A Method for Online Interpolation of Packet-Loss Blocks in Streaming Video

Rumana Aktar, Kannappan Palaniappan, Jeffrey Uhlmann
Department of Electrical Engineering and Computer Science
University of Missouri - Columbia

Abstract—In this paper we examine and apply a linear-time matrix transformation for online interpolation of missing data blocks in frames of a streaming video sequence. We show that the resulting algorithm produces interpolated pixels that are sufficiently consistent within the context of a single frame that the missing block/tile is typically unnoticed by a viewer of the video sequence. Given the strenuous time constraints imposed by streaming video, this is essentially the only standard of performance that can be applied.

1. Introduction

In this paper we examine the use of a linear-time matrix transformation as a basis for an approach to online interpolation of missing data blocks in frames of a streaming video sequence. Unlike conventional in-painting (in-filling) applications, the real-time demands of streaming video tend to constrain the set of feasible algorithms to only those with run-time computational complexity that is linear in the number of pixels to be interpolated. Most such approaches involve simple linear averaging of pixels on the perimeter of the missing block, and the result is a relatively homogeneous blurred patch which tends to stand out strongly even though the frame is visible for only a fraction of a second within the video sequence. Our approach, by contrast, produces a result with sufficient high-frequency detail (texture) to often allow it to go unnoticed by most viewers.

Image inpainting has been subject of interest in computer vision community for long time [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]. Studies have summarized approaches followed in inpainting researches mainly in five categories [23], [24], [25], [1]: structural or partial differential equation (PDE) based, texture-based, exemplar-based, hybrid, and fast inpainting.

Bertalmio et al. [11] proposed the first PDE based inpainting which propagates information from neighboring geometric structures toward isophotes or edge direction. This simple approach works good for small holes but fails to approximate missing data in large textured regions (generating blurring effects). Tschumperle et al. [22] presented a PDE based system where a large set of previous vector-valued regularization approaches were expressed by common local expression. Leveraging the local filtering properties of the proposed equations, it was able to achieve a certain level of accuracy. Another effective inpainting system for recovering sharp edges at a certain level is Total Variation (TV) Regularization [26] which is an adaption of TV model [27]. Though robust to small holes, it also fails to handle large textured area and curved structures. To handle this, Chan et al. [12] extended this work to Curvature-Driven Diffusions (CCD) model which worked better for curved structures.

Alexei et al. [28] proposed a non-parametric texture-based method aiming at preserving as many local structures as possible and grows a new image from an initial seed one pixel at a time based on Markov Random Field model. Another texture-based approach is resynthesis of complex textures [29], [30] which builds up missing region or output by successively adding pixels that closely match target area from input image. Hitoshi et al. [13] combined texture synthesis and image inpainting resulting in a system which overcame the limitations of both approaches. Elad et al. [31] described inpainting model which fills holes in overlapping textures and cartoon image layers based on sparse representation based image decomposition and morphological component analysis.

Exemplar-based methods are designed based on assigning a priority of missing pixels. Once inpainting priorities have been determined, it finds geo-structurally best matching patch from surrounding known region to fill holes. Though computationally expensive, many examples-based methods are proven to be successful in filling larger holes [39], [17], [19], [40], [41].

Often using PDE based methods as basic approaches, hybrid algorithms combine other inpainting methods such as texture synthesis to leverage the advantages of different approaches resulting in more robust recovery [35], [36], [37].

Manuel et al. [32] proposed a simple and fast method based on anisotropic diffusion model extended with the notion of user-provided diffusion barriers. Using information from analyzing stationary first-order transport equations, Folkmar et al. [33] developed a fast noniterative inpainting algorithm. Komal et al. [34] proposed a fast image inpainting algorithm which considers missing pixels as a level set and fills lost pixels assessing image smoothness estimator as well as gradient information.

Past approaches to the inpainting problem have implicitly assumed off-line applications in which their relatively large running times would not be problematic in practice. In other words, their focus was almost exclusively on the
quality of the end result rather than the satisfaction of severe real-time performance constraints. In the next section, we discuss real-time applications in which fast interpolation is required. These include streaming video, in which packet losses produce undefined blocks within frames, and interactive visualization systems in which data may be missing for some patches of the dynamically-changing viewing region. We then introduce our approach and examine its strengths and limitations in realistic broadcast and interactive visualization contexts. We conclude with a discussion of our results and prospects for possible improvements to the algorithm within real-time constraints.

2. Real-Time Video Interpolation

![Figure 1: Example of a corrupted MPEG frame with multiple bit errors leading to corrupted tiles ([6], [1]). In most streaming video contexts it is possible to apply in-painting methods using successive-frame information to fill the missing patches so that they will likely go unnoticed by a casual viewer. However, some high data-rate applications are not amenable to such approaches and thus demand a simpler/faster alternative.](image)

Conventional approaches to image interpolation/inpainting have focused on the preservation of local image properties such that a viewer will not be able to discern any artifacts or discontinuities revealing the use of an algorithm to fill a missing portion of the image. For classical oil-painting restoration the missing portion may be the result of physical damage, and the the in-painting method typically involves an actual painter who attempts to replicate the missing fragment based on knowledge of what existed prior to the damage. In some cases, however, there is no extant information about the details of the missing fragment and the restorer can only try to replace it with something that is consistent both with the surrounding region and with the style of the original artist.

In the case of digitally scanned photographs with missing fragments due to physical degradation, the use of a skilled artist for restoration is often impractical and automated tools are needed. A variety of methods have been developed that span a wide range of mathematics and algorithmic sophistication. The simplest approaches involve determining the color for each missing pixel as a weighted-average of colors based on distances to nearest known pixels. The result of such an approach tends to be a smoothly-varying patch of color with little or no fine-detail consistent with the surrounding region. Thus unless it is very small, a patch interpolated in this way will be readily apparent to even a casual viewer of the image.

More sophisticated methods attempt to maintain the continuity of edge features that appear to extend into the missing region, but the intersection of these extensions can produce artificial features that are highly distinctive and readily distracting to a viewer. Improved methods maintain some continuity of boundary features but enforce constraints on how these features are blended as they intersect within the patch. In other words, an attempt is made to achieve a balance between blurring and detail so as to minimize the extent to which the result “stands out” to a human viewer.

The state-of-the-art for automated filling of image patches is achieved by replicating texture patches surrounding the missing region so that there is strong consistency between the result and the surrounding region. This approach is least discernible to human viewers because it essentially manufactures artificial detail that has no basis in terms of what actually may have existed within the region. In other words, the goal is purely to generate a plausible result that will be accepted as genuine by a viewer. It must be emphasized that this is very different from a goal of inferring as accurately as possible an approximation to the actual details of what may have existed in the missing region.

Real-time interpolation of missing blocks in frames of a streaming video sequence has a unique set of constraints. On the one hand, it only needs to fill the missing patch with fidelity sufficient to not draw attention to itself during the fraction of a second in which it is visible. On the other hand, the interpolated result must be computed within the even smaller fraction of a second that exists between successive frames. It would seem, therefore, that either of the two described approaches may be viable. Unfortunately, the state-of-the-art texture filling methods are too computationally intensive, and the deficiencies of simpler methods become even more pronounced when viewed between successive frames.

One reason why video interpolation is so challenging is because the in-painted region needs to be consistent with both the corresponding regions of the preceding and following frames, and the latter is not generally available in the real-time streaming context. An immediately obvious approach would be to simply replicate the same region

---

1. Even methods that are described as rapid, e.g., [2], do not attempt to achieve real-time performance but are instead focused solely on reducing the high computational cost of prior methods.
Figure 2: Interpolation of missing data in dscale method. (a) Matrix, $M$ represents a typical corrupted block with missing data (each 0 is a missing pixel). (b) Matrix, $S$ shows the scaling data for Matrix, $M$ (numbers are presented up to two decimal for space limitation). Multiplication along each row or column of nonzero elements is equal to 1 (unity) in $S$. Each 0 element is replaced by 1 in (c) Matrix, $S'$. Finally (d) Matrix, $M'$ stages interpolated $A$ matrix or intended recovered data (decimal points are discarded for visualization).

from the preceding frame. If the camera/content is relatively static then such an approach can often produce satisfactory results. When there is fast motion, however, the result will tend to stand out strongly to viewers because of its stark inconsistency with the direction of motion. Motion tracking can potentially be applied to mitigate this issue but is challenging to combine with interpolation within the real-time constraints of high frame-rate video.

Another significant reason why video inpainting is challenging is because the regions to be filled tend to be rectangular. This is because missing data in a video stream is usually a consequence of missing or corrupted data packets corresponding to rectangular tiles defined by the compressed video format. Figure 1 (a) shows this effect in a frame of MPEG video with multiple bit errors [6]. Corrupted tiles can be easily recognized by failed checksums and can either be rendered in corrupted form (as shown in or can be replaced with an interpolated tile Figure 1 (b)). However, unless the interpolation is sufficiently smooth across the horizontal and vertical boundaries of the missing tiles, the rectilinear borders will stand out as distinctive features and will tend to be strongly distracting.

3. Linear-Time Matrix Interpolation

A rigorous definition of what constitutes a “real-time” algorithm is clearly dependent on a variety of application-specific assumptions [3]. For example, faster hardware or parallelization may enable a given algorithm to achieve a specific real-time performance threshold. For present purposes we will avoid such issues by requiring our approach to scale linearly with the number of pixels to be interpolated.

In other words, the coefficient on the linear running time may or may not satisfy demands of a given application, but at least it is unambiguously clear how much improvement is needed from a combination of code optimization and faster hardware.

To this end we have developed an algorithm based on a linear-time algorithm for scaling the rows and columns of a given matrix (not necessarily square) such that the magnitude of the product of the nonzero elements in each row or product is unity [5]. This algorithm has been exploited for efficiently identifying scale-equivalent matrices and for computing a special type of generalized matrix inverse that is consistent with respect to arbitrary diagonal transformations [7], which has found significant applications in robotics and process control applications [10], [9], [8].

To give a feel for the core algorithm, which we will refer to as dscale, consider its use with a nonnegative $m \times n$ matrix $M$ representing pixel intensities of an image. Evaluating $dscale(M)$ gives a decomposition

$$M = D \cdot S \cdot E$$

where $D$ and $E$ are diagonal matrices and the product of the nonzero elements in each row and column of $S$ is unity. This scaling (i.e., $S$) is unique and takes $O(mn)$ time to compute [50]. What is notable about this decomposition is that zero elements of $M$ are unchanged. If missing pixels in $M$ are taken to be zeros, then they will correspond to zero elements in $S$. The critical observation to be made is that if those elements of $S$ are replaced with unity to form $S'$, the product of the elements in each row and column of $S'$ will still be unity. In other words, $S'$ represents a valid dscale
decomposition of a matrix $M'$ with no missing elements:

$$M' = D^{-1} \cdot S' \cdot E^{-1}.$$  

(2)

Moreover, the missing elements of $M'$ are replaced with a value that is scale-consistent with respect to the other values in its row and column. This is an important property because, for example, if the original image represents multispectral information for which the rows and columns are defined in different units, the interpolated result will be consistent with those units, i.e., if the units are changed the resulting interpolation will be identical up to that change of relative scale. Figure 2 presents an example of matrix decomposition in $dscale$ approach.

To apply this approach for image interpolation we need only to identify each region to fill and identify a suitable window so that the missing pixels can be estimated from the frame of existing pixels. The principal constraints in determining the dimensions of the frame are that it include a representative sample of local pixels and that it not extend into a strongly dissimilar region of the image. For our experiments in the following section we interpolate based on a window that is 15% larger than the patch to be in-filled. In other words, no content-based method is applied to optimize the window size.

4. Experimental Results

In this section we examine the approach described in the previous section with examples involving typical streaming video sequences and sequences generated from an interactive visualization system. We begin by noting that standard metrics for comparing the fidelity of processed image results to known ground truth do not generally apply to the problem of interest here because there is no way to divine missing content, so the only meaningful metric is visual consistency. This is demonstrated by the example of Figure 4, where a red car in the missing tile cannot possibly be interpolated from information available in the pixels surrounding the patch.

Examples images used in this section are collected from VIRAT [43] (Figure 6, 9), VIVID [44] (Figure 4, 7) and Maize (Figure 5, 8) dataset. VIRAT is large-scale benchmark dataset which has been widely used in surveillance and computer vision application such as mosaicking [45], [46], [47], [48].

Figure 4: Ground truth (left) contains a red car that cannot be seen in the center image due to the missing tile. No interpolation method can divine the presence of the car from the information available, so comparing the region with the car to that of the $dscale$-interpolated result (right) is not a meaningful measure of effectiveness.

The example of Figure 5 is similar in that the fine details of the rows of crops obscured by the patch cannot possibly be inferred from the surrounding region. In this case the $dscale$ method effectively captures texture features along horizontal and vertical directions. However, the same does not hold for textural features are strongly oriented in diagonal direction. The example of Figure 6 is a glaring example.

In the examples of Figures 7-9 we compare $dscale$ to the partial-differential equation (PDE) method of [49], which
is typical of a family of related methods that attempt to extend features surrounding the missing region into the center of the region. Such methods are iterative and far too computationally expensive for real-time/online applications but are much faster than so-called texture-aware methods [28], [29], [30]. The first comparison example, Figure 7, shows a patch obscuring the boundary between two types of paved surfaces. The dscale result is clearly more consistent with the overall content of the image than the PDE result.

The example of Figure 8 is a frame of video showing rows of crops. Like the previous example, dscale introduces artifacts that are visually more consistent with the surrounding texture features than the blurred result from the PDE method.

The next example, Figure 8, is a frame of video showing a rural highway. The dscale result does not accurately maintain continuity of the road, but its artifacts are certainly less distracting than the blurred result from the PDE method.

5. Discussion

In this paper we have examined a low-complexity linear-time algorithm for the real-time interpolation of lost tiles in frames from streaming video and interactive visualization applications. Our results show that the algorithm is very
effective in many contexts but is susceptible to producing noticeable orientation-related artifacts. Future work will examine means for mitigating the isotropic sensitivity of the algorithm by rotating the interpolation window to align with strong orientation features that may exist. Other potential improvements may derive from a simple method to determine the local direction of motion that can be exploited to further mask distracting artifacts. In all cases, however, the real-time efficiency constraint will be the dominant obstacle.

References


