Crowd Simulation via Texture Synthesis

Aaron Curtis*
UMBC

Abstract

This paper presents a crowd simulation as a novel application of texture synthesis. The motion of autonomous agents is simulated using arbitrarily complex behavior, and captured to a 3D (i.e. animated) texture. The simulation is then expanded to a much larger number of agents using texture synthesis techniques. We make use of *corner cubes* for the synthesis, which are an extension to 3D of corner tiles as described by [Lagae and Dutré 2006], and are similar to Wang cubes.

Corner cubes have an advantage over Wang cubes in that they enforce continuity with all 26 of their neighbors in 3D space; Wang cubes are only continuous with their neighbors in the six cardinal directions, and are as a result subject to the "corner problem". However, both corner cubes and Wang cubes offer a fast, effective method of aperiodically tiling 3D space.

Keywords: texture synthesis, Wang tiles, corner tiles, crowd simulation

1 Introduction

Texture synthesis is perhaps best described as an attempt to generate a texture that is perceptually similar to a given example. (In the context of this paper, we will ignore techniques that generate new textures from scratch.) Typically, the generated texture is also larger than the example. The simplest way to create a larger texture from an example is to repeatedly tile the example, however this tiling is often clearly visible in the output and detracts greatly from the image quality. Texture synthesis algorithms aim to avoid such artifacts, creating textures that could believably have come from the same stochastic process as the example, but that are clearly different from it.

It should be noted at this point that synthesis techniques are most often applied to textures with an underlying stochastic nature. The macroscopic appearance of materials such as stone, wood, paper, or skin is governed by effectively random processes, and the goal of texture synthesis is to capture the essence of these processes. However, synthesis is applicable to domains other than generating material textures with a particular appearance - anything with an underlying stochastic nature is a candidate for application of synthesis algorithms. For example, [Cohen et al. 2003] used texture synthesis to generate Poisson distributions, which govern the growth of plants.

In a similar vein, this paper applies texture synthesis to the behavior of crowds. An individual in a crowd will follow deterministic behavior based on goals, interactions with his peers or his environment, or other arbitrarily complex motives. When viewed on a large enough scale however, the deterministic nature is hidden, and the behavior of any single individual appears effectively random, with only large-scale trends being visible. Crowd simulation can therefore be seen as a good candidate for texture synthesis.

We simulate the behavior of a small crowd using deterministic agents, and capture this behavior in a 3D texture (i.e. one that varies in two spatial dimensions as well as time). We then synthesize a larger crowd using corner cubes, which are an extension to 3D of [Lagae and Dutré 2006]'s corner tiles. Corner tiles are themselves a variant of Wang tiles [Cohen et al. 2003], which have previously been extended to 3D as Wang cubes [Sibley et al. 2004]. The greater portion of this paper is given to describing the implementation of corner cubes and the issues encountered therein.

1.1 Related Work

1.1.1 Texture Synthesis

Texture synthesis has been the subject of research for over a decade, and wide variety of techniques for it have been developed, but they can generally be grouped into two classes. First are pixel-based algorithms, which attempt to match a pixel in the source image to a neighborhood in the output (which may only be partially generated). One of the earliest of these algorithms was introduced by [Efros and Leung 1999]. Pixel-based techniques have a disadvantage of being very slow, and consequently later research centered around ways to accelerate the process. [Wei and Levoy 2000] used vector quantization to reduce the time required to search for neighborhood matches. [Zelinka and Garland 2004] alternatively proposed the use of *jump maps* for similar purposes.

A second class of algorithms works by copying large patches of the input texture into overlapping sections of the output. This type of approach is generally faster than pixel-based synthesis, but also has the advantage of better capturing large scale structures in the input texture. The first of these algorithms, such as that of [Liang et al. 2001] worked by blending the overlapping regions of the patches, however, later techniques were able to give more convincing results by finding an optimal subset of the pixels in each patch to carry over to the output. Image quilting [Efros and Freeman 2001] and graphcut textures [Kwatra et al. 2003] are prime examples.

1.1.2 Wang Tiles

The use of Wang tiles for texture synthesis was described in detail by [Cohen et al. 2003], who used the tiles to construct Poisson distributions in addition to textures. In brief, Wang tiles are textured tiles with "colored" edges, specially constructed so that tiles with matching edge colors can be placed next to one another without visible artifacts at the edges. An arbitrarily large texture can be built from a set of Wang tiles by traversing a grid in scan-line order and selecting a tile for each grid cell, subject to the constraint that its edge colors must match those of its already-placed neighbors. As long as the tile set is large enough that at least two choices exist

*e-mail: acurti1@umbc.edu

when placing any tile, the generated texture is assured to be aperiodic.

The tiles themselves are generated through other texture synthesis techniques. Cohen, et al.'s approach was to quilt (as in [Efros and Freeman 2001]) four samples of an input texture together in a diamond shape, and take the center square as a single Wang tile. One might imagine image quilting easily being replaced by similar methods, such as graph cuts [Kwatra et al. 2003] or improved image quilting [Long and Mould 2007]. (For lack of a better term, we will call all such methods "quilting").

[Wei 2004] demonstrated direct texturing of 3D models in real time using Wang tiles. Rather than generate a large texture by placing tiles in scan-line order, Wei used a hash function in a GPU shader to determine the appropriate tile, given texture coordinates in random order. This necessitated generating a complete set of Wang tiles, i.e. one tile for every possible combination of edge colors; Cohen et al. required far fewer than this number. Wei also developed an arrangement for packing the tiles into a single texture for access on graphics hardware, such that the edges of the packed tiles had matching colors; this avoids artifacts when mip mapping is used.

Wang tiles are readily extensible to 3D space, as demonstrated by [Sibley et al. 2004], who used *Wang cubes* for video synthesis and geometry placement, with time being the third dimension in video synthesis. A difficulty arises in that the image quilting approach used by Cohen, et al. does not have an obvious extension to 3D, as it finds a minimum-error *path* through the overlapping texture samples. What is required instead, is a minimum-error *surface*; [Kwatra et al. 2003]'s graph cut technique can provide this. (Kwatra et al. had investigated video synthesis in their work using graph cuts alone.)

1.1.3 Corner Tiles

Cohen et al. identified a potential problem with Wang tiles, in that the a tile is under no constraints to match its diagonal neighbors. This can lead to artifacts at the tile corners. Although Cohen et al. addressed the "corner problem", it was more elegantly solved by [Ng et al. 2005], who proposed coloring the corners of the tiles instead of the edges. [Lagae and Dutré 2006] described this technique in more detail, and also developed a packing arrangement for corner tiles, similar to that of [Wei 2004] for Wang tiles. ([Ng et al. 2005] refer to their scheme as omega tiles, but we will use Lagae and Dutré's term.)

The techniques for generating Wang tiles apply similarly to corner tiles; one can arrange four samples of the input texture in a square with overlapping seams, then cut a square tile out of the center. [Dong et al. 2007] showed that the sample selection can be optimized using genetic algorithms.

2 Implementation

2.1 Generating Corner Cubes

[Sibley et al. 2004] frames the generation of each Wang cube as a minimum graph cut problem [Kwatra et al. 2003], and solves it using the Ford-Fulkerson method for network flow. Corner cubes are created similarly, and the difference is analogous to the difference between generating Wang tiles and corner tiles. This difference is shown pictorially in Figure 1. Kwatra et al. make use of work by [Max 2004] in their implementation, and we do the same.

Cohen et al.'s method of generating Wang tiles involves selecting a single patch from the input texture for every possible edge color. To generate a single Wang tile, four such patches are arranged in a diamond shape, with an overlap region between them. The four patches are then quilted together and a square tile is extracted from the center; this yields a Wang tile with edge colors corresponding to the colors of the original four patches.

This technique can be easily adapted to corner tiles, resulting in the arrangement shown in Figure 1 (b); the four patches from the input are placed in a square, such that each corner of the output tile is centered on a single patch. [Lagae and Dutré 2006] modify this approach by eliminating the overlap regions, and instead placing additional patches over the center of the square, to be merged in with graph cuts. This ensures that there is at least one unique patch for each generated tile, and increases the total number of samples taken from the input texture. The final scheme is shown in Figure 1 (c).

It should be noted that Lagae and Dutré's method can lead to point defects in the generated tiles, as shown in Figure 1 (d). The four samples from the input texture meet at the center of each edge, but are not merged in any special way. Single-pixel discontinuities are a very minor problem in the 2D case, but these defects become lines when moving to 3D. Therefore, we combine approaches (b) and (c) - the corner patches are quilted together using graph cuts, and then another patch is added to the middle of the combined texture. The 2D analog of our approach is shown in Figure 1 (e). Because of the overlap region, the initial samples must be somewhat larger than the extracted tile. We typically extend them by 50

Sibley et al. adapt Wang tiles to three dimensions by taking octahedral patches from the input texture and quilting six of them together to form a larger patch, from which a single Wang cube is taken. This arrangement resembles Figure 1 (a) when viewed from one of the cardinal directions.

We follow a similar approach in adapting Lagae and Dutré's method to create corner cubes. Figure 2 shows how this is done. First, eight cubic patches are sampled from the input texture and arranged to form a single, larger cube, shown in (a). There is one unique sample cube for each of the possible corner colors. Two graph-cutting steps follow, as shown in Figure 2 (b-c): an additional arbitrary patch is merged into the center of the cube, and the eight sample cubes are quilted together. Finally, a single corner cube is extracted from the center, as shown in Figure 2 (d).

For the central patch, the graph cutting problem is set up such that voxels outside of a fixed radius are guaranteed to come from the initial eight patches, i.e. the cutting surface is bounded by a sphere. We use the square of the L_2 metric between pixel values to determine the edge capacities in the graph, rather than the L_2 metric as is normally used. [Long and Mould 2007] suggests that better results can be obtained by favoring multiple small pixel differences over a single large one, and in any case the square of the metric is cheaper to compute.

As in other applications of patch-based synthesis, the patches do not always quilt together well, and there are some scenarios where visible defects are unavoidable. Therefore, when applying the central patch, we take the best result from multiple different patches, chosen randomly. The best result is determined by the result of the graph cutting algorithm - lower values for the maximum flow are better. We generally do not use more than four trial patches for this purpose due to computational expense. The initial samples may also have visible seams where they are quilted together, but these are in most cases overwritten by the central patch.

To perform the actual synthesis, multiple cubes are arranged in a 3D array, such that their corner colors match. The simplest way to do this is in scan-line order - the first cube is generated randomly, then the cube on its right is determined, with the unconstrained corners assigned randomly, then the next cube to the right is determined,

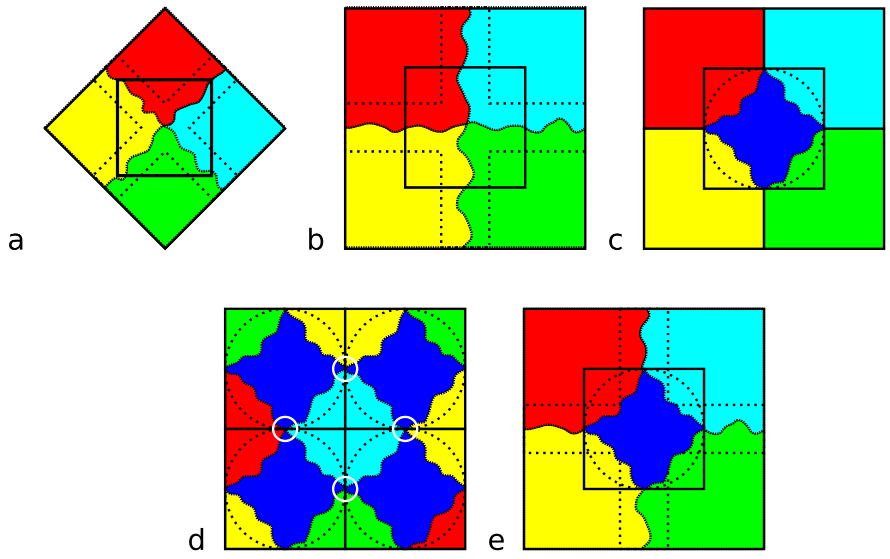


Figure 1: *(a) Wang tiles. Four texture samples are quilted together and a tile is extracted from the center. (b) Simple formulation of corner tiles, analogous to (a). (c) Lagae and Dutré's approach to corner tiles. Four texture samples are arranged and a fifth sample is quilted on top of them. (d) Four corner tiles created using method (c). Potential error regions are circled in white. (e) The approach used in this paper is a combination of (b) and (c), when reduced to two dimensions.*

and so on until a wide enough output is reached. Another line of cubes is then written with one side constrained by the previous line, and so on until an entire plane of cubes is generated. Another plane is then generated constrained by the previous one, and so on for an arbitrary depth.

In general, any cube will have three of its faces constrained by existing cubes (the exceptions being the first plane, the first line in any plane, and the first cube in any line). The three constrained faces correspond to seven constrained corners, leaving one randomly assigned corner for most cubes. Since the constrained corners can have any color, the synthesis algorithm must be able to provide c^7 different cubes, if there are c colors.

Furthermore, since the synthesized texture should be non-repeating, there must always be a choice of at least two possible cubes in any situation. This increases the required number of cubes to $2c^7$. For useful values of c (i.e. 2 or 3), this is not much better than requiring a full set of cubes, that is, one cube for every possible combination of corner colors (c^8). Note that, given the way we generate cubes, two cubes with the same corner colors need not be the same; hence it is possible to generate *more* than a full set.

For values of c larger than 2, generating a full set of cubes quickly becomes infeasible. However, note that in some situations the number of cubes desired for the output is much smaller than the "required" number of cubes. For this reason, we generate cubes on-demand as they are needed for the output and cache them for later use, rather than generating all the cubes in advance. Some results for $c = 4$ are presented in a later section. Most often, however, we use the more practical value of $c = 2$.

2.2 Crowd Simulation

2.2.1 Initial Simulation

The crowd is modeled as a collection of agents, each of which acts independently. Any given agent has a state associated with it, for example REST, WANDER, FLEE, etc. In the REST state, the agent does nothing, in the WANDER state the agent moves slowly through the environment, or in the FLEE state the agent moves quickly. One can imagine various other more nuanced states, but these are sufficient for an example. Furthermore, agents transition between states according to a probability distribution, so for example, REST has a high probability of transitioning to WANDER, and WANDER has a high probability of transitioning to a different instance of itself (that is, the agent wanders in a different direction).

Transition probabilities are easily defined in a matrix, for example:

From	**To**			
	DEAD	REST	WANDER	FLEE
DEAD	0	0	0	0
REST	0	0	0.2	0.01
WANDER	0	0.04	0.08	0.01
FLEE	0	0.02	0.14	0.05

Interesting large-scale behavior arises when an agent's environment is allowed to affect the transition probabilities. The simplest example is that agents that wander out of the bounds should have their chances of entering the DEAD state increased to 1. We have implemented a few other rules that involve agent-agent interaction:

- If an agent is in the FLEE state, other nearby agents have an increased chance of entering the FLEE state as well.
- If an agent is in the WANDER state and there is a resting agent in front of it, the first agent has an increased chance of

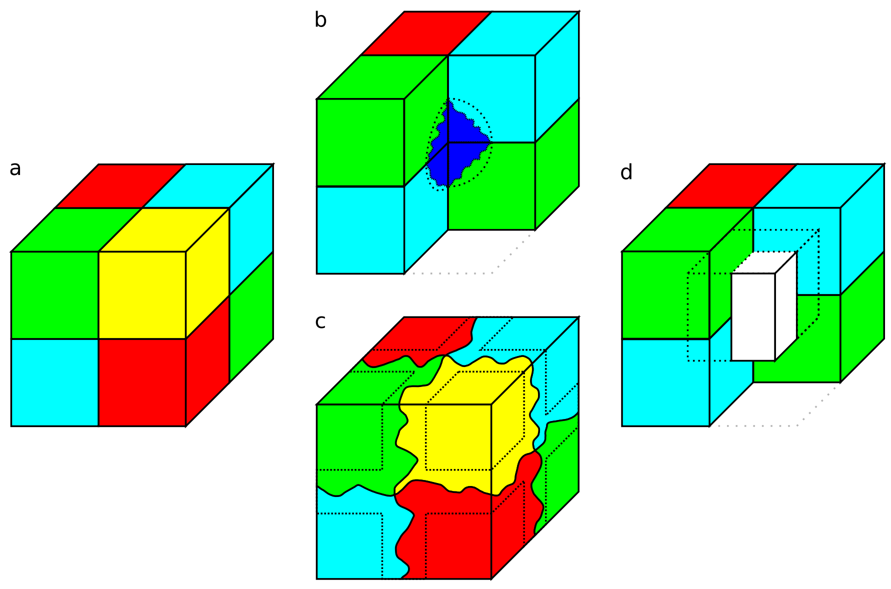


Figure 2: *(a) Corner cube generation begins with eight sample cubes. (b) A ninth is quilted over top of the samples from (a). This is a direct extension of Lagae and Dutré's method to three dimensions. (c) The eight samples are also quilted together (done in addition to (b), but shown independently here). (d) The corner cube is extracted from the center of the samples.*

changing its direction.

- Similarly, if an agent's path is blocked by another wandering agent, the first agent has an increased chance of transitioning to the REST state.

2.2.2 Capture to Texture

The simplest way to capture the crowd's behavior is just to render images of it at successive time steps. We can do this, for example, by drawing every agent as a black pixel on a white background. The resulting images are not be very visually appealing, but can be fed directly into a texture synthesis algorithm. In effect, we are then synthesizing *pictures* of crowds, rather than the actual behavior.

3 Results and Discussion

Figure 3 shows the result of the synthesis algorithm on an ordinary texture. Since our method requires a 3D texture as input, we used a stack of 48 copies of the olive texture. For this instance, the algorithm was run with $c = 4$, meaning there are four *base* samples (32x32 pixels each) taken from the input texture. The output is a 4x4 arrangement of 16 32x32 cubes. Each cube here is unique - this is a case where the output uses far fewer than the full set of cubes.

Some defects are visible, where the samples used to create the cubes did not mesh well. However, it is worth noting that the defects always occur in the interior of the cubes; our method of cube generation ensures that they always tile seamlessly. [Kwatra et al. 2003] uses the same input texture with generally better results, as their algorithm is much less restricted in its placement of new patches.

Figure 4 shows a few slices of texture from the initial crowd simulation, followed by Figure 5, which shows a few slices from the synthesis results. Sample sizes are the same as before, but the algorithm was run with $c = 2$. The motion of the agents in the synthesized result is generally coherent, but some discrepancies exist in the animation that are not apparent in the still images - agents do occasionally disappear and reappear without cause.

However, the larger problem is the obvious repetition in most of the frames - the scenes looks like they were generated with repeating tiles, which we had hoped to avoid. Recall that when generating a cube, the final sample merged into the center is bounded by a sphere. This means that if we look at slices of the cube along the vertical axis, slices near the middle of the cube will contain a large portion of the final sample, and slices near the top and bottom will contain very little of it.

Hence animation frames that are nearly a multiple of the cube size will be composed almost *entirely* of the two initial samples from the input (i.e. one for each of the two colors), with none of the per-cube samples added by the graph cut procedure. Two samples are clearly not enough to convincingly tile the output. Conversely, frames that are offset by half the cube size (here, frames 16, 48, 80, etc.) can have a large portion of unique per-cube samples, and do not show as much evidence of tiling. Hence in the best case, the animation alternates between looking tiled and looking random.

In the worst case, the graph cut procedure does not actually add any new sample data to the cubes, i.e. it finds the minimum error can be obtained by only copying the minimum number of pixels from the central patch (the pixels overlapping the seams between the initial samples are required). This scenario is actually fairly likely given the nature of the textures being used - large portions of it are nothing but white space.

4 Future Work

4.1 Increasing the number of samples

It is evident from the results that a better method of cube generation is needed. Most important is introducing a better distribution of

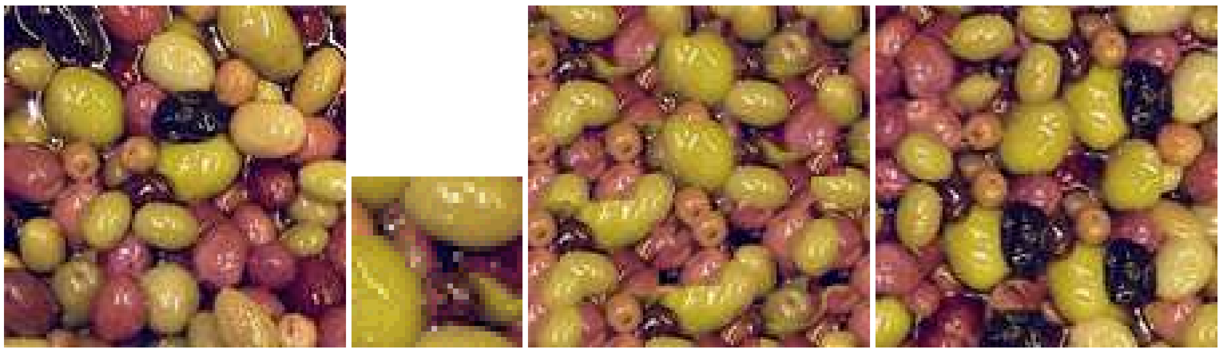

(a) Input texture (b) Single tile (scaled) (c) Synthesis result (d) Kwatra et al. (cropped)

Figure 3: *Corner cube algorithm run on a 2D texture.*

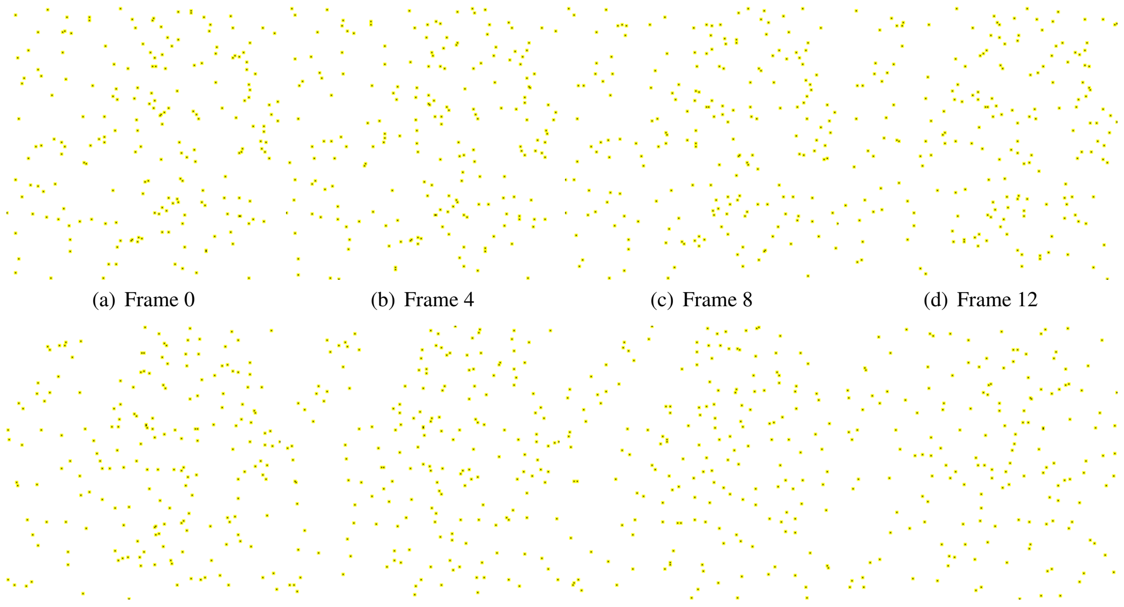

(a) Frame 0 (b) Frame 4 (c) Frame 8 (d) Frame 12 (e) Frame 16 (f) Frame 20 (g) Frame 24 (h) Frame 28

Figure 4: *Sample frames from the initial crowd simulation. 200 agents were used.*

samples from the input, though this is difficult because the number of corner colors is so severely limited. Lagae and Dutré's method of adding a unique patch to the center of each tile/cube is a good start, but it leaves far too much of the cube untouched to effectively address the problem in 3D.

One possibility is to introduce additional per-face samples - imagine the extra sample from Figure 2 (b) shifted out to one of the cube faces. Care would then need to be taken to ensure that cubes with identical corner configurations share the same samples. Since there are c^4 possible face configurations, this approach would introduce 16 new samples in the two-color case (or 48, if each axis is treated independently).

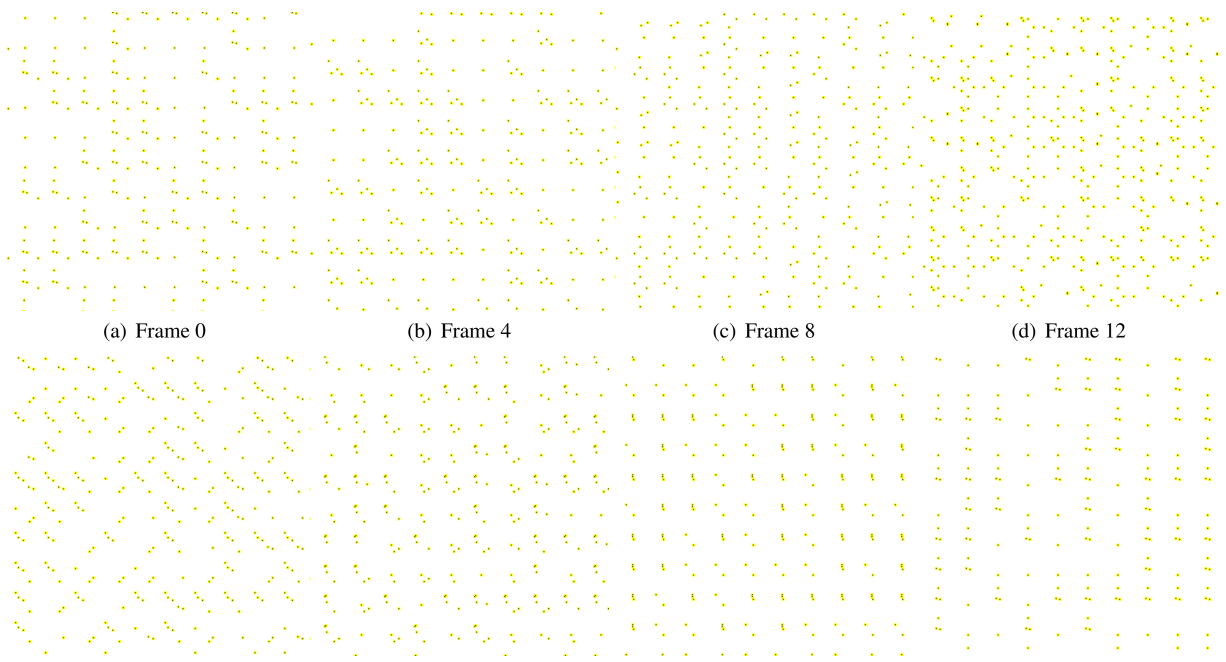
Of course, such a method would begin to resemble Wang tiles, and we would be left wondering why we bothered to color the corners at all.

4.2 Higher quality synthesis

Another area in which our implementation is lacking is patch selection. In their original graph cutting paper, Kwatra et al. devote much effort to choosing patches from the input that will work well together, using FFT-accelerated search through texture space to find close-matching patches. Random selection (as used in this paper) is only left for the most basic textures. Furthermore, researchers such as [Dong et al. 2007] have directly addressed patch selection for Wang tiles (they used genetic algorithms). Incorporating these techniques could lead to much higher quality synthesis.

4.3 Deterministic cube selection

In their work, Lagae and Dutré introduce a hash function to deterministically find the corner colors of an arbitrary tile, i.e. without checking any of its neighbors ([Wei 2004] does the same for Wang tiles). This makes it possible to effectively synthesize texture directly on graphics hardware, but requires generating a full set of


(a) Frame 0 (b) Frame 4 (c) Frame 8 (d) Frame 12

(e) Frame 16 (f) Frame 20 (g) Frame 24 (h) Frame 28

Figure 5: *Results the from texture synthesis algorithm, which generated a 10x10x16 array of cubes, each 32x32x32 pixels. Two possible corner colorings were used.*

tiles (c^4 in their case).

Since we typically generate a full set of corner cubes anyway, it would make sense to implement a hash function for deterministic cube selection. When applied to crowd simulation, this would allow (for example) users to skip forward in the simulation by arbitrary time steps.

4.4 Improved texture capture

The synthesized animation that results from our technique shows the high-level behavior of the crowd, but does not allow any direct analysis of individual agents. Ideally, we would like to extract the locations of the agents after running the synthesis algorithm, then store them in some other data structure that might be more suitable (e.g. a list of coordinates). This would make various operations on the data much easier, including for example, re-rendering the agents as 3D models.

To do this, it would probably be necessary to store separately a texture representing the *appearance* of the crowd, and another representing the actual presence of agents (either a true of false flag at each location). The first texture could be tweaked to best work with existing synthesis algorithms (for example, by using a more diverse range of colors), and synthesis would be performed on it alone.

Whenever patches of the first texture are copied to the output, corresponding patches from the second should be copied as well. Since the resulting cubes would consist of just true/false flags, it would be a simple matter to scan through them linearly and construct a list of all agent locations.

5 Conclusion

This paper has shown that crowd simulation can be recast as a texture synthesis problem. It has also shown that [Lagae and Dutré 2006]'s corner tiles can be extended to 3D to form corner cubes. Although generating corner cubes can be computationally expensive, once created they provide an easy method for generating arbitrary amounts of 3D texture.

What has not been shown is whether or not corner cubes can be used to synthesize texture of high enough quality to be useful for the purpose of simulating believable crowds. When extended to 3D, Lagae and Dutré's approach has severe disadvantages that were either not apparent in two dimensions, or were more easily avoidable. While some of these problems have been addressed, others (particularly undersampling of the input texture) require further work.

References

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. *ACM Trans. Graph. 22*, 3, 287–294.

DONG, W., ZHOU, N., AND PAUL, J.-C. 2007. Optimized tile-based texture synthesis. In *GI '07: Proceedings of Graphics Interface 2007*, ACM, New York, NY, USA, 249–256.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 341–346.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, IEEE Computer Society, Washington, DC, USA, 1033.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph. 22*, 3, 277–286.

LAGAE, A., AND DUTRÉ, P. 2006. An alternative for wang tiles: colored edges versus colored corners. *ACM Trans. Graph. 25*, 4, 1442–1459.

LIANG, L., LIU, C., XU, Y.-Q., GUO, B., AND SHUM, H.-Y. 2001. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph. 20*, 3, 127–150.

LONG, J., AND MOULD, D. 2007. Improved image quilting. In *GI '07: Proceedings of Graphics Interface 2007*, ACM, New York, NY, USA, 257–264.

2004. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell. 26*, 9, 1124–1137. Member-Yuri Boykov and Member-Vladimir Kolmogorov.

NG, T.-Y., WEN, C., TAN, T.-S., ZHANG, X., AND KIM, Y. J. 2005. Generating an /spl omega/-tile set for texture synthesis. In *CGI '05: Proceedings of the Computer Graphics International 2005*, IEEE Computer Society, Washington, DC, USA, 177–184.

SIBLEY, P. G., MONTGOMERY, P., AND MARAI, G. E. 2004. Wang cubes for video synthesis and geometry placement. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Posters*, ACM, New York, NY, USA, 20.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 479–488.

WEI, L.-Y. 2004. Tile-based texture mapping on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, ACM, New York, NY, USA, 67.

ZELINKA, S., AND GARLAND, M. 2004. Jump map-based interactive texture synthesis. *ACM Trans. Graph. 23*, 4, 930–962.