Computational Vision: Example-Based Texture Synthesis

Matthew McMullan

December 11, 2012

1 Overview

Our project focused on the creation of large regions of texture from small example textures. We accomplish this using a method similar to the one described in Parallel Controllable Texture Synthesis, computing our results using a GPU implementation. The general idea is to scale the image up (without interpolation), apply jitter (randomly shift regions of the texture in the image), and to correct the jittered texture by comparing to neighborhoods of the exemplar.

1.1 Data Sources

2 Algorithm

As described in [2], there are three main steps to our implementation: upsample, jitter, and correction. Upsampling takes the example texture and creates a larger version, with the resulting texture storing coordinates to values in the exemplar. This texture of coordinates is then used to upsample further in later iterations. To achieve the upsampling, first the size of the texture is doubled, and the current pixel first creates an extra row and column to the right and down to accomodate the additional pixels before propagating its stored coordinate values to create a larger region containing the value. This, when used with multiple passes, causes the image to be doubled in size while retaining only the color and intensity data available in the exemplar.

We then apply jitter, using simplex noise. The generated noise results in a deterministic texture. This introduces some randomness to reduce the tiled appearence of the texture. Simplex noise is also spatially coherent, resulting in a jittered texture with shifted patches instead of pixel by pixel as would occur with a different algorithm such as a Mersenne twister.

Before performing the correction step, we perform an additional step to utilize k-coherence to find a variety of neighborhoods to correct to. For each neighborhood in the synthesized image, we look at each pixel in the neighborhood and find similar neighborhoods in the exemplar by calculating and comparing neighborhood distances as a summed squared Euclidean distance over the neighborhood. The best k choices are then stored and passed along to the correction step. For our purposes, we used $k = 4$. [1, 3]

To correct, we look at each of the neighborhoods found from k-coherence and alters the synthesized neighborhood to match the nearest neighborhood from the exemplar.

3 Implementation

4 Results

4.1 'Good' Cases

4.2 'Bad' Cases

5 Conclusion

References

[1] Michael Ashikhmin. Synthesizing natural textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, I3D '01, pages 217–226, New York, NY, USA, 2001. ACM.

[2] Sylvain Lefebvre and Hugues Hoppe. Parallel controllable texture synthesis. *ACM Trans. Graph.*, 24(3):777–786, July 2005.

[3] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, and Heung-Yeung Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Trans. Graph.*, 21(3):665–672, July 2002.