Volume 0 (1981), Number 0 pp. 1–13 COMPUTER GRAPHICS *forum*

Just-in-Time Texture Synthesis

Lili Wang [1],Yulong Shi[1],Yi Chen[1] and Voicu Popescu [2]

[1]State Key Laboratory of Virtual Reality Technology and Systems,
School of Computer Science and Engineering, Beihang University, Beijing China
wanglily@buaa.edu.cn
[2]Computer Science, Purdue University, West Lafayette, Indiana, USA
popescu@purdue.edu

Abstract

Texture bombing is a texture synthesis approach that saves memory by stopping short of assembling the output texture from the arrangement of input texture patches; instead, the arrangement is used directly at run time to texture surfaces. However, several problems remain in need of better solutions. One problem is improving texture diversification. A second problem is that mipmapping cannot be used since texel data is not stored explicitly. The lack of an appropriate level-of-detail (LoD) scheme results in severe minification artifacts. We present a just-in-time texturing method that addresses these two problems. Texture diversification is achieved by modeling a texture patch as an umbrella, a versatile hybrid 3-D geometry and texture structure with parameterized appearance. The LoD is adapted continuously with a hierarchical algorithm that acts directly on the arrangement map. Results show that our method can model and render the diversity present in nature with only small texture memory requirements.

Keywords: texture synthesis, texture bombing, diversification

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism-Texture—Color, shading, shadowing, and texture

1. Introduction

Texture mapping is a uniquely powerful method for enhancing surface appearance in interactive computer graphics. Texture synthesis research efforts have produced techniques that construct high resolution textures based on input texture patches and patterns. Unfortunately, the high resolution synthesized texture requires large amounts of texture memory. Texture bombing addresses this challenge by stopping short of assembling the high-resolution texture from the arrangement of patches; instead, the arrangement is used directly at run time to texture surfaces. The texture is synthesized just in time and is never stored explicitly, which brings considerable texture memory savings.

However, several problems related to texture bombing remain in need of better solutions. One is improving texture diversification. Given a small number of input patches, a plausible large texture can only be synthesized if the appearance of the input patches is modulated sufficiently to reflect the diversity present in nature. Moreover, the diversification has to be achieved at run time, during the actual texture mapping. A second problem is that mipmapping cannot be used since texel data is not stored explicitly. The lack of an appropriate level-of-detail (LoD) scheme results in severe minification artifacts.

In this paper we present a just-in-time texture synthesis method that addresses these two problems (please also see the accompanying video). Texture diversification is achieved by modeling a texture patch as an *umbrella*, a versatile hybrid 3-D geometry and texture structure with parameterized appearance. The input patch umbrellas are modified and arranged to synthesize a large, high-resolution, and diverse texture. The input patch is modified substantially by interpolation to new colors and 2-D and 3-D shapes. In Figure 1 the 4 base umbrellas are sufficient to create hundreds of unique modified umbrellas (left), so the synthesized texture (middle) does not suffer from repetitiveness, which would be readily noticeable if the texture were synthesized only from the 4 base umbrellas (right). In Figure 3 umbrellas are mapped to ellipsoids of various curvature which implements

Figure 1: *Just-in-time texture synthesis (left, 9MB) and conventional texture of equivalent resolution (right, 190MB).*

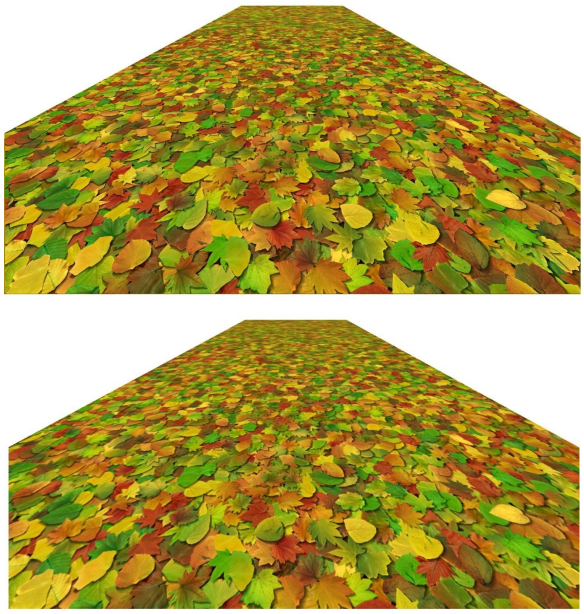

Figure 2: *Just-in-time texture synthesis (top, 9MB) and conventional texture of equivalent resolution (bottom, 190MB).*

3-D shape diversification and adds 3-D detail to the synthesized texture.

For the second problem we propose a hierarchical LoD algorithm for just-in-time texturing that acts directly on the arrangement map. Lower LoD arrangement maps are computed off-line by merging umbrellas and are used at runtime to avoid minification artifacts. A trilinear color interpolation or a geometric morph between consecutive LoDs allows transitioning between LoDs continuously, with good texture stability and clarity. The result is quality similar to that of conventional mipmapping at a fraction of the storage cost (Figure 2). Moreover, our LoD algorithm switches gradually from 3-D to 2-D to only render costly 3-D detail where needed. To the best of our knowledge, our method is the first to provide a seamless transition between 3-D and 2-D surface detail.

Figure 3: *3-D shape diversification (top) and comparison between texture with and without 3-D detail (bottom).*

2. Related Work

A variety of texture synthesis methods have been developed [WLKT09]. Methods can be classified according to the periodicity of the data of the generated texture, which can be regular, such as a brick wall, irregular, such as fallen leaves on the ground, or purely stochastic, such as a rough surface. Regular textures have been modeled procedurally [LP00]. Other methods separate the sample texture into a regular and an irregular component, e.g. by using fractional Fourier analysis [NMMK05], which are then modeled independently, diversified and combined during texture synthesis [LTcL05]. We target the synthesis of irregular textures.

Texture synthesis methods can also be classified as procedural or sample-based methods. Procedural methods use for example turbulence or Perlin noise functions to generate textures that have repetitive patterns or self similarities [Pea85,Per85]. Sample-based methods, like ours, assemble the texture from modified versions of the input patches. A sample-based texture synthesis method needs to address three tasks.

The first task is to extract texture patches from input images, which can be either done manually, or automatically, through random selection of a rectangular window [LLX*01, KSE*03] or with the help of image processing techniques [DMLG02, ZZV*03, WY04, LH06]. The second task is to arrange the extracted texture patches in the output texture

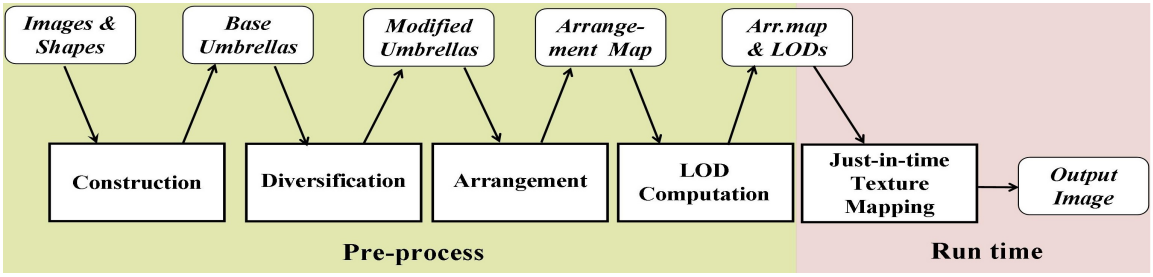

Figure 4: *Overview of the just-in-time texture synthesis pipeline.*

domain. Some textures require an overlapping arrangement, which can be achieved by random placement of patches while controlling patch density [DMLG02, HQXT05]. Other textures require a seamless tiling of patches, achieved using graph cuts [KSE*03, EF01], patch stitching [DLC05], wang tiles [CSHD03], or sparse linear system optimization [PFH00]. For example, wang tiles use square patches whose edges are labeled with colors. A valid tiling requires all edges shared between two tiles to have the same color and it is computed with a stochastic algorithm that tiles a plane non-periodically with a small set of wang tiles [CSHD03]. The arrangement of the patches is either learned from an example [IMIM08, MWT11], random [CSHD03, KCoDL06, TW08], or defined with the help of user input [LN03]. The third task is to diversify the input patches. Diversification methods rely on many-knot spline interpolation [HQXT05], on regular lattice combined with deformation fields [LLH04], on texture meshes inspired from image meshes [DZ06], or on multi-scale descriptors which allow for appearance-space jitter that retains the structure on the input texture patches [RHDG10].

Our method relies on prior work solutions for the first and second tasks, i.e. patch extraction and patch arrangement, and contributes a powerful approach for the third task, patch diversification. In the examples shown in the paper patch extraction is performed manually and the patches are automatically arranged randomly, with uniform density, and in overlapping fashion. However, our method could be used with any patch arrangement method, including wang tiles [CSHD03, Wei04, LD05], lapped textures [PFH00], and based on user input [LN03].

Most texture synthesis approaches compute the new texture off-line, and the synthesized texture is then used at run time like a conventional texture. However, the resulting texture can be large and redundant, as it is obtained by integrating a large number of similar patches. Texture bombing–the idea of saving memory by reusing a few texture patches placed at random locations–was pioneered over thirty years ago [SA79]. The advent of programmable graphics hardware brought renewed interest in the approach [Gla04, LHN05]. Our method takes the texture bombing approach.

In summary, the goal of our work is the development of a texture synthesis method for real time rendering that achieves good diversity of the output texture without requiring considerable texture memory resources. Whereas we can and do rely on prior art methods for patch extraction and arrangement, reaching our goal requires innovation beyond the prior art as follows:

1. *Powerful color, 2-D shape, and 3-D shape patch diversification integrated with the texture bombing approach.* Whereas in prior diversification methods the diversified texture patches were actually converted to texture maps as a preprocess, in our case diversification has to be efficient such that it can be performed just-in-time during texture mapping.

2. *An LoD algorithm for texture bombing that supports arbitrary minification.* Since texture bombing renders directly from patches, mipmapping [Wil83] cannot be used across patches. The lack of an appropriate LoD scheme for texture bombing is a fundamental limitation that precludes its use.

A direct comparison between our method and each of the many relevant prior art techniques is beyond the scope of this paper. We limit the discussion to two recent and highly visible techniques: discrete element textures [MWT11] and structured image hybrids [RHDG10]. Like our method, discrete element textures model texture patches using a mix of geometry and image data. However, the geometry of a discrete element is a simplified 3-D triangle mesh, which is rendered by rendering each of its triangles, and which does not support real time LoD adaptation from 3-D to 2-D or across patches. On the other hand, the umbrella we propose is designed to capture 2-D and 3-D detail, while serving as a new rendering primitive. The entire umbrella is rendered directly with specialized rasterization and shadow algorithms, and it supports real-time LoD adaptation from 3-D to 2-D and across patches. Moreover, the umbrella supports drastic shape and color changes at run time, allowing for example one type of leaf to morph into a completely different type of leaf. Finally, the discrete element work focuses on the arrangement of 3-D patches based on a neighborhood similarity metric and on energy optimization, which is complementary to our work.

The structured image hybrids synthesis technique [RHDG10] extends appearance-space jitter [LH05] to preserve structure. The technique is pixel-based, it is fully auto-

matic, and it generates very convincing hybrids from only a small set of input exemplars. The method is not suitable for the context of texture-bombing as the hybrids cannot be generated on the fly. Moreover, the method achieves excellent fine grain diversification of appearance, but it is ill-suited for interpolating between exemplars with greatly different color. By comparison, our method targets simple patches, whose structure is well captured and preserved by our umbrella geometric structure, and whose color and shape can be varied quickly and automatically by manipulating a very small number of parameters on the fly, as needed for texture bombing.

Our method saves texture memory by taking the texture bombing approach. Texture memory can also be saved by compression at texel level. The main approaches are based on block partitioning [KE02, SR06], on vector quantization [BAC96, TF08], and on wavelets [BIP00, DCH05, STC09]. All of these techniques allow looking up the compressed texture directly (e.g. [DCH05]). Compared to texture compression, our method achieves compact storage while avoiding compression artifacts: powerful diversification allows creating a large texture from only a small number of input patches, which are stored uncompressed.

Several procedural geometric modeling methods target foliage specifically. The methods rely on L-Systems [RSL*02, PTMG08], on probabilistic [DGAG06] algorithms, on diversification of low-count polygonal models [MGGA10], and/or on particle systems [RCS04] to simulate ecosystems and autumn scenery. Compared to these methods, our technique achieves diversification based on examples and not based on rules, and our technique generates a texture defined compactly in a 2-D domain as opposed to a 3-D geometric model which needs to be processed in its expanded form. Structural layering [ACo12] is a recent approach for generating a large number of texture patches from a small number of input examples, demonstrated in the context of leaves. The texture patch is decomposed into layers, i.e. veins, spots, and background, and layers are diversified individually. Compared to our method, structural layering achieves finer grain diversification, e.g. leaves with different spot patterns. The diversification achieved by our method has the advantages of greater 2-D shape variation (i.e. polygon morphing versus coarse 2-D warping), of supporting 3-D shape variation, and of computation efficiency that enables the just-in-time approach. Moreover, our method can be be used with many types of textures and appears to be more general than structural layering which has only been demonstrated in the context of leaves.

Finally, our method captures surface 3-D detail. Previous techniques for modeling and rendering surface 3-D detail include bump mapping [Bli78], horizon mapping [Max88, SC00, HDKS00], displacement mapping [Coo84, KS01], view dependent displacement mapping [WWT*03], parallax mapping [KTI*01], and relief texture mapping [POC05]. In texture synthesis, solid texturing approaches [DHR12] replace the 2-D texture patch with a 3-D particle. Compared to these techniques, our method trades 3-D modeling fidelity for rendering efficiency by mapping an umbrella patch to an ellipsoid, which can be rendered efficiently on the GPU [Gum03]. Moreover, we only render 3-D detail where needed with a smooth transition to 2-D.

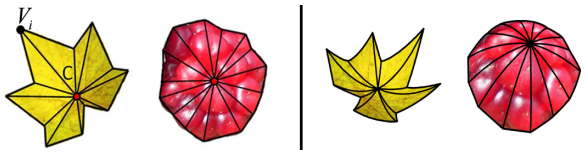

Figure 5: *Base umbrellas (left) and 3-D shape modif. (right).*

3. Just-in-time texture synthesis overview

The texture is synthesized off-line in four major steps (Figure 4). First, a small number of base umbrellas (e.g. 4 in Figure 1) are constructed from input images and shapes containing the desired texture elements. Second, the base umbrellas are diversified to hundreds of unique modified umbrellas by varying color, 2-D shape, and 3-D shape parameters. Third, the modified umbrellas are arranged in the 2-D texture domain. Umbrella construction, diversification, and arrangement are described in Section 4. Fourth, the umbrellas and the arrangement map are fed into an algorithm that computes the LoDs needed to accommodate any minification level (Section 5). The umbrellas, the arrangement map, and the LoDs are then used at run-time to texture surfaces as needed for the current output image (Section 6).

4. Umbrella Texture Patches

We define an umbrella as a texture-mapped 2-D geometric primitive with a central vertex C and peripheral vertices V_i (Figure 5, left). The umbrella need not be convex, but all segments V_iC have to be inside the umbrella. The texture of the base umbrella is derived from input images that contain the desired texture elements. We construct base umbrellas with an interactive editor. The center is chosen at a natural feature convergence point (e.g. for leafs), or as the center of mass of peripheral vertices (e.g. for berries). A base umbrella is created in seconds. Only a few umbrellas are needed (e.g. 4-10), which are then diversified automatically.

In order to modify the color of a base umbrella, its vertices are assigned colors that are used to modulate the texture of the umbrella. The color c_p at a point P inside the umbrella is computed as follows:

$$\begin{aligned} c_p &= c_t + f c_v \\ c_v &= \sum d_i c_i / \sum d_i \end{aligned} \tag{1}$$

where c_t is the color looked up in the base umbrella texture

Figure 6: *Umbrella color diversification.*

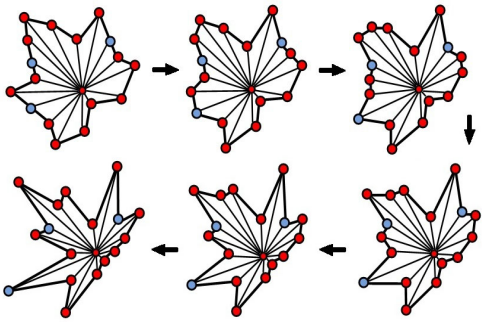

Figure 7: *2-D shape diversification by morphing.*

and c_v is a weighted average of the vertex colors c_i. A weight d_i is defined as an inverse of the distance between vertex i and P, which achieves a mean-value interpolation [Mic03]. The coefficient f controls how much the original texture colors are modified, and can assume negative values (we use random values in the (-1, 1) interval). We set the vertex colors using additional reference images of similar texture elements. In Figure 6, a leaf with different colors (left) is used to set the base umbrella vertex colors (middle) to generate a realistic leaf new leaf (right).

The 2-D shape of a base umbrella is modified by moving peripheral vertices. In Figure 7 the base umbrella has collinear peripheral vertices (blue) which allow creating significantly different leaf shapes. The destination vertex positions can be designed by the user, or can be derived from the shapes of other leaves. We support modeling and diversification of 3-D surface detail by associating an umbrella to a semi-ellipsoid (Figure 5, right). The semi-ellipsoid height is a parameter set during 3-D shape diversification and then tapered by the LoD algorithm as needed for a smooth transition from 3-D to 2-D. The ellipsoid provides a good trade-off between modeling power and rendering cost. The semi-ellipsoid modulates the 3-D shape of the umbrella during texture mapping as described in Section 6.1.

The texture is synthesized by arranging modified umbrellas in the 2-D texture domain. We exemplify our just-in-time texturing method using an overlapping arrangement defined with a regular grid. Modified umbrellas are assigned to grid cells. A grid cell is assigned all umbrellas that intersect it (Figure 8). The umbrellas are stored in back to front order. For the examples shown in this paper the base umbrella color and shape diversification parameter values, as well as the location, rotation, and scale of the modified umbrellas are chosen randomly.

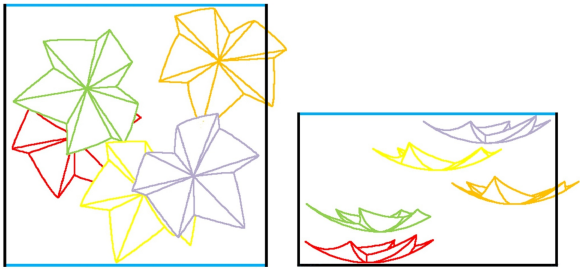

Figure 8: *Top and side views of grid cell umbrellas.*

5. Level of detail pre-processing

Level-of-detail adaption is needed in order to address the following concerns:

- First, expensive 3-D detail should only be rendered where it matters, i.e. close to the eye. This requires switching gradually from 3-D detail to a flat (2-D) surface.

- Second, when umbrellas have a small image footprint, mipmapping individual umbrella textures is not sufficient to avoid minification artifacts, and umbrellas have to be merged. Whereas traditional texture synthesis methods actually compute a large texture and individual patches are merged implicitly through mipmapping, just-in-time texture synthesis requires merging umbrellas explicitly to compute coarser LoDs of the arrangement grid.

- Third, a high-quality stable texturing method requires a continuous adaption of the level of detail, with smooth transitions from one LoD to the next. In the case of mipmapping this is achieved using trilinear interpolation. We are using two approaches for continuous transition between one LoD and the next. The first approach looks up each LoD and blends the resulting two colors, similar to conventional trilinear interpolation. The second method increases texture clarity by relying on a precomputed geometric morph between the umbrellas of the finer LoD to the merged umbrellas of the coarser LoD.

Figure 9 illustrates how, at the highest level of detail, the synthesized texture is rendered with full 3-D detail, then the height of the 3-D detail is tapered off gradually, and then coarser and coarser LoDs of the arrangement grid are used. Whereas tapering off 3-D detail can be done at run-time, the coarser LoDs of the arrangement grid (Section 5.1) and the

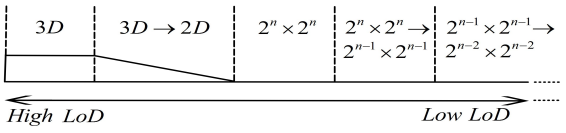

Figure 9: *Just-in-time texture synthesis LoD continuum.*

morphs between them (Section 5.2) have to be pre-computed off-line, akin to pre-computing the mipmap levels of a conventional texture.

5.1. Arrangement map LoDs

We pre-compute coarser levels of detail of the arrangement grid hierarchically from the bottom up. Consider a grid with at most k modified umbrellas per grid cell. The next coarser level is computed by merging 4 neighboring cells into 1 cell with k umbrellas using Algorithm 1.

Algorithm 1 *Arrangement map LoD computation.*

```
Group up to 4k umbrellas into k clusters
for each cluster C_i do
    Compute center o_i of C_i
    Compute convex hull h_i of C_i
    Simplify h_i to s_i
    Create new umbrella {o_i, s_i}
    Compute new umbrella vertex colors
end for
```

The up to $4k$ umbrellas are grouped into k clusters by running the k-means algorithm on the umbrella centers. In Figure 10 each of the 4 cells (left, white squares) contains up to k=8 umbrellas (leaves delimited by blue lines), also counting umbrellas that only partially overlap with a cell. The cells are merged into a single cell (right, white square) with k=8 new umbrellas (red lines). A new umbrella is constructed for each cluster. The center o_i of the new umbrella is set as the center of mass of the centers of the umbrellas in the cluster. The peripheral vertices s_i of the new umbrella are derived from the convex hull h_i of the cluster. h_i is simplified to stop the proliferation of vertices as the algorithm is run hierarchically. A maximum number of peripheral vertices is enforced by removing vertices with edge angles closest to 180°.

Once the shape of the new umbrella is known, its color is defined by computing colors for each of its vertices. New umbrellas are not texture mapped, thus they do not incur a significant additional storage cost. Figure 10 right shows the vertex colors for the new umbrellas. The color of a vertex is computed as a weighted sum of the color samples in the neighborhood of the vertex and inside the new umbrella. We use a raised cosine reconstruction filter with a base of half the distance from the vertex to the umbrella center. For the center, the base is half the distance to the peripheral vertices.

Our LoD algorithm essentially implements mipmapping directly in the grid of umbrellas. Just like in conventional mipmapping, LoDs are pre-computed off-line to avoid the performance penalty of on-the-fly LoD adaptation. The algorithm is designed such that it can be applied recursively, which requires that complexity does not increase (i.e. accumulate) from one level to the next. The coarser level of detail has the same per cell complexity as the finer level from which it is computed: a grid cell in the coarser level stores umbrellas (and not more complex polygonal representations), and the number of umbrellas and the number of vertices per umbrella are still bound by the same global upper limits. To achieve this, the algorithm resorts to two main approximations. First, the cluster of umbrellas is approximated with its convex hull. Second, the color information stored in the textures of the umbrellas is approximated using vertex colors. These approximations work well since the output image footprint of the umbrellas is small and since the colors of merged umbrella vertices are computed from samples, which could include background samples. Figure 11 (left) shows the output of our algorithm for the case shown in Figure 10. The output image footprint of the merged cell is 8×8 pixels, shown here magnified for illustration purposes. The top row shows a nearest neighbor magnification and the bottom row a bilinear magnification. The result is comparable to mipmapping the corresponding high resolution texture (right). Like all texture bombing approaches, our method has to antialias the edges of the umbrellas, which we achieve with little per-

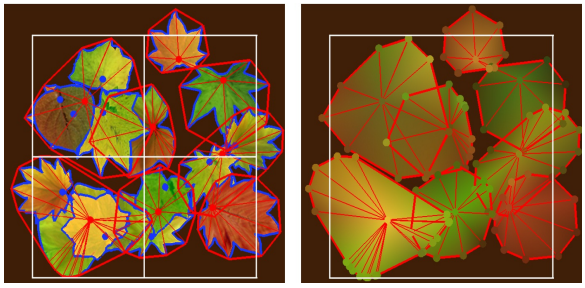

Figure 10: *Arrangement grid LoD. Four neighboring grid cells (left) are merged into one (right).*

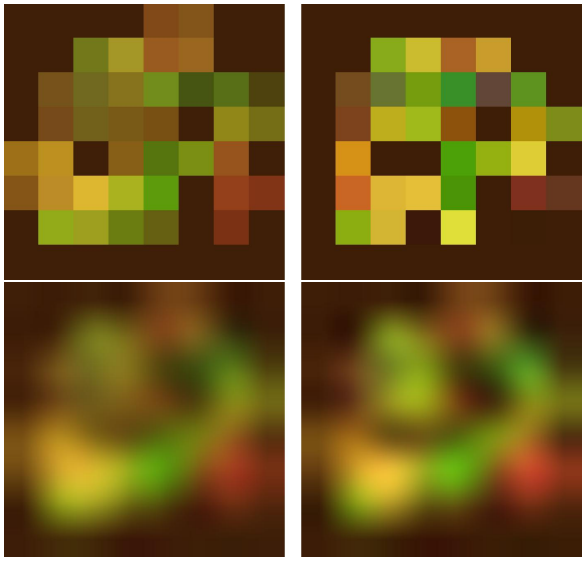

Figure 11: *Comparison between minification with our LoD algorithm (left) and with mipmapping (right).*

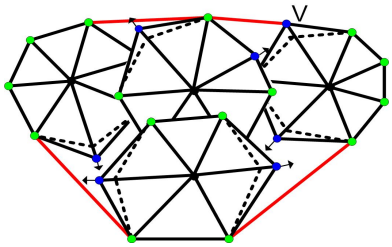

Figure 12: *LoD morphing for a group of umbrellas.*

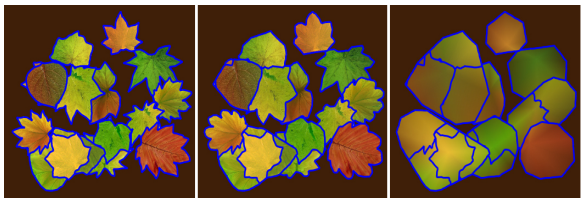

Figure 13: *LoD morph: initial(left), intermediate(middle), and final stage(right).*

formance penalty by computing multiple color samples per output pixel.

5.2. LoD morphing

As discussed, an abrupt transition from one arrangement LoD to the next is not acceptable. A simple solution for switching gradually from one LoD to the next is to lookup both LoDs and to blend with weights determined by the fractional value of the actual LoD needed. For textures where the difference between the two colors looked up in consecutive LoDs is considerable, texture clarity can be improved by a more complex transition between LoDs.

What is needed is a gradual transition of the geometry of the umbrellas to the geometry of the merged umbrellas. For this, we pre-compute a morph between each group of umbrellas and their corresponding merged umbrella. The morph moves the vertices of the umbrellas gradually to define a shape that closely approximates the convex hull. In Figure 12, a group of 4 umbrellas (black lines) is merged to its convex hull (red lines). The morph moves the vertices (blue) that are not inside an umbrella and that are not on the convex hull, radially away from the centers of their respective umbrellas until they touch an umbrella boundary or the convex hull. Vertex V has reached its final destination on the convex hull. Figure 13 illustrates LoD morphing for several groups, from the initial, to an intermediate, and then to the final configuration. The blue lines delimiting the umbrellas are shown for illustration purposes. Also see the accompanying video segment LoDMorphing.mov.

6. Just-in-time texture mapping

The umbrellas, the arrangement map, and the arrangement map LoD are used at runtime to texture the polygons on which the synthesized texture is mapped. Section 6.1 gives the high level algorithm for just-in-time texture mapping a polygon. If 3-D detail has to be rendered, the intersection between the ray of the current pixel and the closest umbrella ellipsoid is found as described in Section 6.2. Finally, the final color is looked up in the 2-D umbrella as described in Section 6.3.

6.1. Just-in-time texture mapping algorithm

The synthesized texture is encoded using a 1-D array of base umbrellas, a 1-D array of modified umbrellas, and a hierarchy of 2-D arrays for the arrangement grid. A base umbrella is encoded with a texture map and with the texture coordinates of its vertices. A modified umbrella is encoded with the index of its base umbrella, with per vertex color and position, and, if 3-D shape is desired, with parameters defining the underlying ellipsoid. The arrangement grid stores an array of modified umbrella indices for each cell. This encoding is used to texture surfaces as required by the output frame.

Consider a polygon to be textured with our technique. In order to render the 3-D detail with the correct silhouette, the polygon is extruded to form a prism with height h, where h is the maximum height of the 3-D detail. Figure 8 (right) shows the 3-D nature of the texture and the need to extrude the base polygon in order to ensure that all pixels that need to be textured are touched. In other words, not all pixels needing to show 3-D detail are covered by the base polygon. This is common practice for techniques that render 3-D detail on the GPU (e.g. relief texture mapping). Each pixel touched by the prism is textured using Algorithm 2.

Algorithm 2 *Per-pixel just-in-time texturing.*

```
P0P1 = pixel ray intersected with prism
cell = GetCell(P0)
while cell do
    Set3DLoD(cell)
    if cell is 3-D AND Intersect3D(cell, P0P1) then
        return 3-D sample
    end if
    if cell is 2-D then
        return LookUp2D(P1)
    end if
    cell = NextCell(cell, P0P1)
end while
return no-sample
```

The ray at the current pixel is first intersected with the prism to find the ray segment P_0P_1 (Figure 14). Then P_0P_1 is traced through the grid of cells extruded to height h, starting from the grid cell that contains the starting point P_0.

The first step in processing a cell is to set the amount of

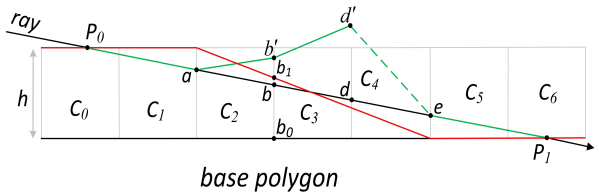

Figure 14: *Intersection between ray and grid cells ($c_0 - c_6$). The red line shows the height of the 3-D detail tapering off over cells $c_2 - c_4$. The green line shows the virtual ray modification that implements the tapering of 3-D detail.*

3-D detail that has to be rendered according to the desired 3-D to 2-D LoD adaptation. In Figure 14, the ray traverses cells c_0 to c_6. Cells c_0 and c_1 have full-height 3-D detail, the height of the 3-D detail is tapered off over cells c_2 - c_4, and then cells c_5 and c_6 have no 3-D detail (see red line).

A cell with 3-D detail is intersected with the ray as described in Section 6.2 (Algorithm 3). If an intersection is found, the traversal stops and the sample is returned. Otherwise the algorithm continues with the next cell traversed by the ray. The traversal terminates the first time a 2-D cell is encountered, i.e. a cell where the 3-D detail has been tapered off completely. The texture is looked up at the intersection point between the ray and the base polygon and the sample is returned. The lookup algorithm is given in Section 6.3 (Algorithm 4). In our example the texture is looked up at P_1 when 2-D cell c_4 is processed.

6.2. 3-D texture lookup

A grid cell with 3-D detail (*cell*) is intersected with a ray (P_0P_1) according to Algorithm 3. The ray P_0P_1 is first clipped with the axis aligned bounding box of the cell, obtaining R_0R_1. The ray segment R_0R_1 is then modified to account for the possible reduction in height of the 3-D detail. Modifying the ray and intersecting the uncompressed cell with the modified ray is easier than compressing the cell and intersecting it with the original ray. In Figure 14 the ray is not modified for cells c_0 and c_1 which are rendered with full-height 3-D detail, but ray segment ab is modified for cell c_2 to ab', by moving b to b'. b' is found such that $b'b_0$ / h = bb_0 / b_1b_0. The resulting ray ab' has the same endpoints with respect to the uncompressed cell as the original ray segment ab with respect to the compressed cell. ab' is intersected with the uncompressed cell. Similarly, bd is modified to $b'd'$, maintaining ray continuity from cell c_2 to cell c_3 (green line $ab'd'$). For cell c_4 de is above the compressed cell thus no intersection needs to be computed (dotted green line).

The cell is intersected with the modified ray by intersecting each umbrella in the cell, and by recording the closest intersection. An umbrella is intersected by first intersecting the ellipsoid defining its 3-D shape. The ellipsoid intersection

Algorithm 3 *Intersection of pixel ray with 3-D cell.*

```
R0R1 = ClipRayWithCellBoundingBox(P0P1, cell)
Q0Q1 = ModifyRay(R0R1, cell)
S = no-sample
for all modified umbrellas u in cell do
    if (Q = Intersect(Q0Q1, u.ellipsoid)) != 0 then
     if (Si = LookUp(Q, u.2Dpolygon)) != 0 then
      S = ClosestToEye(S, Si);
end for
return S
```

implies solving a quadratic. Once the intersection is found, the (x, y) coordinates of the intersection (where the x and the y axes define the umbrella plane), define a 2-D point where the umbrella is looked up, as described in Section 6.3.

6.3. 2-D texture lookup

Given point P on the base polygon with texture coordinates (s,t), the color at P is looked up with Algorithm 4. Since the grid is uniform, the cell containing P is found directly by dividing s and t by the width and height of the cell. The modified umbrellas in the cell are traversed in front to back order in search of an intersection. The base umbrella sector possibly containing P is found using the angle φ between the vector defined by P and the horizontal axis (Figure 15, left). If P is actually inside the triangle sector, an intersection has been found and a color is returned. The color is computed by blending the base umbrella texture color with the interpolated color of the triangle sector (found using Equation 1). If no umbrella covers P, the background color is returned.

Algorithm 4 *2-D texture lookup at point P.*

```
Find grid cell that contains P
for all modified umbrellas u in cell do
    Compute angle φ of P with the horizontal axis
    Use φ to find sector triangle Tj containing P
    if P outside peripheral edge ej then continue
    Compute barycentric coordinates (α, β, γ) of P in Tj
    Lookup base umbrella texture color ct at (α, β, γ)
    Compute interpolated vertex color cv
    return blended final color ct+fcv
end for
return background color
```

The level of detail is adapted by examining the derivatives of the texture coordinates in arrangement grid units. While these derivatives are sufficiently small (i.e. 0.125), the LoD is simply adapted by looking up the base umbrella textures with mipmapping. Once the derivatives become too large, the coarser LoDs of the arrangement grid will be used. The lookup algorithm is run on the two LoDs that bracket the desired LoD, and a linear interpolation provides a smooth transition between LoDs (Figure 2 and video).

We enhance the appearance of the texture (Figure 15,

right) by approximating shadows with a small addition to Algorithm 4. As the umbrellas of the cell are traversed we also test whether the lookup point is in the shadow cast by the current umbrella. This is done by moving the point towards the light on the texture plane, and by testing whether the displaced point is inside the umbrella, which would indicate that the original point is in the umbrella's shadow. In Figure 15, left, P_1 is translated along the light vector l to P_1' which is inside the umbrella thus P_1 is in shadow.

So far we have discussed our LoD scheme designed to support arbitrary minification, a challenging and serious problem that previous texture bombing techniques ignore. Regarding magnification, i.e. the case when texture resolution is exceeded by output image resolution, our technique, like any bombing technique, outperforms conventional texturing as the edges of the umbrellas remain thin no matter what the magnification factor.

7. Results and Discussion

We have applied our technique to generate and use several textures: *Fall leaves* (Figure 1), *Berries* (Figure 16, row 1), *Green leaves* (row 2), *Pepper* (row 3), and *Flowers* (row 4). The textures *Fall leaves*, *Berries*, and *Green leaves* encode surface 3-D detail, whereas the other two do not. The resolution of the base umbrella textures is 256×256, which allows zooming in with good detail. Our LoD algorithm provides quality results even at extreme minification rates. As illustrated in the video, our technique is stable which preserves quality in sequences of frames.

7.1. Storage Reduction Performance

In order to quantify the texture memory savings brought by our just-in-time texture encoding, let's assume that there are b base umbrellas, each with v vertices and with a texture of resolution $w \times h$. The storage cost of a base umbrella is wh + $2v$ four-byte words, where we counted 2 floats per vertex for the texture coordinates. Let n be the number of modified umbrellas. The cost of a modified umbrella is 1 + $3v$ + 3 + 1 words, which accounts for the base umbrella index, for the positions and colors of the vertices, for the 3 parameters

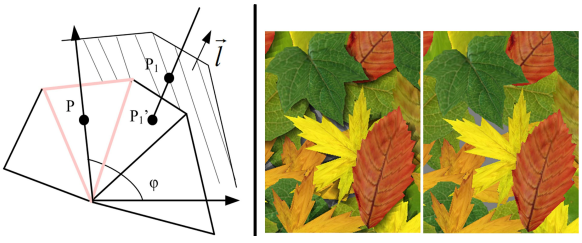

Figure 15: *Umbrella 2-D lookup with shadowing (left), and comparison between texture w/ and w/o shadows (right).*

Figure 16: *Additional texture synthesis examples and corresponding base umbrellas.*

defining the ellipsoid, and for the height of the umbrella, respectively. Each grid cell records the modified umbrellas it stores with integer indices. Thus, for a $W \times H$ grid with at most k modified umbrellas per cell, the overall cost in four-byte words J of our just-in-time texture encoding is:

$$J = bwh + 2bv + n(3v + 5) + kWH \qquad (2)$$

In order to compare this cost to that of a conventional approach storing the synthesized texture explicitly, first we have to determine the resolution of the synthesized texture. Since modified umbrellas have different sizes, the resolution of the texture has to be determined by examining the resolution at individual modified umbrellas. A modified um-

brella U_i with an arrangement grid axis aligned bounding box of $x_i \times y_i$ implies a synthesized texture resolution of $w/x_i \times h/y_i \times W \times H$. The dimensions x_i and y_i are measured in grid cell units. Thus, in order to not lose information at any of the modified umbrellas, the synthesized texture should have a resolution T of:

$$T = max(w/x_i) \times max(h/y_i) \times W \times H \qquad (3)$$

where the maxima are computed over all modified umbrellas. Using min instead of max in the equation above corresponds to a synthesized texture that loses color resolution at all modified umbrellas but the one with the largest arrangement grid footprint. A third option is to use the average resolution over all modified umbrellas. Table 1 gives the storage reduction factors achieved by our method versus conventional texture synthesis, for each of these 3 options. Just-in-time texture synthesis achieves lossless storage reduction with substantial factors. Base umbrella texture resolution $w \times h$ is 256×256, arrangement grid resolution $W \times H$ is 64×24 (10×13 for *Flowers*), and the values for the other texture synthesis parameters are given in the table.

7.2. Rendering Performance

Just-in-time texture synthesis achieves storage savings by shifting texture expansion from pre-processing to run-time, a classic trade-off between storage and computation cost. Since conventional texture mapping does not support rendering surface 3-D detail, we first analyze the performance of just-in-time texture synthesis without surface 3-D detail.

Instead of a single mipmapped lookup, the fragment program has to compute the intersection between the sampling location and the modified umbrellas at the current grid cell. As such, the rendering cost depends on two main factors: the number of modified umbrellas per grid cell k and the complexity of the umbrellas v. Figure 17 shows the variation of the rendering performance with k and v for *Fall leaves*. The output image resolution is 512×512, $W \times H = 32 \times 32$, and $w \times h = 256 \times 256$. Rendering was done by computing 4 color samples per output image pixel, in order to antialias umbrella edges. Performance was measured on an Intel Core i7-2600 3.40GHz PC with an NVIDIA GeForce GTX 580, 1,280 MB graphics card.

Rendering 3-D detail adds the cost of intersecting a pixel ray with the cells it traverses, until an intersection is found or until the 3-D detail tapers off. Since the height of the 3-D detail is small compared to the size of the base polygon, and since typical rays are not grazing the base polygon, the number of cells considered for 3-D intersection is typically small. For a texture like the one shown in Figure 2 most pixels are computed without 3-D intersection, and, for the pixels where 3-D detail is rendered, the median and maximum number of cells considered for 3-D intersection is 3 and 6, respectively.

Table 1: Storage reduction performance

Texture	b	v	k	n ($\times 1{,}000$)	Min	Avg	Max
Fall L.	9	63	19	6.1	16:1	25:1	47:1
Berries	3	58	16	14	10:1	23:1	48:1
Green L.	6	51	68	25	5:1	13 :1	53 :1
Peppers	4	35	37	14	21:1	32:1	56:1
Flowers	6	74	12	.39	5:1	10:1	18:1

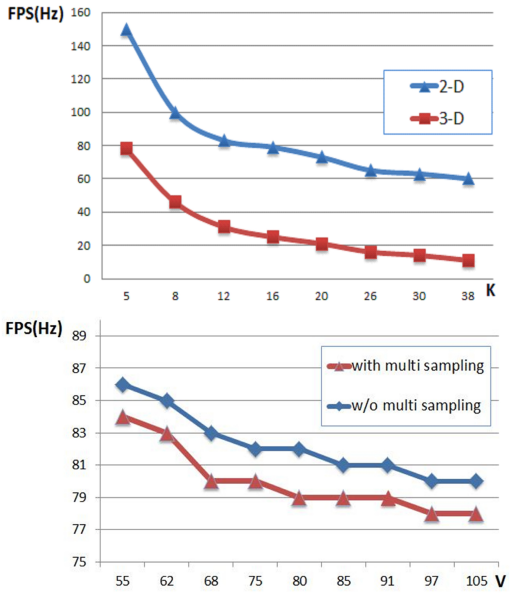

Figure 17: *Rendering performance variation with k (top, v = 76), and v (bottom, k = 13) for Fall leaves.*

All umbrellas have their 3-D shape modeled with an ellipsoid thus the intersection cost only depends on the number of umbrellas k (Figure 17, top).

7.3. Limitations

Even though the umbrella is a versatile geometry and color representation, which models well many discrete texture elements, not all discrete textures can be modeled efficiently with umbrellas (e.g. grass blades). Another limitation is that the LoD hierarchy is built using the convex hull, which exaggerates the size of the merged umbrella. The convex hull is computed easily, it can be easily transformed into an umbrella, and it works well in practice (Figure 11).The approximation error is more noticeable in the case of sparse umbrellas, Figure 18. In the case of umbrellas with great color variation, the color information culling from one level of the LoD hierarchy to the next could be too aggressive–there is a single color sample inside the convex hull. Our LoD scheme first tapers off 3-D detail completely before 2-D shape and color resolution is reduced. This works well for typical umbrellas which have a far greater x-y than z extent, like the examples shown in the paper. However, to support thick layers

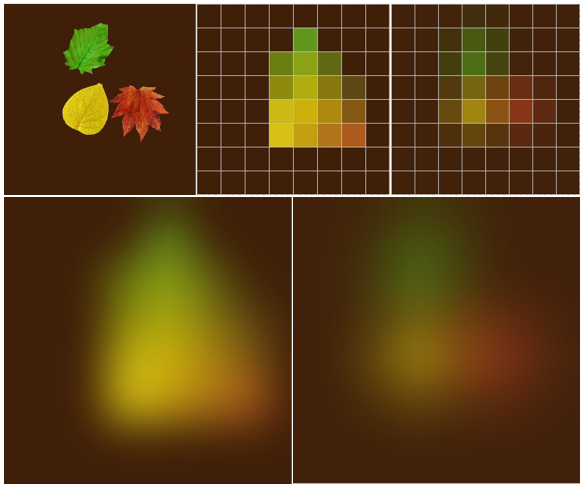

Figure 18: *Illustration of convex hull approximation limitation. Three umbrellas (top left) are minified to an 8x8 screen region with our method (top middle) and through bi-cubic down-sampling (top right), which provides down truth for comparison.Our method overestimates The top middle and top right images are also shown bi-cubically upsampled to 512x512 resolution for illustration purposes. Our method based on the convex hull overestimates the footprint of the merged umbrellas, which is noticeable in the case of sparse umbrellas over a contrasting background, as shown in this example.*

of tiny umbrellas, the LoD scheme should allow for simultaneous 3-D and 2-D detail adjustment. Finally, rendering performance of just-in-time texture synthesis is lower than for conventional texture mapping and it decreases with the overlap factor.

Our custom rasterization of umbrellas precludes the use of standard hardware antialiasing and requires implementing multi-sampling explicitly. However, graphics hardware is now sufficiently powerful such that multi-sampling can be practical even for the most demanding interactive applications. The performance penalty brought by multi-sampling is shown in Figure 17, bottom. Finally, our present implementation does not support anisotropic filtering, which is needed to avoid excessive blurriness when minification is strongly direction dependent in the texture plane.

8. Conclusions and Future Work

We have presented a novel texture synthesis approach that models texture patches with parameterizable color, 2-D, and 3-D shape. A few input patches are sufficient to mimic the diversity present in nature. Other texture diversification techniques can be plugged into our pipeline. Texel data is computed just-in-time, which brings substantial storage savings. An LoD algorithm only renders 3-D detail where needed, and supports artifact-free continuous minification at any level. The LoD algorithm solves a fundamental problem in texture bombing–texture bombing without adapting detail across patches results in severe artifacts that make the approach unusable.

One possible direction of future work is to alleviate some of the limitations discussed above. The 2-D shape and color modeling power of umbrellas can be further increased by introducing additional vertices on the radii connecting the center to the peripheral vertices. This would allow for greater color diversification and LoD adaptation flexibility. Color modeling power could also be increased by resorting to more sophisticated interpolation schemes, such as those developed in the context of image and video processing for propagating user edits [AP08, XLJ*09]. The LoD shape fidelity could be improved by not requiring that the merged umbrella be convex, but rather by shrink wrapping it to the actual perimeter of the umbrellas in the cluster it replaces. This could be done by starting from the convex hull and by pulling in (i.e. towards the center) peripheral vertices that map over background. Anisotropic filtering could be supported in the future with a RIP map approach that precomputes umbrella LoDs on non-uniformly scaled versions of the arrangement grid, or by approximating the pixel footprint in texture space at run time with multiple lookups, a more costly but more accurate solution.

Our method has the merit of placing 3-D and 2-D surface detail on a continuum, which allows switching gradually and automatically from 3-D to 2-D. Compared to an approach that renders 3-D triangles, our approach bypasses the difficult task of 3-D level of detail adaptation through simplification. Our method is general and supports any representation of 3-D detail. For example the 3-D modeling power of our technique could be increased by using a more powerful 3-D representation such as a height field, which only requires replacing the ray/ellipsoid intersection with a ray/height field intersection.

Our paper focuses on the texture synthesis sub-problems of diversification and LoD adaptation. Another possible direction of future work is to integrate our method with prior solutions for automatic extraction of texture elements and of arrangement patterns, and for distortion-free tiling on 2-D and 3-D domains. Just-in-time texture synthesis takes advantage of the programmability sophistication of graphics hardware by replacing the texel with a higher-level texturing primitive. Our method brings benefits whose importance will only grow as increases in computation performance continue to outpace increases in storage and bandwidth.

Acknowledgment

We would like to thank Qijiang Jin for help with the implementation. This work is the part of Project 61272349 supported by National Natural Science Foundation of China and

Project 043/2009/A2 funded by Macao Science and Technology Development Fund, and is also supported by Beijing Science Technology Star Plans and Technology Star Plans (No. 2009B09).

References

[ACo12] ASSA J., COHEN-OR D.: More of the same: Synthesizing a variety by structural layering. *Computers and Graphics 36*, 4 (2012), 250–256.

[AP08] AN X., PELLACINI F.: Mean value coordinates. *ACM Transactions on Graphics 27*, 3 (2008).

[BAC96] BEERS A., AGRAWALA M., CHADDHA N.: Rendering from compressed textures. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New Orleans, LA, USA, 1996), ACM Press, pp. 373–378.

[BIP00] BAJAJ C., IHM I., PARK S. H.: Compression-based 3D texture mapping for real-time rendering. *Graphical Models 62*, 6 (2000), 391–410.

[Bli78] BLINN J.: Simulation of wrinkled surface. In *Proceedings of the 5th annual conference on Computer Graphics and Interactive Techniques* (Atlanta,Georgia,USA, 1978), ACM Press, pp. 286–292.

[Coo84] COOK R.: Shade trees. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (Minneapolis, USA, 1984), ACM Press, pp. 223–231.

[CSHD03] COHEN M. F., SHADE J., HILLER S., DEUSSEN O.: Wang tiles for image and texture generation. *ACM Transactions on Graphics 22*, 3 (2003), 287–295. SIGGRAPH 2003.

[DCH05] DIVERDI S., CANDUSSI N., HOLLERER T.: *Real-time rendering with wavelet-compressed multi-dimensional textures on the GPU*. Computer Science Technical Report 2005-05, 2005.

[DGAG06] DESBENOIT B., GALIN E., AKKOUCHE S., GROSJEAN J.: Modeling autumn sceneries. In *Eurographics(EG) short paper* (2006).

[DHR12] DU S., HU S., R.R. M.: Semi-regular solid texturing from 2d exemplars. *IEEE Transactions on Visualization and Computer Graphics* (2012).

[DLC05] DONG F., LIN H., CLAPWORTHY G.: Cutting and pasting irregularly shaped patches for texture synthesis. *Computer Graphics Forum 24*, 1 (2005), 17–26.

[DMLG02] DISCHLER J. M., MARITAUD K., LEVY B., GHAZANFARPOUR D.: Texture particles. *Computer Graphics Forum 21*, 3 (Sept. 2002), 401–410.

[DZ06] DISCHLER J. M., ZARA F.: Real-time structured texture synthesis and editing using image-mesh analogies. *The Visual Computer 22*, 9 (2006), 926–935.

[EF01] 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* (Los Angeles, California, USA, 2001), ACM Press, pp. 341–346.

[Gla04] GLANVILLE S.: Texture bombing. In *GPU Gems chapter 20* (2004), Nvidia Corporation. http://http.developer.nvidia.com/GPUGems/gpugems_ch20.html.

[Gum03] GUMHOLD S.: Splatting illuminated ellipsoids with depth correction. In *Proceeding of the 8th International Fall Workshop on Vision, Modelling and Visualization* (Munich, Germany, 2003), Aka GmbH, pp. 245–252.

[HDKS00] HEIDRICH W., DAUBERT K., KAUTZ J., SEIDEL H.: Illumination micro geometry based on precomputed visibility. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New Orleans, Louisiana, USA, 2000), ACM Press/Addison-Wesley Publishing Corporation, pp. 455–464.

[HQXT05] HUANG J., QI D., XIONG C., TANG Z.: Foreground-distortion method for image synthesis. In *Proceedings of the 9th International Conference on Computer Aided Design and Computer Graphics* (Hong Kong, China, 2005), IEEE, pp. 509–513.

[IMIM08] IJIRI T., MECH R., IGARASHI T., MILLEI G.: An example-based procedural system for element arrangement. *Computer Graphics Forum 27*, 2 (2008), 429–436.

[KCoDL06] KOPF J., COHEN-OR D., DEUSSEN O., LISCHINSKI D.: Recursive wang tiles for real-time blue noise. *ACM Transactions on Graphics 25*, 3 (2006), 509–518. (Proceedings of SIGGRAPH 2006).

[KE02] KRAUS M., ERTL T.: Adaptive texture maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware* (Saarbrucken, Germany, 2002), Eurographics Association Aire-la-Ville, pp. 7–15.

[KS01] KAUTZ J., SELDEL H.: Hardware accelerated displacement mapping for image based rendering. In *Proceedings of Graphics Interface* (Ottawa, Ontario, Canada, 2001), Canadian Information Processing Society, pp. 61–70.

[KSE*03] KWATRA V., SCHODL A., ESSA I., TURK G., BOBICK A.: Graphcut textures : Image and video synthesis using graph cuts. *ACM TOG 22*, 3 (July 2003), 277–286.

[KTI*01] KANEKO T., TAKAHEL T., INAMI M., KAWAKAMI N., YANAGIDA Y., MAEDA T., TACHI S.: Detailed shape representation with parallax mapping. In *Proceeding of the 11th International Conference on Artificial Reality and Telexistence* (Tokyo, Japan, 2001), Addison-Wesley, pp. 205–208.

[LD05] LAGAE A., DUTRE P.: A procedural object distribution function. *ACM Transactions on Graphics 24*, 4 (2005), 1442–1461.

[LH05] LEFEBVRE S., HOPPE H.: Parallel controllable texture synthesis. *ACM Transactions on Graphics 24*, 3 (2005), 777–786.

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. *ACM Transactions on Graphics 25*, 3 (July 2006), 541–548. (SIGGRAPH 2006).

[LHN05] LEFEBVER S., HORNUS S., NEYERT F.: Texture sprites: Texture elements splatted on surfaces. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games* (Washington, DC,USA, 2005), ACM Press, pp. 163–70.

[LLH04] LIU Y. X., LIN W. C., HAYS J. H.: Near-regular texture analysis and manipulation. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 368–376. (SIGGRAPH 2004).

[LLX*01] LIANG L., LIU C., XU Y.-Q., GUO B., SHUM H.-Y.: Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics 20*, 3 (July 2001), 127–150.

[LN03] LEFEBVRE S., NEYRET F.: Pattern based procedural textures. In *ACM-SIGGRAPH Symposium on Interactive 3D Graphics* (Monterey, CA, USA, 2003), ACM Press, pp. 203–212.

[LP00] LEFEBVRE L., POULIN P.: Analysis and synthesis of structural textures. In *Proceeding of Graphics Interface* (Montreal,Canada, 2000), Canadian Human-Computer Communications Society, pp. 77–86.

[LTcL05] LIU Y., TSIN Y., CHIEH LIN W.: The promise and perils of near-regular texture. *International Journal of Computer Vision- Special Issue on Texture Analysis and Synthesis 62*, 1-2 (April-May 2005), 145–159.

[Max88] MAX N.: Horizon mapping: Shadows for bump-mapped surfaces. *The Visual Computer 4*, 2 (1988), 109–117.

[MGGA10] MARECHAL N., GALIN E., GUERIN E., AKKOUCHE S.: Component-based model synthesis for low polygonal models. In *Proceedings of Graphics Interface 2010* (Ottawa,Ontario,Canada, 2010), Canadian Information Processing Society, pp. 217–224.

[Mic03] MICHAEL J.: Mean value coordinates. *Computer Aided Geometric Design 20*, 1 (2003), 19–27.

[MWT11] MA C. Y., WEI L. Y., TONG X.: Discrete element textures. *ACM Transactions on Graphics 30*, 4 (2011), 1–10. (SIGGRAPH 2011).

[NMMK05] NICOLL A., MESETH J., MULLER G., KLEIN R.: Fractional fourier texture masks: Guiding near-regular texture synthesis. *Computer Graphics Forum 24*, 3 (Sept. 2005), 569–579.

[Pea85] PEACHEY D.: Solid texturing of complex surfaces. *Computer Graphics Forum 19*, 3 (1985), 17–26.

[Per85] PERLIN K.: An image synthesizer. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (San Francisco, USA, 1985), ACM Press, pp. 287–296.

[PFH00] PRAUN E., FINKELSTEIN A., HOPPE H.: Lapped textures. In *Proceeding of the 27th annual conference on Computer graphics and interactive techniques* (New Orleans, Louisiana, USA, 2000), ACM Press, pp. 465–470.

[POC05] POLICARPO F., OLIVEIRA M., COMBA J.: Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games* (Washington, DC, USA, 2005), ACM Press, pp. 155–162.

[PTMG08] PEVRAT A., TERRAZ O., MERILLOU S., GALIN E.: Generation vast varieties of realistic leaves with parametric 2gmaps l-systems. *The Visual Computer 24*, 7-9 (2008), 807–816.

[RCS04] RODKAEW Y., CHONGSTITVATANA P., SIRIPANT S.: Modeling plant leaves in marble-patterned colours with particle transportation system. In *Proceedings of the 4th International Workshop on Functional-Structural Plant Models* (Montpellier,France, 2004), UMR AMAP, pp. 391–397.

[RHDG10] RISSER E., HAN C., DAHYOT R., GRINSPUN E.: Synthesizing structured image hybrids. *ACM Transactions on Graphics 29*, 4 (2010), 1–6.

[RSL*02] RODKAEW Y., SIRIPANT S., LURSINSAP C., CHONGSTITVATANA P., FUJIMOTO T., N. C.: Modeling leaf shapes using l-system and genetic algorithms. In *Proceeding of NICOGRAPH International* (Tokyo, Japan, 2002), Addison-Wesley, pp. 73–88.

[SA79] SCHACHTER B., AHUJA N.: Random pattern generation processes. *Computer Graphics Forum 10*, 2 (1979), 95–114.

[SC00] SLOAN P., COHEN M.: Interactive horizon mapping. In *Proceedings of the Eurographics Workshop on Rendering Techniques* (Brno, Czech Republic, 2000), Springer-Verlag Berlin, pp. 281–286.

[SR06] STACHERA J., ROKITA P.: Gpu-based hierarchical texture decompression. In *Eurographics short paper* (Vienna, Austria, 2006), Eurographics Association, pp. 33–36.

[STC09] SUN C. H., TSAO Y. M., CHIEN S. Y.: High-quality mipmapping texture compression with alpha maps for graphics processing units. *IEEE Transactions on Multimedia 11*, 4 (2009), 589–599.

[TF08] TANG Y., FAN J.: Incremental texture compression for real-time rendering. In *Proceedings of the 4th International Symposium on Advances in Visual Computing (ISVC '08), Part II* (Las Vegas, Nevada, USA, 2008), Springer-Verlag New York, pp. 1076–1085.

[TW08] TZENG S., WEI L.: Parallel white noise generation on a gpu via cryptographic hash. In *Proceeding of symposium on Interactive 3D graphics and games* (Redwood City, CA, USA, 2008), ACM Press, pp. 79–87.

[Wei04] WEI L.: Tile-based texture mapping on graphics hardware. In *Proceeding of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (Grenoble, France, 2004), Eurographics Association, pp. 55–63.

[Wil83] WILLIAMS L.: Pyramidal parametrics. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques* (Detroit, Michigan, USA, 1983), ACM Press, pp. 1–11.

[WLKT09] WEI L.-Y., LEFEBVRE S., KWATRA V., TURK G.: State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR* (Munich, Germany, 2009), Eurographics Association, pp. 1–25.

[WWT*03] WANG L., WANG X., TONG X., LIN S., HU S., GUO B., SHUM H.: View-dependent displacement mapping. *ACM Transactions on Graphics 22*, 3 (2003), 334–339.

[WY04] WU Q., YU Y. Z.: Feature matching and deformation for texture synthesis. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 362–365.

[XLJ*09] XU K., LI Y., JU T., HU S., LIU T.: Efficient affinity-based edit propagation using k-d tree. *ACM Transactions on Graphics 28*, 5 (2009).

[ZZV*03] ZHANG J., ZHOU K., VELHO L., GUO B., SHUM H.: Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics 22*, 3 (July 2003), 295–302. (SIGGRAPH '03).