Malcolm Wetzstein

6.865

5/20/19

Implementation of “Adaptive Manifolds for Real-Time High-Dimensional Filtering”

1. I chose to implement the paper the algorithm proposed by the paper “Adaptive Manifolds for Real-Time High-Dimensional Filtering”. The algorithm is a generalized method for computing a fast and accurate estimate of any high-dimensional filter with compact closure (roughly speaking in layman’s terms, the domain of kernel function is a bounded region, such as a truncated gaussian). The filter may operate of the domain as well as the range of an input signal. An example of such a filter is the 5D Bilateral filter, which filters over x and y pixel coordinates as well as RGB color values. I wrote my implementation such that I can easily change the filter being computed by the algorithm, the definition of the filter is passed via a function pointer. I chose this paper for my project because I have an interest in real-time computer graphics, especially when it comes to creating computationally cheap and dirty tricks that yield convincing and aesthetically pleasing visuals. The original paper shows impressive results for applying this algorithm to denoising real-time path-tracing with low sample count. It’s my instinct that this algorithm can have many more creative applications for cheap real-time rendering effects. I test my implementation of the algorithm by applying it to the bilateral filter and to a non-local means filter. I also test the effectiveness of the algorithm at performing a fast, pseudo depth-of-field post-process of my own design that has similar computational requirements as the bilateral filter, showing its usefulness for real-time visual effects.

2. Filters that operate over the range of a signal as well as the domain are non-linear and, if implemented naively, require a lot of computation to convolve with a signal. The general approach to obtaining a fast approximation is to only sample the response of the signal to the filter at a few discrete locations and then smartly interpolate those samples. Several papers have tried to propose fast algorithms for computing the bilateral filter, or even higher dimensional filters, using such a process. “A Fast Approximation of the Bilateral Filter using a Signal Processing Approach” uses an approach similar to the paper I implemented involving manifolds to approximate the bilateral filter. However, theirs is not as accurate because their manifolds are linear and so do not fit the signal’s response to the filter well. “Adaptive Manifolds for Real-Time High-Dimensional Filtering” use manifolds that are only locally linear but shaped to better fit the signal’s response. “Gaussian KD-Trees for Fast High-Dimensional Filtering” uses a KD-tree to distribute the samples for interpolation. “Guided Image Filtering” computes fast high-dimensional filters through dimensionality reduction; pixels are compared indirectly through their relation to a third point. Some of these techniques obtain runtimes for interactive applications, however none of them are fast enough for real-time applications. A major contribution of “Adaptive Manifolds for Real-Time High-Dimensional Filtering” is its real time performance.

3. The algorithm that I implemented approximates the response of a d-dimensional filter to an input signal. The signal has a S-dimensional domain and a R-dimensional range, where S+R = d. We'll refer to it as the adaptive manifold filtering algorithm. There are two major steps to this algorithm, computing a set of sampling points for each adaptive manifold, and computing the filter response from the sampling points. The run time of the algorithm is O(dNk), where d is the dimensions of the filter, N is the number of pixels in the input signal, and k is the number of manifolds used. Parameter k is independent of d and N (depends on the size of the filter).
    a. To compute the sampling points for each adaptive manifold, we compute it recursively. The sampling points for each manifold are distributed over the domain dimensions identically to the input signal, so for each pixel there is a corresponding sampling point. The goal is that each manifold is locally linear, but approximates the input signal well. For each pixel in the input, as long as there is at least one manifold with a sampling point for which the pixel is in close proximity to, we will be able to compute an accurate filter response for that pixel. If this is not the case for a pixel, it is considered an outlier. We only need to recursively compute more manifolds until the majority of pixels are no longer outliers (~95%). If joint filtering, we use the joint image data for manifold computation. For a manifold eta_0, two new manifolds eta – and eta + can be computed by segmenting eta_0 based on the distance from each pixel in the input signal to eta_0. The sampling points for the initial adaptive manifold is computed by using a low pass filter on the input signal that operates only in the domain dimensions. From there, we compute the distance squared from the input signal pixels to their corresponding sampling point. We perform a fast principal component analysis on these squared distances and use the sign of the dot product between the first eigenvector and the squared distances to cluster the pixels into either belonging to eta – or eta +. For each of the two clusters, a low-pass filtering of the input signal that gives zero weight to pixels outside the cluster yields the new manifolds eta – and eta +. When recursing on these manifolds, the PCA computation should only consider pixels inside the parent manifold's cluster. We also exclude any pixels not in the parent manifold's cluster from the clusters of the child manifolds. This method of cluster segmentation allows child manifolds to adapt to pixels from the input signal that were originally outliers from the parent manifold, so that each time we recurse we reduce the overall number of outliers.
    b. Contributions from each adaptive manifold can be accumulated separately into the final filter response. To accumulate an adaptive manifold's contribution, we first project the pixels from the input signal onto their corresponding sample point on the adaptive manifold. If phi is the kernel function for the filter we are approximating, then this projection is done by weighting the input signal by phi of the distance from the input signal to the corresponding sample point. After we have finished the projection, we blur the projected values over the manifold using a "gaussian" filter (fake gaussian in O(N) time). The "gaussian" filter is computed by first performing a domain transform that takes into account the curvature of the manifold in d-dimensional space (the full dimensions of the filter we are approximating). The same blur is also applied to the phi weights that were used to compute the projected input signal in the transformed domain. After blurring the projected input signal over the adaptive manifold, we accumulate the projected input signal, weighting it using the same phi weights that

were used for the original projection, giving us the contribution of the manifold to the final image. We also accumulate the blurred version of the phi weights, weighting them by the original phi weights. Once contributions from all adaptive manifolds have been accumulated, we normalize by the accumulated, blurred phi weights to compute the final image. The results for outlier pixels in the final image are not too noticeable, but for certain applications it may be desirable to weight outlier pixels back toward their original signal value.

4. I was able to implement a bilateral filter, non-local means denoising, and a depth-of-field post-process by supplying a gaussian kernel as a definition for phi and by changing the joint image signal. Bilateral filter is implemented by using the image colors as the joint image signal. Non-local means denoising is implemented by supplying the non-local means basis as the joint signal. Pseudo depth of field is done by supplying a depth map as the joint image data and then linearly interpolating the pixels of the filter result with the original image based on distance from the focal plane. The most difficult thing about implementing the algorithm is that, due to the recursive nature, the output of individual steps are heavily dependent on one another. Not all steps can be isolated during testing, and for those that can, it's hard to tell when the output is correct by looking at it. A wrong implementation of the low pass filter for creating manifolds can still look like a blurry image but completely ruin your final output. These challenges made debugging difficult.

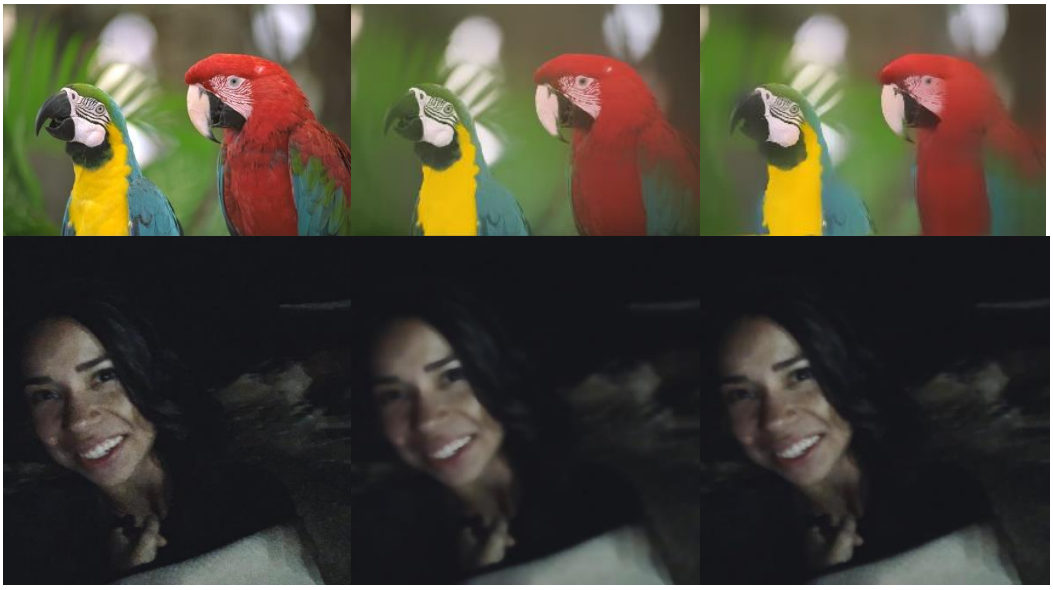
5. Test Cases
    a. Low Pass Filter Constructing First Manifold (input on left, output on right):


    b. Domain Transform Filter for Blurring Manifold (input on left, output on right):


    c. Recursive Clustering of Input Signal (following one branch of the recursive tree; white inclusive to cluster, black exclusive)

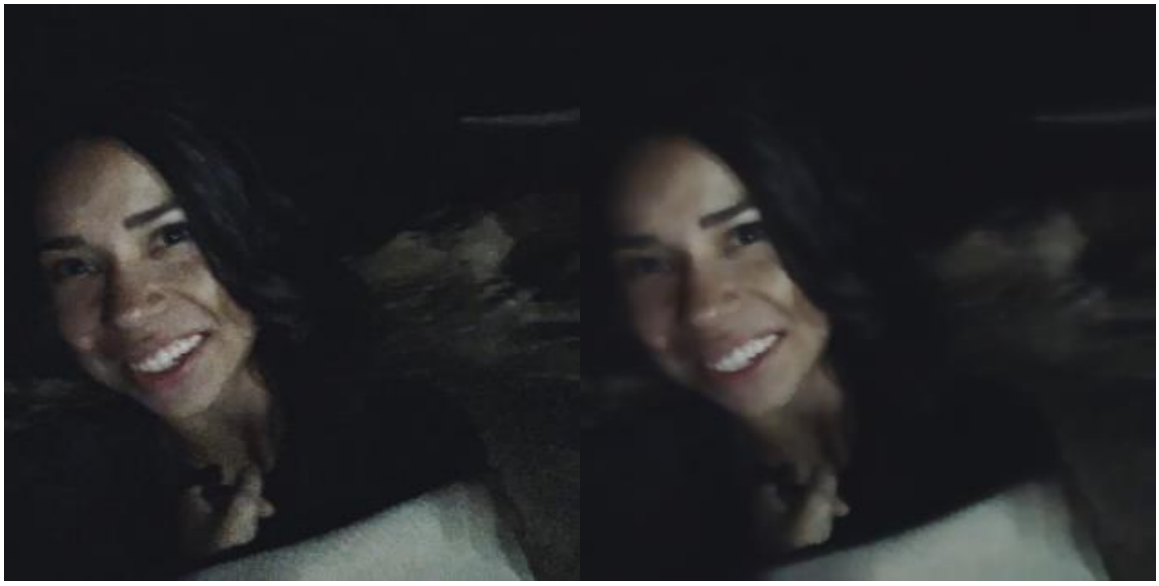

d. Recursive Manifold Construction (following one branch of the recursive tree)


e. Final Image Output (gaussian blur over domain and range dimensions, aka bilateral; input left, naïve bilateral middle, adaptive manifold bilateral right)



6. The performance gains for using adaptive manifold filtering over naïve implementations of high-dimensional filters is quite substantial. 5-dimensional bilateral filtering on a low resolution 640x640 image using the adaptive manifold filtering algorithm takes about 2.16 seconds. The naïve bilateral filter implementation (provided in pset starter code) takes about 52.96 seconds, making the adaptive manifold approach about 25 times faster. For other images with different choices of standard deviations for the filter blur, the speedup can be as much as 1000 times faster. 149-dimensional non-local means denoising with adaptive manifold filtering on the same 640x640 image takes about 125.16 seconds, showing reasonable run time for very high dimensional filters. My depth of field technique on a 1200x900 image takes about 0.39 seconds.

7. Results for Non-Local Means and Depth of Field
    a. Non-Local Means (7x7 neighborhoods, 149-dimensional filter)



    b. Depth of Field (focal distance 0.2 in normalized range of depth map)