Just-in-Time Texture Synthesis

paperID:1022
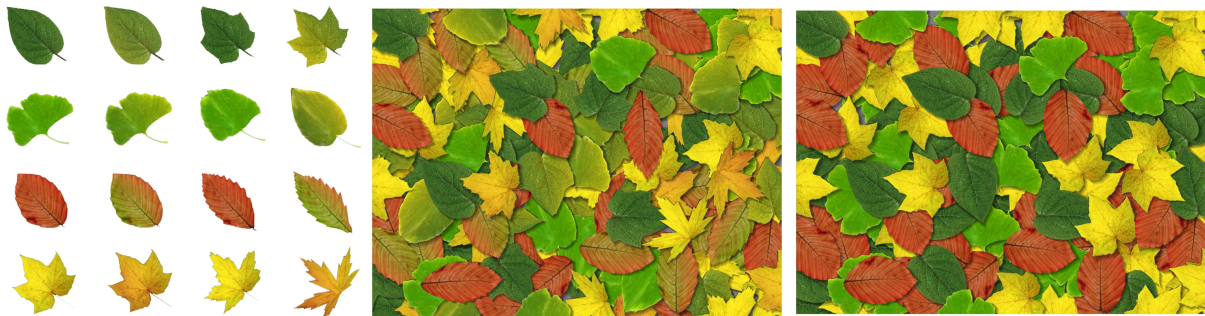

Figure 1: *(Left) umbrella diversification: the columns show, from left to right, the 4 base umbrellas, color modification, shape modification, and both color and shape modification. (Middle) texture synthesized with our method by diversifying the 4 base umbrellas. (Right) texture with same arrangement but only using the 4 base umbrellas, for comparison.*

Abstract

We present a method that generates large, high-resolution textures which mimic the diversity present in nature from only a small number of input texture patches. A texture patch is modeled as an umbrella, a versatile structure with parameterized color and 2-D shape. A small number of base umbrellas constructed from the input texture patches are sufficient to produce adequate patch diversity over a large texture. Our method computes the usual arrangement of patches but stops short of assembling the final texture at texel level. The base patches, the patch diversification parameters, and the patch arrangement map are used at run time to texture surfaces as required by the current frame. This just-in-time texture synthesis approach brings substantial texture memory savings. The storage savings come at the cost of a more complicated texture level-of-detail computation, for which we describe a hierarchical algorithm that operates directly on the arrangement map.

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 applications. Texture synthesis research efforts have produced techniques that construct high resolution textures based on input texture patches and patterns. However, several problems remain in need of better solutions. One problem 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. A second problem is reducing texture storage: the large, high-resolution synthesized texture requires large amounts of texture memory, a particularly scarce resource in the context of interactive graphics applications.

In this paper we present a texture synthesis method that addresses these two problems. Texture diversification is achieved by modeling a texture patch as an *umbrella*, a versatile 2-D geometry and texture structure with parameterized appearance. The input patch umbrellas are modified and
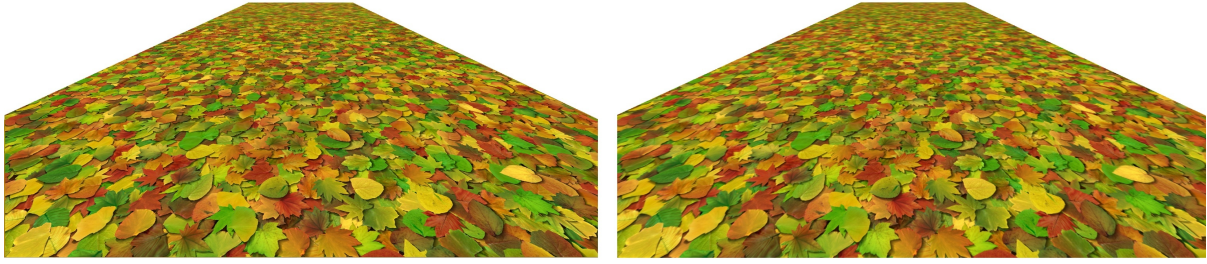
Figure 2: *Just-in-time texture synthesis (left, 9MB) and conventional texture of equivalent resolution (right, 190MB).*

arranged to synthesize a large, high-resolution, and diverse texture (Figure 1). The input patch is modified substantially by interpolation to new colors and shapes (left). The 4 base umbrellas are sufficient to create hundreds of unique modified umbrellas, so the synthesized texture (middle) does not suffer from repetitiveness. The repetitiveness artifact would be readily noticeable if the texture were synthesized only from the 4 base umbrellas (right). We also refer the reader to the accompanying video illustrating our results.

Texture storage reduction is achieved by stopping short of generating the 2-D array of texels for the synthesized texture. Instead, the texture is stored in much more compact form as a collection of input umbrellas, a set of input umbrella modification parameter values, and a modified umbrella arrangement map. This implies that texels are computed from this condensed representation just-in-time, as surfaces are textured for the current frame. Just-in-time texture synthesis brings substantial storage gains since the synthesized texture is never stored explicitly. However, this comes at the cost of a more complicated level-of-detail (LoD) adaptation. Conventional mipmapping cannot be used since that would require storing the texture LoDs explicitly. Instead, we propose a hierarchical LoD algorithm for just-in-time texturing that acts directly on the arrangement map. Lower LoD arrangement maps are computed offline by merging modified umbrellas. Figure 2 compares just-in-time texture synthesis with a conventional texture of similar resolution. As can be seen, the two images are very similar while just-in-time texture synthesis brings a storage space reduction of over 20:1.

Prior work considers texture synthesis and compact storage of textures for interactive rendering as separate problems: a large texture is synthesized first, followed by some form of storage reduction (i.e. encoding), and finally followed by run-time decoding. As the actual texture is synthesized, important information about the structure of the texture is lost before it can be leveraged for storage reduction. Instead, just-in-time texture synthesis advocates a tight coupling of texture synthesis and rendering. The structure of the synthesized texture, which is akin to an efficient encoding of the output texture, is preserved and used directly to decode texels as needed for the output image.

In summary, the main contributions of our paper are: i) the

umbrella structure for versatile modeling of texture patches; ii) the just-in-time approach to texture synthesis for storage space reduction; and iii) a hierarchical algorithm for LoD adaptation for just-in-time texture synthesis .

2. Related Work

As discussed in the introduction, prior research separates texture synthesis from the texture compression and decompression that is required in real time rendering.

2.1. Texture synthesis

A variety of texture synthesis methods have been developed. We also refer the reader to a comprehensive state-of-the-art report [WLKT09].

Texture synthesis 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 leafs covering the ground or a wall built from fitted river stones, or purely stochastic, such as a rough surface or the surface of the sea. One method that excels at synthesizing regular textures is based on modeling sample textures procedurally [LP00]. Other methods separate the sample texture into a regular and an irregular component, for example by using fractional Fourier analysis [NMMK05], which are then modeled independently, diversified and combined during texture synthesis [LTcL05]. Our method targets the synthesis of irregular textures.

Texture synthesis methods can also be classified as procedural or sample-based methods. Procedural methods use the input texture to derive a complex model that allows synthesizing completely new textures of the same type as the input provided. One of the most successful texture modeling approaches is based on Markov Random Fields [Pag04]. Sample-based methods use input texture patches repeatedly with minor modifications induced by changing a few parameters. Our method falls in the sample-based category which we review in more detail.

A sample-based texture synthesis method needs to address three tasks. The first task is to extract the texture patch from input images. Extraction is usually done with the help
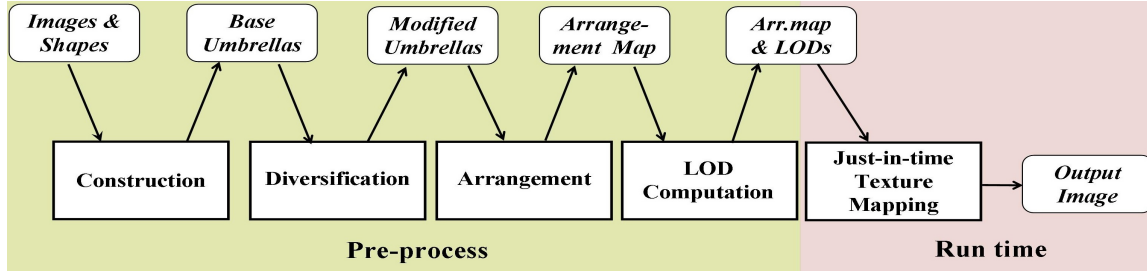
Figure 3: *Overview of the just-in-time texture synthesis pipeline.*

of image processing techniques for finding features and for segmentation [DMLG02, ZZV*03, WY04, LH06], but can also proceed through random selection of a rectangular window [LLX*01, KSE*03]. The second task is to arrange the extracted texture patches in the output texture domain. Some textures require an overlapping arrangement, which can be achieved by random placement of patches while controlling patch density [DMLG02, HQXT05]. Some textures require a tight, seamless tiling of patches, which can be achieved through graph cut techniques [KSE*03, EF01].

The third task in sample-based texture synthesis is to allow for the diversification of the appearance of input patches such that large textures that convey the diversity present in nature can be synthesized from only a small number of 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].

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. We rely on prior art solutions for the problems of patch extraction and arrangement, while we focus on achieving powerful patch diversification.

2.2. Texture encoding, decoding, and LoD

Although textures are images and image compression methods can be readily applied, the need for rapid decoding has spawned research targeting textures specifically. 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. For example wavelet data has been encoded into a standard 2-D texture memory region that is then sampled directly using drop-in shaders [DCH05]. The problem of texture level-of-detail adaptation has been solved by the introduction of mipmapping [Wil83]. Compression is orthogonal to mipmapping, as one can simply compress the individual mipmap levels.

Whereas prior work assembles a large texture that is then compressed, we propose to leverage the known repetition and parameterization of input patches to achieve more efficient compression. Since we render directly from patches, mipmapping can only be used within individual patches, and cannot be used between different patches as required for high levels of minification. Patches are modeled with triangle meshes and any of the techniques developed for geometry simplification can be used [LRC*02]. However, our umbrella patches are modeled with special, 2-D meshes and we have developed a hierarchical simplification approach that takes advantage of these known properties.

2.3. Procedural geometric modeling

Our just-in-time texturing technique is related to procedural geometric modeling. Several methods target specifically the procedural geometric modeling of foliage. The methods rely on L-Systems [RSL*02, PTMG08], on particle systems [RCS04] and probabilistic [DGAG06] algorithms, and on the diversification of low-count polygonal models [MGGA10] to capture leaf shape and color variation, and to simulate ecosystems and autumn scenery. Compared to these geometric procedural modeling 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 which is used directly during rendering as opposed to a 3-D geometric model which needs to be processed in its expanded form.

3. Algorithm overview

Just-in-time texture synthesis proceeds according to the diagram shown in Figure 3. The texture is synthesized offline in four major steps. First, base umbrellas are constructed from input photographs, renderings, and shapes containing the desired texture elements. A base umbrella is a parameterized encoding of the texture coordinates and color of a texture patch. Only a small number of base umbrellas are constructed (e.g. 4 for the texture in Figure 1). Second, the base umbrellas are diversified to hundreds of texture patches with unique appearance. Diversification is achieved by varying base umbrella color and shape parameters. Third, the modi-
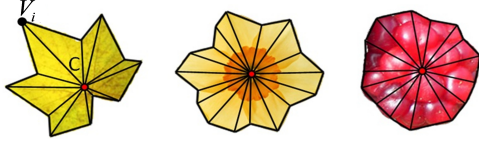
Figure 4: *Base umbrella examples.*

Figure 5: *Reference image (left), base umbrella and vertex colors (middle), and umbrella with modified color (right).*

fied 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 levels of detail needed to accommodate any minification level (Section 5). The umbrellas, the arrangement map, and the LODs are then used at runtime to texture surfaces as needed for the current output image (Section 6).

4. Umbrella Texture Patches

In this section we describe umbrella construction, diversification, and arrangement.

4.1. Construction

We define an umbrella as a texture-mapped 2-D geometric primitive with a central vertex C and peripheral vertices V_i (Figure 4). The umbrella need not be convex, but all segments V_iC have to be inside the umbrella. The umbrella is a flexible light-weight representation that captures many texture elements present in nature with high fidelity.

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 user first lays down the polyline defining the contour of the umbrella and then selects the center. For texture elements where features converge, the center is chosen at the convergence point for improved diversification results as discussed below. For example in the case of a leaf (Figure 4) the center is chosen at the convergence of the leaf veins. For the berry example the center is simply chosen as the centroid of the peripheral vertices. The interactive editor allows creating a base umbrella in seconds. Only a few umbrellas are needed (e.g. 4-10), which are then diversified automatically.

4.2. Color diversification

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:

$$c_p = c_t + fc_v$$
$$c_v = \sum d_i c_i / \sum d_i \tag{1}$$

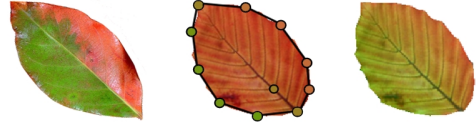where c_t is the color looked up in the base umbrella texture and c_v is a weighted average of the vertex colors c_i. A weight

d_i is defined as an inverse of the distance between vertex i and P. The coefficient f controls how much the original texture colors are modified. c_p is clamped to [0,1].

The colors of the vertices are chosen to produce realistic modified colors. The approach we use is to derive these colors from additional reference images of similar texture elements. For example, in Figure 5, an image of a leaf with different colors (left) was used to set the colors of the peripheral vertices of the base umbrella (middle). Such color diversification keeps the number of base textures low, while capturing a wide range of colors (right).

4.3. Shape diversification

The shape of a base umbrella is modified by moving peripheral vertices. A vertex can move to any new location as long as this does not create a fold. In order to allow for complex shape modifications, some base umbrella edges might be split into multiple segments. Figure 6 shows that the base umbrella has collinear peripheral vertices (blue) which can move progressively to create a final leaf shape that is significantly different from the starting shape. The shape morph is specified with a second position for each of the peripheral vertices. These positions can be designed by the user, or can be derived from the shape of other leafs. The shape morph allows producing a large number of plausible leaf shapes inexpensively and automatically.

4.4. Arrangement

The texture is synthesized by arranging modified umbrellas in the 2-D texture domain. Depending on the texture a
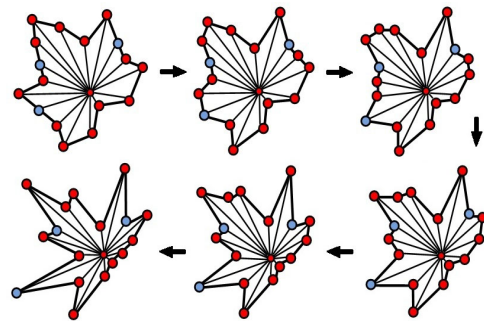

Figure 6: *Shape diversification by morphing.*

tiling or an overlapping arrangement is needed. The umbrella shape is sufficiently flexible to be compatible with previously developed arrangement methods. We exemplify our just-in-time texturing method using an overlapping arrangement defined using a regular grid. Modified umbrellas are assigned to grid cells. A grid cell is assigned all umbrellas that intersect it. 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.

5. Level of Detail

Whereas traditional texture synthesis methods actually compute a large texture that can be minified through conventional mipmapping, just-in-time texture synthesis requires a minification algorithm that works directly on umbrellas in the arrangement map. For moderate minification factors the mipmapping of the textures of the individual umbrellas is sufficient. However, once umbrellas have a small output image footprint (e.g. 1-4 pixels), they need to be merged.

We 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 a single cell with k umbrellas using the following algorithm.
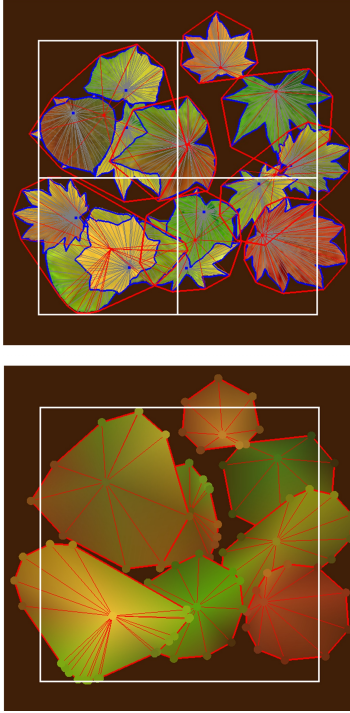

Figure 7: *Arrangement grid LoD. Four neighboring grid cells (top) are merged into one (bottom).*

```
Group 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 4 adjacent grid cells contain up to $4k$ umbrellas which are grouped into k clusters by running the k-means algorithm on the centers of the umbrellas. The algorithm is exemplified in Figure 7. The 4 cells (top, white lines) containing 16 umbrellas (leafs delimited by blue lines) are merged into a single cell (bottom, white lines) with k=8 new umbrellas (red lines). A new umbrella is constructed for each cluster. The shape of the umbrella is defined first. The center of the umbrella o_i is set to 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 original umbrellas. The convex hull is simplified to stop the proliferation of vertices as the algorithm is run hierarchically. A maximum number of peripheral vertices is enforced by removing the 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 7 bottom 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 our algorithm pre-computes levels of detail offline avoiding the performance penalty of on-the-fly LoD
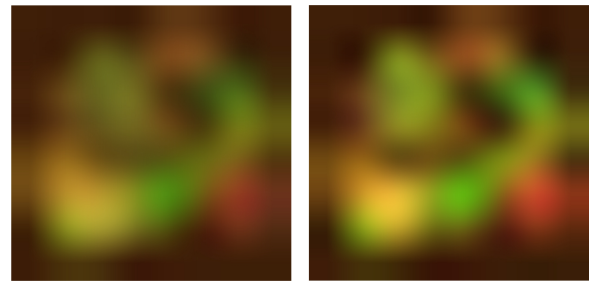

Figure 8: *Comparison between minification with our LoD algorithm (left) and with mipmapping (right).*

adaptation. 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. Figure 8 (left) shows the output of our algorithm for the case shown in Figure 7, for an output image footprint of the merged cell of 8×8 pixels. The 8×8 image is shown magnified for illustration purposes. The result is comparable to mipmapping of actual texture (right).

6. Just-in-time texture mapping

The synthesized texture is encoded using a 1-D array of base umbrellas, a 1-D array of modified umbrellas, and a 2-D array 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, and with per vertex color and position. The arrangement grid stores an array of modified umbrella indices for each cell. This encoding is used directly to texture scene surfaces as required by the output frame.

6.1. Texture lookup

Given a point P with texture coordinates (s, t), the color at P is looked up with the following algorithm.

Find grid cell G that contains P
for all modified umbrellas U_i in G **do**
    Compute angle φ of P with the horizontal axis
    Use φ to find sector triangle T_j containing P
    if P outside peripheral edge e_j **then**
        continue
    end if
    Compute barycentric coords. (α, β, γ) of P in T_j
    Lookup base umbrella texture color c_t at (α, β, γ)
    Compute interpolated vertex color c_v
    return blended final color $c_t + f c_v$
end for
return background color

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 9). 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. The interpolated color is found using Equation 1. If no modified umbrella covers P, the background color is returned.
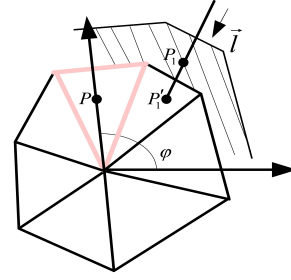

Figure 9: *Identification of umbrella triangle containing lookup point and approximate shadow computation.*

6.2. Shadows

Often the synthesized texture is called upon to approximate surface detail that is not truly flat. Consequently the visual quality of the resulting texture is greatly improved if neighboring umbrellas cast shadows on each other. We compute approximate shadows on the fly with a small addition to the texture lookup algorithm above.

As the umbrellas of the grid cell are traversed in front to back order, not only do we check whether the lookup point is inside the current umbrella, but we also test whether the 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 9 P_1 is translated along the light vector l to P_1' which is inside the umbrella thus P_1 is in shadow. The shadow of multiple umbrellas are aggregated and the returned color is modulated accordingly. Figure 10 shows the same texture fragment with and without shadows.

6.3. Level of detail

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,

Figure 10: *Fall leafs texture with (left) and w/o shadows (right)*

Figure 11: *Additional texture synthesis examples and corresponding base umbrellas.*

the coarser LoDs of the arrangement grid will be used. The lookup algorithm is run on two adjacent LoDs that bracket the desired LoD, and a linear interpolation produces the final color, similarly to the trilinear interpolation of conventional mipmapping. The algorithm provides a smooth transition between LoDs (Figure 2 and video).

7. Results and Discussion

We have applied our technique to generate and use several textures: *Fall Leafs* (Figure 1), *Berries* (Figure 11, row 1), *Pepper* (row 2), *Green Leafs* (row 3), and *Flowers* (row 4). The resolution of the base umbrella textures is 256×256, which allows zooming in with good detail. Our LoD algo-

rithm provides quality results even at extreme minification rates. As illustrated in the video, our technique is stable which preserves quality in sequences of frames. Umbrella edges are antialiased using conventional multisampling.

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 $1 + 3v$ words, which accounts for the base umbrella index and for the positions and colors of the vertices. Let us assume that the arrangement grid has a resolution of $W \times H$ and that there are at most k modified umbrellas per grid cell. Each grid cell records the modified umbrellas it stores with k integer indices. The overall cost in four-byte words J of the just-in-time texture encoding is thus

$$J = bwh + 2bv + n(1 + 3v) + kWH \qquad (2)$$

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 umbrella 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. 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 non-lossy compression 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.

So far the storage cost analysis was conducted under the

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 12: *Rendering performance variation with k (top, v = 77), and v (bottom, k = 13) for Fall Leafs.*

assumption that texel data is stored uncompressed. This precludes any loss of quality due to compression and avoids decoding costs. Texel data compression can be used with just-in-time texture synthesis by compressing the base umbrella textures. Compression benefits just-in-time texturing less than it benefits conventional texture synthesis, because the just-in-time encoding is already a compressed representation. Even so, for example for *Fall Leafs*, when both the base umbrella textures and the conventional synthesized texture are stored in jpeg format with a quality factor of 50%, just-in-time texture synthesis still achieves a storage reduction factor of 3:1.

7.2. Rendering Performance

Just-in-time texture synthesis achieves storage savings by shifting the texture expansion from pre-processing to run-time. This is a classic tradeoff between storage and computation cost. 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 12 shows the variation of the rendering performance with k and v for *Fall Leafs*. The output image resolution is 512×512, $W \times H = 32 \times 32$, and $w \times h = 256 \times 256$. Rendering was done with 4x multisampling and with the approximate shadow algorithm described. The shadow algorithm brings only a small frame rate penalty of under 2 Hz. Performance was measured on an Intel Core i5-760 2.80GHz PC with an NVIDIA GeForce GTX 470, 1,280 MB graphics card.

7.3. Limitations

Even though the umbrella is a versatile geometry and color representation, not all texture elements can be modeled efficiently with umbrellas (e.g. grass blades). Another limitation is that the LoD hierarchy is built using the convex hull, which increases the size of markedly concave structures. Also 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. Finally, rendering performance of just-in-time texture synthesis decreases as
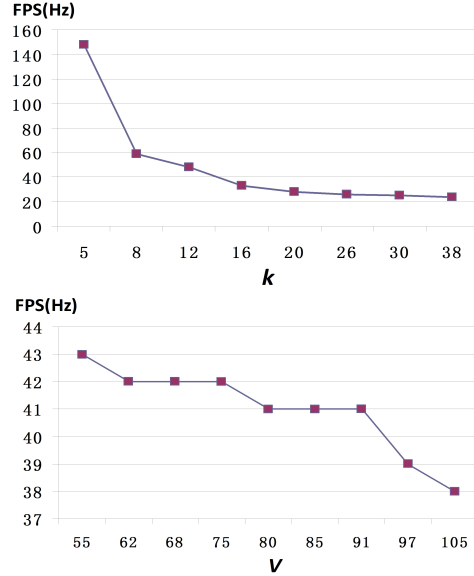
the overlap factor increases, and is lower than for conventional texture mapping.

8. Conclusions and Future Work

We have presented a novel texture synthesis approach that models texture elements from nature with umbrellas, a versatile representation with parameterizable color and shape. A small number of base umbrellas are sufficient to mimic the diversity present in nature. The texel data is computed just-in-time, which brings substantial storage savings.

Just-in-time texture synthesis is a hybrid method in between polygonal rendering and texture mapping. Like in the case of texture mapping, most of the surface detail stems from color maps, and the level of detail is adapted in quadtree fashion as a pre-process. Like in the case of polygonal rendering, the color information is scaffolded using triangle meshes. However, all umbrellas are coplanar in the texture domain, they are sorted implicitly and not through z-buffering, and color is looked up inside an umbrella by finding the relevant triangle directly, as opposed to by rasterizing or ray tracing the triangles of the umbrella.

Our paper focuses on the texture synthesis sub-problems of diversification, LoD adaptation, storage, and runtime lookup, which are directly affected by our just-in-time strategy. Our method can be readily integrated with prior solutions for other texture synthesis sub-problems such as distortion-free tiling on 2-D and 3-D domains, and automatic extraction of texture elements and of arrangement patterns.

Another future work direction is to increase the model-

ing power of the umbrella even further. One possibility is to remove the 2-D limitation. A concave or convex 3-D umbrella could model dried-out leafs, flowers, or berries with increased fidelity without a substantial additional cost. Another possibility is to increase the *color* modeling capability of the umbrella 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. 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 structures it replaces.

Just-in-time texture synthesis takes advantage of the programmability sophistication of modern graphics hardware by essentially introducing 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.

References

[BAC96] BEERS A., AGRAWALA M., CHADDHA N.: Rendering from compressed textures. In *Proceedings of the ACM SIGGRAPH '96* (1996), pp. 373–378.

[BIP00] BAJAJ C., IHM I., PARK S. H.: Compression-based 3D texture mapping for real-time rendering. In *Graphical Models-Pacific Graphics '99* (2000), vol. 62, pp. 391–410.

[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., GROS-JEAN J.: Modeling autumn sceneries. In *Eurographics(EG) short paper* (2006).

[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. In *The Visual Computer* (2006), vol. 22, pp. 926–935.

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. In *SIGGRAPH 2001, Computer Graphics Proceedings* (2001), pp. 341–346.

[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* (2005), pp. 509–513.

[KE02] KRAUS M., ERTL T.: Adaptive texture maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware* (2002), pp. 1–10.

[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.

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

[LLH04] LIU Y. X., LIN W. C., HAYS J. H.: Near-regular texture analysis and manipulation. *ACM TOG 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 TOG 20*, 3 (July 2001), 127–150.

[LP00] LEFEBVRE L., POULIN P.: Analysis and synthesis of structural textures. In *Graphics Interface* (2000), pp. 77–86.

[LRC*02] LUEBKE D., REDDY M., COHEN J., VARSHENEY A., WATSON B., HUEBNER R.: *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2002.

[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.

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

[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.

[Pag04] PAGET R.: Strong markov random field model. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 3 (2004), 408–413.

[PTMG08] PEVRAT A., TERRAZ O., MERILLOU S., GALIN E.: Generation vast varieties of realistic leaves with parametric 2gmaps l-systems. In *The Visual Computer* (2008), vol. 24, pp. 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* (2004), pp. 391–397.

[RHDG10] RISSER E., HAN C., DAHYOT R., GRINSPUN E.: Synthesizing structured image hybrids. *ACM TOG 29*, 4 (2010), 85:1–85:6.

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

[SR06] STACHERA J., ROKITA P.: Gpu-based hierarchical texture decompression. In *Eurographics '06* (2006).

[STC09] SUN C. H., TSAO Y. M., CHIEN S. Y.: High-quality mipmapping texture compression with alpha maps for graphics processing units. In *IEEE Transactions on Multimedia* (2009), vol. 11, pp. 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* (2008), vol. 5359, pp. 1076–1085.

[Wil83] WILLIAMS L.: Pyramidal parametrics. *SIGGRAPH '83, Proceedings of the 10th annual conference on Computer graphics and interactive techniques 17*, 3 (1983).

[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* (2009).

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

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