

UNIVERSITY OF VIRGINIA

COMPUTER VISION

CS 4102

---

# Diminished Reality

---

*Authors:*

Ben HAINES

Travis DEAN

April 26, 2015

# 1 Introduction

This report describes our attempted implementation of a diminished reality system incorporating object tracking, removal, and inpainting. The project was originally inspired by a paper written by Herling and Broll. Many challenges, both anticipated and unforeseen, were encountered during the course of the project and as a result our goals and approach to these goals changed many times.

# 2 Past Work

As mentioned in the introduction, the project was inspired by the paper *Advanced Self-contained Object Removal for Realizing Real-time Diminished Reality in Unconstrained Environments* published by Herling and Broll in 2010. We were initially drawn to the paper by the impressive demonstration videos released by the authors and later the company fayteq founded based on this technology. This was a fairly recent paper and was based on a significant amount of previous work.

The paper was divided into three parts regarding object tracking, inpainting, and optimizations respectively. The authors treated the first of these sections as an already solved problem. They mentioned use of active contour algorithms for tracking but didn't discuss the topic further or cite past work in the area. The problem of tracking objects was treated as a necessary prerequisite but largely outside the scope of the paper. The second section of the paper, inpainting, relied heavily on previous work. Most notable of the papers referenced in this section was the 2009 SIGGRAPH publication *PatchMatch: a randomized correspondence algorithm for structural image editing* by Barnes, et al. Professor Barnes discussed this paper and his work in the area in his guest lecture. The authors based their algorithm on PatchMatch's randomized nearest field search and also incorporated elements of the bidirectional similarity function described by Simakov in 2009.

The primary contribution of this paper was a series of optimizations that permitted the system to work in real time. In this section the primary contributing past work was a framework created by the authors that allowed for fast multicore access to video streams.

### 3 Design and Implementation

Our choice of topic for this project was a last minute one and as a result a number of design decisions were made with an insufficient amount of research to justify them. The choice of this paper at all proved to be a suboptimal decision. While at first reading the high level concepts of the paper are simple to understand, when it came time to implement many of the components the descriptions in the paper were ambiguous. Additionally, because the primary contribution of the paper was an optimization of existing techniques, it assumed a level of familiarity with past work that we did not possess. In retrospect it would have been better to limit ourselves from the beginning to implementing PatchMatch, a more fundamental prerequisite component of Herling and Broll's system that has the benefit of having complete accompanying source code.

We chose to code the project in C++ and make use of the openCV computer vision library. This decision was made with the idea that eventually we would want our code to be as fast as possible in order to run in real time. When that goal was eventually abandoned this choice of language over something more familiar and user-friendly such as Python proved to be a hindrance rather than a benefit.

#### 3.1 Object Tracking

There are two components to the system, object tracking and inpainting. For the first step we initially attempted to use an active contour algorithm to track objects between frames. This was the technique, chosen for its speed, used in the paper. Active contours take an input contour selected by the user and apply a gradient descent approach to minimize an energy function that contrasts properties of the image such as its gradient, with the smoothness of the contour.

We attempted to implement this component of the system by using the openCV function `cvSnakeImage`. This particular function was the only relevant function in the openCV library but only existed in legacy code. Additionally, after experimenting with different test videos and parameters to the function we found that the results were not adequate on most inputs. For the very simplest videos involving two dimensional objects moving across starkly contrasting backgrounds the results were both accurate and fast but in more complex scenes the contours had a tendency to shrink into nothing and were

too easily waylaid by distracting background elements.

After determining that a new approach was needed we decided to use the CAMshift algorithm. CAMshift is an algorithm based on the Mean Shift object tracking algorithm that is typically used for face tracking in embedded systems. It tracks objects based on distribution of hue which results in an algorithm that is very fast but is easily thwarted when objects are multi-hued or have a similar hue to their environments.

The first step of the CAMshift algorithm is to create hue histograms for the image as a whole and the particular region that contains the object to be tracked. The mean shift search window begins centered on this region of interest. At each frame a backprojection is created for a variable size window that grows and shrinks with respect to the zeroth moment. The intensity of each pixel in the window then represents the likelihood that it is a member of the object being tracked. Mean shift is iterated and repeatedly centers the tracking window over the centroid of the backprojection until convergence or a maximum number of iterations have been completed.

The algorithm outputs a rotated rectangle which for simplicity in the second step is converted into a standard bounding rectangle.

### 3.2 Infilling

The paper describes a somewhat complex process by which the portion of the image masked by the tracking stage of the algorithm is filled in. The general approach involves randomly searching for patch correspondances between patches of the image inside of the masked area and patches outside of the mask. After correspondances have been established an iterative function described by Simakov is used to update the value of each pixel inside of the masked region until convergence. In order to do this first the image is scaled down multiple times, then each pixel under the mask is filled in with the average value of its neighboring unmasked pixels. The authors argue that at such low resolution this serves as an acceptable initial approximation. Then correspondances are established as described above, the updating function is run, and results are passed up to the next higher resolution level in the pyramid.

At this point in our implementation we had realized that obtaining real time results was not a feasible goal. In light of this, and in order to maintain simplicity in the code, some of the optimizing components of this process were omitted. In particular, no image pyramid is created. After the mask

is created we immediately fill in the masked pixels with the average values of their neighbors. Then Barnes's PatchMatch is run to establish correspondances. Each pixel in the region of interest is replaced with the average value of the corresponding pixel in every patch that contains it.

This was not the first approach to inpainting that we attempted. At first, before discovering the iterative pixel-by-pixel update function we attempted to take the information obtained from PatchMatch and use it directly. We set the masked region to contain all 0s and then found correspondances. We gradually filled in the area by pasting in patches that overlapped the boundary by a small amount. We also tried initializing the masked area to contain average values of the neighbors, as we do currently, and then finding correspondances and simply filling in the area by pasting in non-overlapping patches. Both of these approaches suffered from a common problem that seems obvious in retrospect. Although they should all be somewhat similar, patches found in the source image can come from very different locations and pasting them directly into the region of interest creates a very blocky discrete fill. The result is jarringly different than the smooth textures one would hope to see on most surfaces.

## 4 Results

HA THERE ARE NONE BECAUSE IT DOESN'T WORK LOL JOKE'S ON YOU FOR READING THIS THING.