck,zalesny,vangool}@vision.ee.ethz.ch

***Abstract*— Texture synthesis through so-called 'smart copying' requires a seamless meshing of texture subparts. Current systems first select subparts that seem to globally fit well and then optimise the cut between them on a rather local scale. This order of first selecting patches and then caring about the seamless meshing reduces one's leeway in the choice of seamless cuts to a small zone of overlap between the patches. Therefore, even in the latest and smartest of smart copying approaches seams still tend to show up in the results. Here we present an approach that first looks for promising cuts, and uses these as the point of departure. It is shown that even a simple criterion for the quality of seams already supports high-quality smart copying and texture tessellation.**

*Index Terms* – texture synthesis, smart copying, seamless cuts, texture tessellation.

## I. Introduction

Texture synthesis has made large advances over the last years. An interesting strand is so-called 'smart copying'. It goes back to the work of De Bonet [2], who started from an original texture image – i.e. a texture sample – and rearranged the input data in a coarse-to-fine manner and in keeping with the original filter response distributions. Later approaches started to extend a kernel of original texture by appending other subparts. Originally, this was done pixel by pixel, as in the seminal work by Efros and Leung [3] and the accelerated version of Wei and Levoy [7]. Later, Efros and Freeman [4] generalised this approach to extensions with complete texture patches at a time, a process they called 'image quilting'. Implementing such faster advancing front stands to reason, as even a pixel-by-pixel extension often leads to the verbatim copying of complete subparts within the original texture. After choosing an appropriate patch to add, Efros and Freeman then looked for a least visible cut in the overlap region between the currently synthesised texture and the newly added patch. Very recently, a similar approach has been proposed by Kwatra *et al.* [6], who allow more flexibility in the choice of good cuts. Patches can be more or less piled up, and pieces from several can be combined at places where the original seams still show up. This allows some additional editing through human intervention.

In summary, these smart copying approaches make a collage of subparts of texture samples that mimics these originals. Simply copying and pasting together will not work, as clear seams will show up. Hence, people select subparts in specific orders, so that they go together well. Moreover, subparts are typically given some overlap, and a local cut between them is defined in such a manner that it is minimally visible. As the area in which the cut is supposed to lie tends to be rather small, the leeway in getting seamless transitions is restricted and boundaries may remain visible. Here we consider a strategy that reverses the order of steps, and first looks for cuts in the texture that support nearly invisible seams. Then, subparts can be defined on the basis of these cuts, which can be designed more freely and will therefore better support seamless knitting.

Apart from smart copying for the synthesis of more of a desired texture, the cut optimisation scheme can also be used to create periodic tessellations of seamless texture as in a wallpaper design, but also in the synthesis of textures that are to be wrapped around closed surfaces. This latter application resembles the Escherisation problem [5] where the goal is to approximate the shape of a pattern so that it can be repeated periodically without leaving holes. A first difference is that we do not only want a seamless periodicity in terms of the tile shapes that are being repeated, but also in terms of their texture. In the case of a wallpaper design, the dimensions of a period may also not be specified beforehand.

Sections II, III, IV describe how the cut selection is made in different applications. Then, section V shows results that have been obtained, both for texture synthesis through this novel type of smart copying, and for the creation of seamless, periodic wallpaper tessellations. Section VI concludes the paper.

## II. 1D-tessellation

For simplicity, we start with the description of a 1D-tessellation where we select a tile and tessellate it along a single axis. Assuming this axis to be nearly horizontal, we select a tile by finding cuts on the left and right side in such a way to make them match when translating along a more or less horizontal axis. This means that both cuts have identical shapes, separating pixels belonging to the tile from the rest of the texture. Gluing tiles together along these cuts, a visible seam may appear, as originally not neighbouring pixels are now connected by the cuts. To avoid such visible seam, one can try to find a path of the same shape – henceforth referred to as an auxiliary path – in the original texture, such that the colour patterns on both sides look maximally similar to those around the seam that is being created. By doing so the seam pattern is very similar to a natural part of the texture itself and, hence, will not be conspicuous.

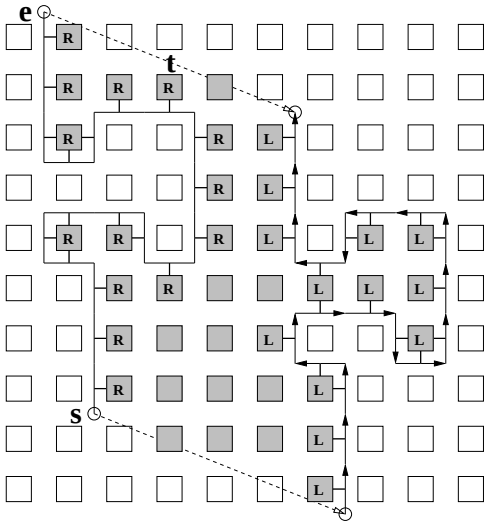


Fig. 1. 1D-tessellation: tile-pixels are grey coloured, the right pixels along the left cut together with the left pixels along the right cut may produce a visible seam.

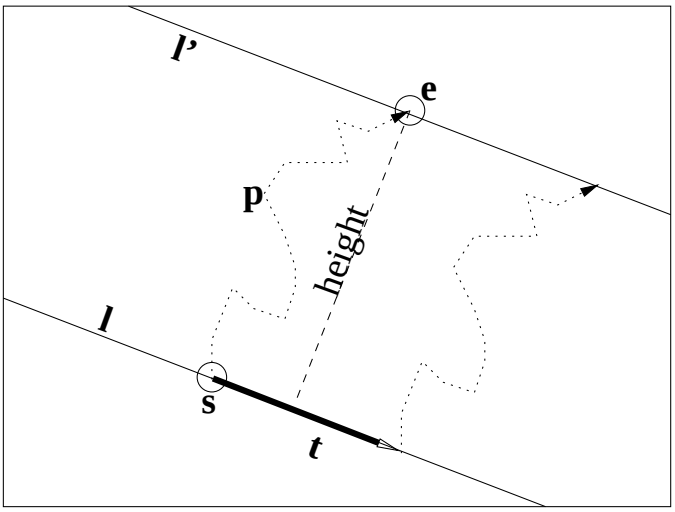


Fig. 2. stripe: all tiles with translation vector $\mathbf{t}$, start and end vertices $\mathbf{s}$, $\mathbf{e}$ on $l$ and $l'$ resp. have the same height.

### A. Definitions

To formalize this approach we introduce a directed graph on top of the texture, where the *vertices* lie between four neighbouring pixels, the *edges* connect the vertically or horizontally neighbouring vertices in both directions, hence lying between two neighbouring pixels. Further we define $L(e)$ (resp. $R(e)$) as the pixel which lies on the left (resp. right) hand side when walking along the directed edge $e$. A sequence of connected edges $(e_1, ..., e_n)$ is a *path* (or equiv. *cut*) $p$. A second path $q$ having the same shape as $p$ will be written as $p^{\rightarrow \mathbf{t}} = \left(e_1^{\rightarrow \mathbf{t}}, ..., e_n^{\rightarrow \mathbf{t}}\right)$, where $\mathbf{t}$ is the translation vector between corresponding positions.

Using this definitions a tile is described by a path $p$ and a period vector $\mathbf{t}$. In fig. 1 one can see the gray coloured region of a tile bordered from the left by $p$, from the right by $p^{\rightarrow \mathbf{t}}$ and from the top and bottom by straight lines. By bending the tile to a cylinder or taking two identical tiles glued together after the appropriate translation, we obtain the so-called *seam*, whose right pixels are from the left cut and whose left pixels are from the right cut. The seam visibility is defined as the minimum distance between the intensity patterns around the seam and all auxiliary paths. Moreover, a preference for larger resulting tiles is built-in through a normalisation based on the tile area:

$$\min_a \sum_{i=1}^{n} \frac{\left|R(e_i) - R(e_i^{\rightarrow \mathbf{a}})\right|^2 + \left|L(e_i^{\rightarrow \mathbf{t}}) - L(e_i^{\rightarrow \mathbf{t}+\mathbf{a}})\right|^2}{area(tile)}. \quad (1)$$

### B. Basic Algorithm

The search for tiles with hidden seams can now be reformulated as an optimisation problem: minimise the visibility in (1) over all possible tiles ($p$, $\mathbf{t}$). That means to minimise over all possible paths $p$, whose number is exponentially growing with the size of the tile. In (1) you can see that all summands are nonnegative and, if the area of the tile is fixed, they only depend on the translation vector $\mathbf{t}$ and the relative positioning $\mathbf{a}$ of the auxiliary path. Thus the translation vector $\mathbf{t}$, the translation of the auxiliary path $\mathbf{a}$ and the start $\mathbf{s}$ and end $\mathbf{e}$ vertex of $p$ are the fixed parameters and the minimisation becomes a shortest path search, which can be solved efficiently by Dijkstra [1]. More precisely, we use Dijkstra with priority queue having a complexity of $O(E \cdot log(V))$, where $E$ is the number of edges and $V$ is the number of vertices. As each vertex has at most 4 outgoing edges and the number of vertices is linear in the number $N$ of pixels, we get $O(N \cdot log(N))$ for the shortest path search. Each of the fixed parameters ($\mathbf{t}$, $\mathbf{a}$, $\mathbf{s}$, $\mathbf{e}$) takes $O(N)$ possible values, for each of the $O(N^4)$ possible combinations one Dijkstra is run, resulting in a total complexity of $O(N^5 \cdot log(N))$, which is even for very small images ($N \approx 10 \times 10$) unfeasible.

As long as the edge weights are static, Dijkstra allows searching for the shortest path between two sets of vertices, without changing the complexity of a single pass. Thus we will determine start and end vertices in such a way that each combination results in the same edge weighting. Assuming $\mathbf{t}$ and $\mathbf{a}$ being fixed, edge weights only depend on the area of the tile, i.e. the product of length and height. These two values are visualized in fig. 2, where the length corresponds to the length $|\mathbf{t}|$ of the translation vector and the height to the point-to-line-distance between $\mathbf{e}$ and $l$, where $l$ is a line through $\mathbf{s}$ with direction $\mathbf{t}$. When moving the end vertex (resp. start vertex) parallel to the line $l$ the height doesn't change. Therefore instead of minimising the shortest path between all combinations of start and end vertices lying on $l$ resp. $l'$, one can search for the shortest path between the two sets of vertices. The total complexity gets reduced by the number of combinations tested in one pass, i.e. saving one order of magnitude ($O(N)$).

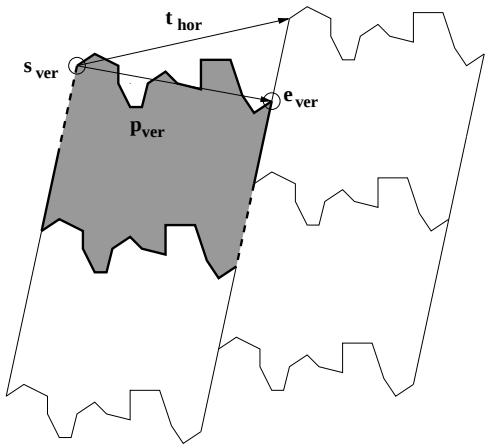


Fig. 3. defect 2D-tessellation: tile-pixels are grey coloured, invisible seams are bold, visible seams are dashed. The defect occurs as the periodicity $\mathbf{t_{hor}}$ of the intermediate texture is ignored.

## III. 2D-TESSELLATION

In this section we extend the 1D-tessellation such that the final tile will tessellate the whole plane along one nearly horizontal and one nearly vertical axis (see fig. 4). The idea is to start with an horizontally periodic texture, which is the result of the 1D-tessellation, and select a tile within this stripe by finding cuts on the upper and lower side in such a way to make them match when translating along the vertical axis. We assume that there are no visible seams in the periodic texture and therefore we don't need the cutting paths of the horizontal tessellation anymore. Except for the orientation there seems to be no difference to the 1D case and the result of such a straight forward application is shown in fig. 3. The horizontal tileability is corrupted at the dashed lines, as the relative positioning $\mathbf{e_{ver}} - \mathbf{s_{ver}}$ of the start and end vertices differs from the periodicity vector $\mathbf{t_{hor}}$. In the next subsection the basic 1D-tessellations will be modified to satisfy this additional constraint. We will use the definitions of the previous section augmented with the indices $hor$ resp. $ver$ to distinguish between horizontal and vertical tessellation.

### A. Modifications of vertical tessellation

To preserve the horizontal tileability the difference vector $\mathbf{e_{ver}} - \mathbf{s_{ver}}$ between start and end vertices of the path $p_{ver}$ must match the horizontal periodicity $\mathbf{t_{hor}}$. As Dijkstra chooses only the best pair out of a set of start and end vertices, it is not guaranteed that our additional constraint is satisfied and therefore we cannot use this feature to speed up the vertical tessellation. Instead of that, we run Dijkstra for each possible start vertex and in return there is only one possible end vertex compensating the loss of speed.

The horizontal periodicity restricts the search space further as most of the paths will lead to the same weight. Given an arbitrary path which satisfies the periodicity constraint, one can cut off the first edge and append it to the last one without changing the total weight. This rule allows placing the start vertices of all possible paths on a single vertical line, i.e. it is sufficient to run Dijkstra for each vertex along this line.

If a shifted path gets self-intersecting the appearing loop can simply be skipped resulting in a shortened and therewith lighter path. This reasoning implies that paths, minimising our visibility criterion, will never be self-intersecting.

```
input:
  periodic texture stripe
  horizontal periodicity vector t_hor

select an arbitrary vertical line V
for all t_ver do
  for all a do
    for all s_ver along V do
      e_ver := s_ver + t_hor
      p_ver := shortest path search
               between s_ver and e_ver
      if   visibility of (p_ver, t_ver)
         < visibility of (p_best, t_best)
        (p_best, t_best) := (p_ver, t_ver)
return (p_best, t_best)
```

The complexity of this algorithm is $O(N^3 \cdot \sqrt{N} \cdot log(N))$.

### B. Modifications of horizontal tessellation

Sequencing the horizontal and vertical tessellation two parameters of the horizontal optimisation influence the vertical task: firstly the horizontal periodicity $\mathbf{t_{hor}}$, which constrains the final tile shape, and secondly the height of the stripe, which restricts the variability of the final tile, i.e. number of start vertices $\mathbf{s_{ver}}$. As we want to run the vertical tessellation on the best results of the horizontal tessellation, the seams of these intermediate tiles must not only be hidden, but also these tiles must be sufficiently high providing a basis for hiding second seams.

Such a compromise between horizontal seam visibility and height can be achieved by enforcing the same variability of start and end vertices in both tessellations, i.e. the number of vertices along a vertical connection between $l$ and $l'$ should be the same as along $l$ and $l'$. As there is only a small number of different heights, one can search through all of them choosing the height best matching the condition.

```
input:
  texture

for all t_hor do
  select l and l'
  for all a do
    p_hor := shortest path search
             between l and l'
    if   visibility of (p_hor, t_hor)
       < visibility of (p_best, t_best)
      (p_best, t_best) := (p_hor, t_hor)
return (p_best, t_best)
```

The complexity of this algorithm is $O(N^3 \cdot log(N))$.

### C. Acceleration

In spite of the previous speed-ups it is still prohibitive to run the tessellation on common texture sizes. Therefore a multiscale approach is chosen, where we scale down several times by a factor of two until a full search is feasible with the proposed algorithms ($N \approx 32 \times 32$). The horizontal and vertical translation vectors are propagated to the next scale and there refined by varying the vectors in a $5 \times 5$ neighbourhood. This procedure reduces the complexity of the finer layers by $O(N)$ because the outermost loops over all translation vectors are replaced by loops over $25$ different vectors.

Also we record the currently shortest path weight over all paths to interrupt Dijkstra as soon as this threshold is passed. Although our experiments have shown that this is a very frequent situation yielding a serious improvement in speed, there is no guarantee in general.

Under the supposition of the texture's near translation invariance it should be sufficient to choose auxiliary paths only within the constructed tile. We had no input texture, where this restriction resulted in worse results.

Combining this ideas with the proposed tessellation algorithms we get an overall complexity of $O(N^2 \cdot log(N) + M^3 \cdot log(M))$, where $M$ is the size of the coarsest pyramid layer.

## IV. Smart copying

So far we described periodic tessellations. By combining different cut shapes in the two directions, non-periodic tessellations can be created. The same principles apply, i.e. one looks for cuts that all generate seams with low visibility. Variability in cut shapes and corresponding seams is imposed through a stochastic selection procedure, that only considers a part of possible cut starting points. This 1D procedure is explained in more detail shortly. In order to get at a 2D non-periodic tessellation, we again follow a 2-stage procedure. First, a 1D stripe is created with a width larger than that of the intended texture. Then, the same selection of multiple cuts is performed in the horizontal direction along the stripe.

The creation of 1D, non-periodic stripes proceeds as follows. Starting from an arbitrary, horizontal stripe of fixed, chosen height, a cut spanning it from top to bottom, is selected that now becomes its right border. This selection is based on the existence of a similar, second cut at an arbitrary position in the original texture so that intensities on both sides of the cuts match well. Then the stripe is extended horizontally to the right of this new, second cut. Note that no auxiliary path is used here, as the search problem would become huge. Although this comes at the expense of flexibility in the choice of cuts, we have more freedom in the positioning of the second cut, which to some degree makes up for this loss. This process is then repeated until a sufficiently wide stripe is ready.

The 2D extension works with horizontal cuts along the stripe. The width of these cuts is shorter than that of the stripe. The process of vertical extension from the first cut follows exactly the same procedure as that of horizontal extension just described.

Fig. 5 shows an original texture sample and a 128-pixel high stripe of texture that has already been synthesised up to some point. In the stripe three cuts are highlighted, as are their corresponding cuts in the texture sample. The seam generated by the middle cut is hardly visible, although it contains texture up to the left and starting from the right for the sample cuts. The next and previous cuts are also indicated in the images. It illustrates that rather large patches can be added in a single stroke, in comparison to the traditional smart copying algorithms, which tend to progress pixel by pixel or, more recently, with small patches.

## V. Results

First, we show several examples of 2D smart copying. Fig. 7 shows original $256 \times 256$ texture samples, and some synthetic $512 \times 512$ results. As can be seen, the examples include quite a diversity of texture types. Some are more stochastic in nature, others quite strongly period, some are quite coarse, others fine-grained. All cases have been dealt with using exactly the same algorithm. In contradistinction, in traditional smart copying algorithms the choice of patch sizes is typically tuned for the texture manually. Also, no additional seam removal steps like feathering (as e.g. applied after cut optimisation by Kwatra et al. [6]) were applied. The images shown here are the direct result of our cut optimisation.

Fig. 6 shows two examples of 2D tessellation. On the left side one sees the original texture sample and the patch used for tessellation. The image in the middle is a result that was obtained with the recent smart copying method proposed by Kwatra *et al.* [6]. The seams are quite visible. The image on the right shows the result of our 2D tessellation scheme. Although the tessellation is more restrictive in its choice of seam shapes – as they have to come in pairs and the cuts in one direction impose the endpoints for those in the other – hardly any seams can be discerned in this case. As a matter of fact, several such tessellations could be found, that all have less visible seams than the state-of-the-art smart copying result in the middle.

## VI. Conclusions and future work

We have proposed an approach to 'smart copying' that starts with looking for opportunities to make seamless cuts, rather than first choosing good patches to attach to the texture and only then optimising the cut between them. This alternative order of steps offers some advantages. If seamless cuts can be formed at all, they will. This is not guaranteed with the usual approaches that confine the search for seamless cuts to a rather small region of overlap between patches. In a sense, these approaches work from the large scale to the small, whereas the propounded approach counts on being allowed to infer global consistency from local consistency. The strong statistical connections that exist among the intensities within textures underpin this strategy.

This paper was mainly intended to launch this idea of performing smart copying by first going after good cuts and achieving this based on surprisingly local information around

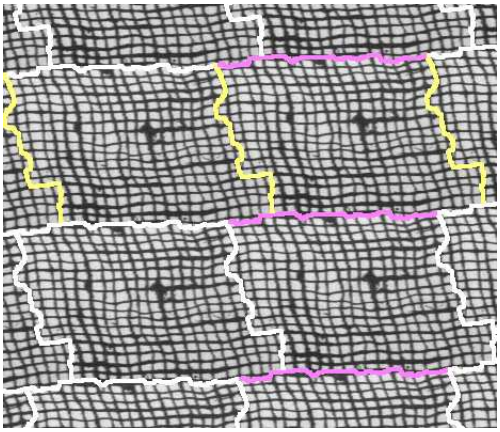

Fig. 4. 2D-tessellation with highlighted seams.

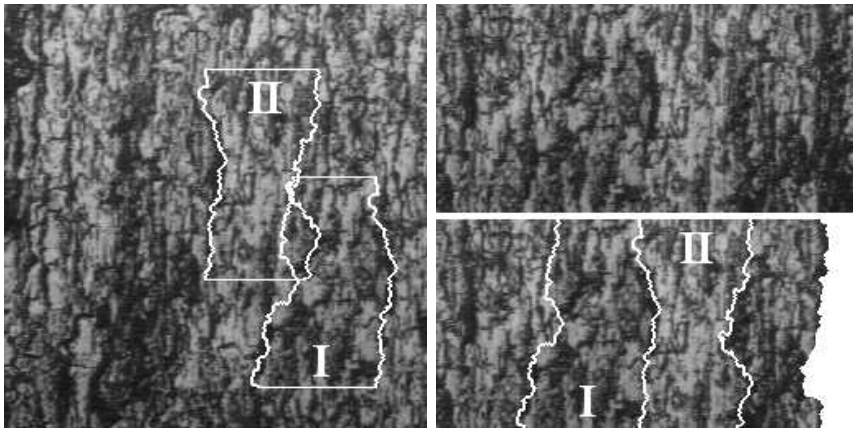


Fig. 5. left: sample with two highlighted patches, top: synthesised stripe, bottom: hidden seams.

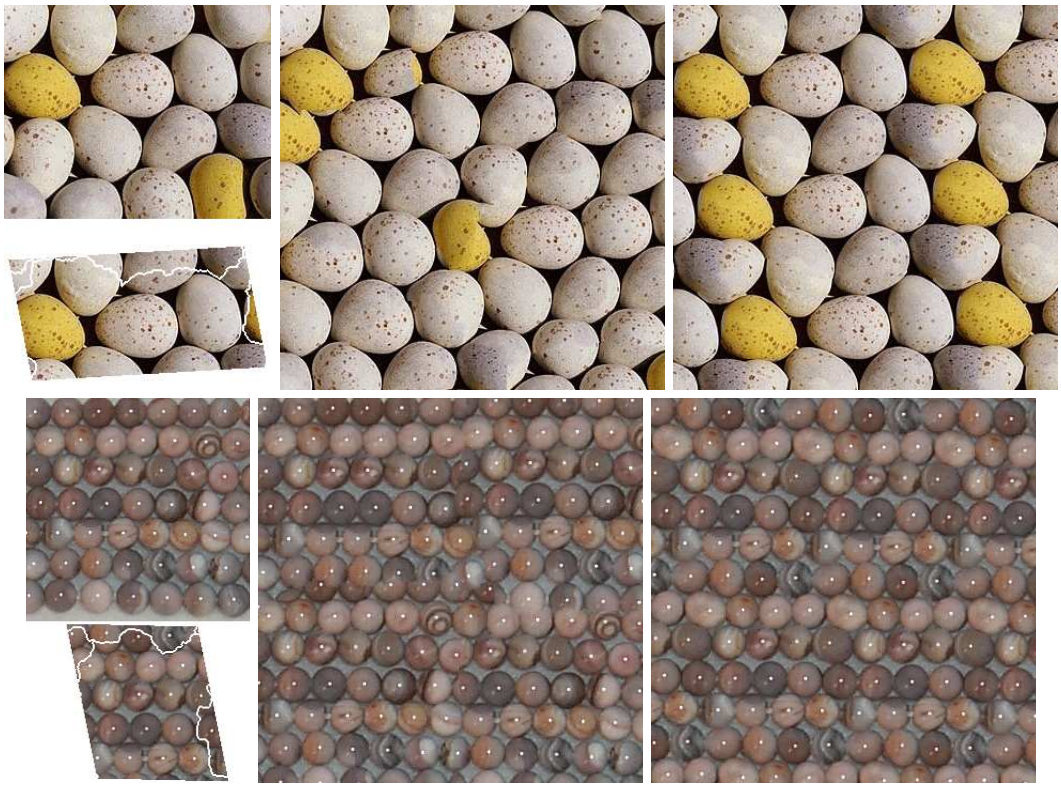

Fig. 6. top: original, bottom: normalised parallelogram with highlighted cuts, middle: cut optimisation by Kwatra, right: proposed tessellation.

the cuts, without any prior patch meshing. This said, a full-fletched 2D smart copying strategy has not been implemented yet. This work is ongoing at the time of writing. Such 2D meshing is computationally more demanding, but also opens additional possibilities to bring variations to the synthesised texture. We would also want to bring in some memory function into the system, in order to avoid the reuse of identical parts close to each other in the synthetic textures. Such repetitions are quite salient and therefore have to be avoided.

**Acknowledgement:** Support by EC project MURALE is gratefully acknowledged.

## REFERENCES

[1] E. W. Dijkstra, A note on two problems in connection with graphs, Numerical Mathematics, vol. 1, pp. 269-271, 1959
[2] J. De Bonet, Multiresolution sampling procedure for analysis and synthesis of texture images, Proc. Siggraph, pp. 361-368, 1997
[3] A. Efros and T. Leung, Texture synthesis by non-parametric sampling, Proc. Int. Conf. on Computer Vision, pp. 1033-1036, 1999
[4] A. Efros and W. Freeman Image quilting for texture synthesis and transfer, Proc. Siggraph, pp. 341-346, 2001
[5] C. S. Kaplan and David H. Salesin, Escherization, Proc. Siggraph, 2000
[6] V. Kwatra, A. Schoedl, I. Essa, G. Turk, and A. Bobick, Graphcut textures: image and video synthesis using graph cuts, Proc. Siggraph, 2003
[7] L.-Y. Wu and M. Levoy, Fast texture synthesis using tree-structured vector quantisation, Proc. Siggraph, pp. 479-488, 2000

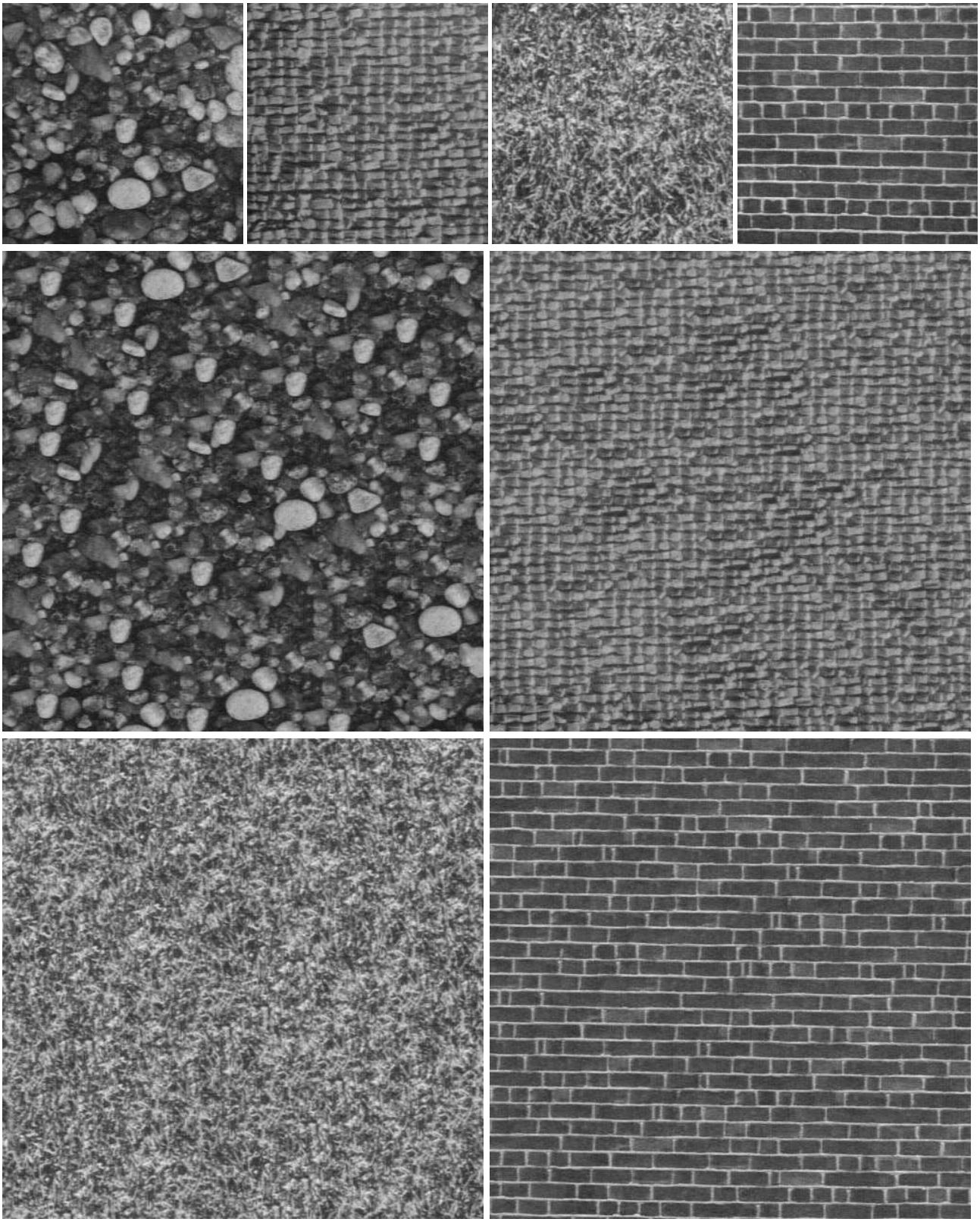

Fig. 7. top: original texture, bottom: 2D smart copying.