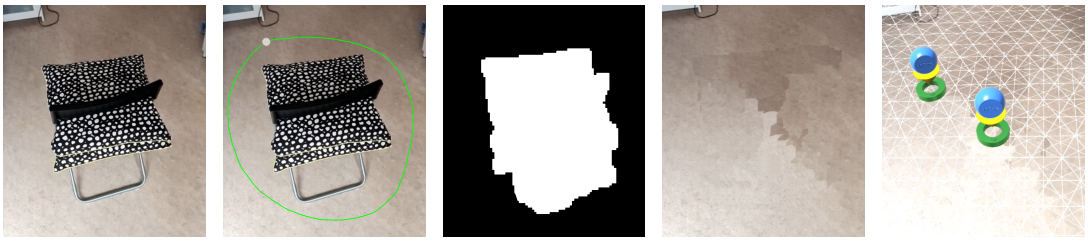
Towards Real-Time Object Removal and Inpainting Through A Diminished Reality Application For Smartphones

Henry Maximilian Cording
Supervisors: Baran Usta, Dr. Michael Weinmann, Dr. Elmar Eisemann
EEMCS, Delft University of Technology, The Netherlands

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Towards Real-Time Object Removal and Inpainting Through A Diminished Reality Application For Smartphones

Henry Maximilian Cording
Supervisors: Baran Usta, Dr. Michael Weinmann, Dr. Elmar Eisemann



(a) (b) (c) (d) (e)

Figure 1: Demonstration of our inpainting pipeline running on an Android smartphone. (a) The input frame; (b) the user draws a rough selection around the object to remove; (c) the automatically generated removal mask; (d); the inpainted result; and (e) plane estimation and augmentation of the inpainted frame with virtual objects using Google's ARCore framework.

ABSTRACT

Diminished reality (DR) is an extension of augmented reality (AR) in which real objects are concealed, removed, or replaced. State of the art DR implementations are written and evaluated on desktop platforms, and those which are aimed at smartphones use auxiliary data or hardware for image completion. This paper introduces a DR app for Android smartphones which can select, track, and remove objects in an unrestricted environment from live video at interactive speed. The application relies exclusively on RGB frames from the phone's camera as input to the DR pipeline. Multiple algorithms for object selection and tracking, as well as a patch-based inpainting approach were adapted. Selection and tracking of objects was very reliable at interactive speed. Inpainting led to convincing results for planar surfaces and unstructured textures, often delivering acceptable results even in more complicated scenes. Future optimizations are required for real-time performance of DR algorithms even on a flagship smartphone, as the hardware was a constraining factor. The performance of inpainting in particular needs to be improved to achieve acceptable frame rates.

1 INTRODUCTION

The advancements in smartphone technology over the last decade have brought a tremendous amount of potential for augmented reality (AR) applications to be used in everyday life [1–3]. Today's AR applications are vast, but usually limited to adding virtual objects to a scene captured by the phone's camera [4], popularized by games such as Pokémon GO, or applications for e-commerce, such as IKEA's "Place" app.

Classic AR is limited in the sense that any existing objects in the scene cannot be altered, but instead only augmented by adding virtual information. Diminished Reality (DR) aims to solve this issue by removing, concealing, or modifying appearances of real objects [5] in the scene. To achieve this, an image completion algorithm is applied, which fills the space previously occluded by the removed object with background information. DR creates new applications, such as refurnishing a space (e.g., [6]) or seeing through a vehicle that is obstructing the driver's field of view [7].

Despite its potential to extend the feature set of mobile AR, the majority of DR implementations presented in the literature are written and tested on desktop CPUs, and often not real-time capable (i.e., frame rates above 30 are not achieved). Moreover, they usually rely on additional cameras or auxiliary data, such as depth maps and 3D scene models. As such, they are not suitable for most commodity smartphones, even though the mobile platform is arguably the most important for AR and DR, together with wearable devices such as Microsoft's HoloLens. To our knowledge, no DR applications for object removal in live video exist on the Apple and Android app stores today.

The aim of this research is to address these shortcomings by integrating DR into a smartphone app while aiming for real-time performance. Constraints arising from auxiliary data or hardware are eliminated by using only a single RGB camera of the phone and no prior scene scans. To achieve this, a key focus of this work is to find a suitable combination of proposed DR techniques for object selection, tracking, and inpainting, and optimizing them where possible. Limitations of these methods and of smartphone technology in general, and the resulting implications for the feasibility of real-time DR apps, are an integral part of the analysis.

To summarize, we make the following contributions:

(1) An Android app which supports DR through selecting and tracking objects, and inpainting the region of interest in live video at interactive speed, as well as "classic" AR to add virtual objects;
(2) An open source codebase for mobile DR using Google's ARCore framework [8] and the OpenCV library [9], allowing future researchers to easily expand and build upon the presented work. The source code will be made available upon acceptance of the paper; and

(3) A comprehensive evaluation of how various algorithms for DR perform on a flagship smartphone, indicating if the mobile platform is yet suitable for such applications.

The paper is presented with the following structure. In Section 2, the related work on DR is discussed. Section 3 describes the methodology of research, including the DR pipeline and a detailed description of the algorithms which were studied and implemented. The developed prototype is evaluated in Section 4, and various results regarding performance and visual quality are provided. Following these findings, Section 5 discusses the results in terms of strengths and limitations, and compares the work to results from the literature. Section 6 concludes with the findings of this research and suggests directions for future work in the field of mobile DR. Lastly, section 7 features a discussion on responsible research.

2 RELATED WORK

An early concept of DR was already described in 2002 [10], e.g. by replacing an advertisement on a billboard. DR in the sense of object removal and image completion dates back further, e.g., removing objects from an image by decomposing it into layers [11]. Later it was shown that using multiple calibrated views of a scene, industrial pipes could be seamlessly removed from a reference image [12]. This idea was also extended to remove objects from video (cf. [13–15]). Such approaches cannot be used on smartphones, since only a single viewpoint is captured. A single-view approach for object removal in videos was proposed by Wexler et al. [16], but it relied on manually created removal masks for each frame that are not available in real-time applications.

More recent approaches to DR often construct a 3D model of the scene by scanning it with help of a depth sensor, and use this data to provide a plausible background for removed objects (cf. [6, 17, 18]). Scan-based approaches would be possible to implement given that some modern smartphones carry LIDAR [19] sensors, and even depth estimation without auxiliary hardware has improved significantly. However, a prior scene scan would notably reduce the usability of the application for frequent interactions, whereas online depth estimation may reduce the computational headroom needed for inpainting.

Several other contributions have been made to the area of DR which focus on improving the accuracy of image completion (e.g, [20]). Such works have not been the focus for this research because the accuracy of inpainting is severely constrained by the real-time requirements, and even the simplest inpainting algorithms are difficult to implement for real-time use with higher resolution video.

Although all the works discussed so far mark important advancements in DR research, they impose constraints on either the technology or data that is required for the implementation. Herling and Broll [4, 21] recognized these limitations and proposed a real-time DR application capable of removing objects and inpainting the space, using only a single RGB camera and information from the previous and current frame. Their work is the most relevant for this research.

To select and track objects, the authors used the active contour algorithm [22] in their earlier work (2010). It was later noted that its range of application is rather limited due to the algorithm's simplicity [21]. To overcome this, the authors later proposed their own, much more rigid implementation for object selection, based on the clustering of characteristic image points (so-called "fingerprints") located near the object to select [21].

For image completion, they employed an inpainting algorithm based on the PatchMatch procedure [23, 24]. Applying this method iteratively using an image pyramid [25] accelerated the convergence and allowed for real-time performance. Gray scale images were used for further optimization, and color was recovered in the last pyramid layer by applying the calculated patch correspondences to the original video frame.

The work of Herling and Broll is undoubtedly remarkable, but it has some limitations w.r.t. mobile DR, as development and evaluation were carried out on a desktop operating system, and a much lower video resolution was used. Also, the application did not support adding virtual objects. These points are addressed in this research, together with the contributions outlined in Section 1.

3 METHOD

The approach for this research was largely based on the main components of DR: object selection, tracking, and inpainting. The DR pipeline is described in section 3.1. Since the application also had to support the AR use case of adding virtual objects, a simple AR application was developed first, and the DR functionality was then built on top of it. The implemented algorithms for selection, tracking, and inpainting are discussed in sections 3.2, 3.3, and 3.4, respectively.

3.1 Overview of the DR pipeline for this research

Any diminished reality application consists of at least three major components [5, 20, 26]:

(1) Selecting the ROI to modify, and possibly an object within it,
(2) Tracking this selection as the camera is moved, and
(3) Replacing the pixels in the selected region with background information (this step is also referred to as image completion).

Figure 2 illustrates the DR pipeline, which for this research was purely image based, as no 3D reconstruction or auxiliary data could be used.

3.2 Object selection

Research into different approaches to object selection was conducted, and two techniques were examined further. Extracting contours in the region of interest (ROI), and using GrabCut [27], an interactive image segmentation algorithm based on graph cuts that allows the user to manually adjust the found selection.

Object selection based on contour extraction. A simple and fast approach to select objects is finding their contours. After an initial outline is given by the user, the bounding box of the selection is used as the ROI. This ROI is converted to grayscale and downsampled for performance. Then, a bilateral filter [28] is applied, which removes noise while preserving edges. The Canny edge detector [29] then produces a binary image from this input containing the most prominent edges. Edges are dilated to ensure that small gaps in an otherwise closed contour are filled, and to cover small shadows cast by the object.

The result is upscaled back to full resolution. The `findContours` function in the OpenCV library [9], which is based on [30], extracts

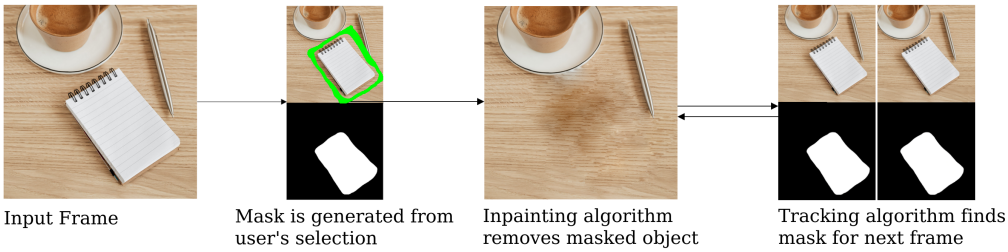


Figure 2: Diminished reality pipeline. After the first frame is inpainted based on the user's selection, the tracking and inpainting algorithms work automatically to inpaint subsequent frames.

the image contours. For this first frame, the largest contour is chosen as result. Figure 3 shows the process of selecting an object by extracting its contours.

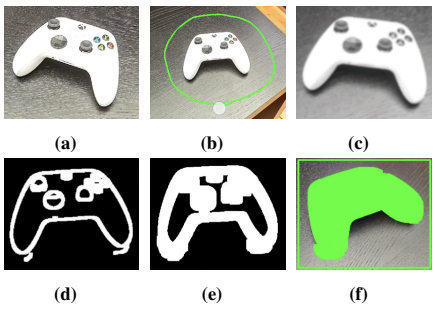


Figure 3: Selecting an object by extracting contours. (a) The object to select; (b) user draws a rough selection; (c) conversion to grayscale, blurring, and downsampling; (d) Canny edge detection; (e) dilation; and (f) contours are detected and the outermost contour is filled.

Object selection based on GrabCut. The interactive GrabCut image segmentation algorithm [27] is based on graph cuts. Pixels are represented as nodes in a graph. The goal is to find a minimum cut of the graph, segmenting it into foreground and background pixels. Anything outside the given ROI is marked as certain background. Then, the algorithm iteratively finds a suitable graph cut. In this process, the user can guide the algorithm with hard constraints by drawing strokes to adjust the representation of foreground and background. The output of the algorithm is a binary mask that segments the image.

Challenges arise when implementing GrabCut for video. After initialization using a bounding rectangle (ROI), the resulting mask is refined on each iteration. However, this mask will be displaced if the camera moves. Figure 4 illustrates this issue.

There are two solutions to this problem. The mask could be translated w.r.t. the displacement of the bounding box that is returned by the tracking algorithm (discussed in detail in Section 3.3). While this operation is cheap, it will provide erroneous results for any asymmetrical object. In such cases, the user needs to manually redraw the selection after only minor movement. In our implementation, we instead opted for the following approach:

(1) Receive the new bounding rectangle from the tracker.

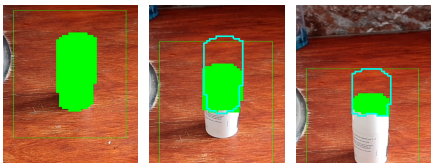


Figure 4: Issue of mask displacement when using GrabCut for video. Blue represents the (approximate) mask of the previous frame, green represents the refined output mask. (a) Initial selection; (b) the camera is moved up, parts of the object are no longer selected even though the bounding box fully encloses it; and (c) the initial mask hardly covers the object anymore, GrabCut has difficulty selecting the object.

(2) Translate and scale the user's adjustment strokes (hard constraints) from previous frames based on the displacement of the bounding rectangle.
(3) Calculate a segmentation mask with GrabCut by using the received bounding rectangle.
(4) Apply the scaled and rotated adjustment strokes to the resulting mask.
(5) Execute the algorithm again, this time using the adjusted mask.

The complete process of selecting an object with GrabCut is shown in Figure 5.

3.3 Object tracking

Two approaches were investigated. Firstly, the KCF (kernelized correlation filter) [31] and CSRDCF (discriminative correlation filter with channel and spatial reliability) [32] tracker implementations in OpenCV were examined. CSRDCF is also referred to as CSRT. Secondly, the observation was made that a contour found in the previous frame can be recovered in the current frame based on its size and position, as well as visual appearance. A simple algorithm was developed from these ideas.

Object tracking based on KCF and CSRT. The KCF and CSRT trackers allow for high performance tracking of a rectangular ROI. For the later inpainting procedure, the entire updated ROI returned by the tracker can be marked for removal, or a technique from Section 3.2 is applied to find a more accurate mask surrounding the object of interest inside the ROI.

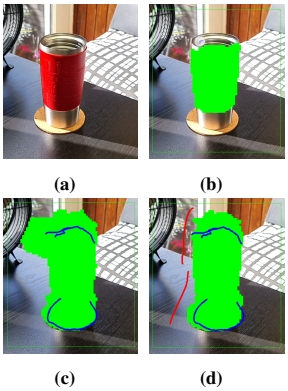


Figure 5: Object selection using GrabCut, the mask is no longer displaced. (a) The object to select; (b) the initial mask found by GrabCut (1 iteration) is not sufficiently accurate; (c) the user expands the mask manually, some areas are now erroneously included in the mask; and (d) the user removes undesired parts of the mask.

The tracker is initialized by providing the current frame and a ROI. On subsequent iterations, only the frame is provided, and the tracker returns the estimated ROI in the current frame. This is achieved using online learning in both KCF and CSRT.

Object tracking based on extracting contours. We now describe the design of a much simpler tracking algorithm with better performance and lower reliability, which is generally acceptable for real-time use. The goal of this tracker is to identify the contour belonging to the object of interest, out of all contours that were found by the contour extraction routine in the current frame. This algorithm is based on the assumption that the user's initial selection will be approximately centered around the object of interest, and that the camera movement between frames is not extreme.

In frame f the largest contour in the initial ROI is found, denoted by c_f, with centroid o_{c_f}. In frame $f+1$, the position of the contour may have changed, and new (possibly larger) contours may have moved into view. An algorithm that always selects the largest contour would now fail. Instead, one can select the contour c_{f+1} which minimizes the sum of centroid distance and visual dissimilarity w.r.t. c_f.

An arbitrary distance measure can be used for the visual dissimilarity. In our implementation, we first construct masks for c_f and each contour c in $f+1$ by painting a white polygon with the shape of the contour onto a black image, giving M_c and M_{c_f}. Then, the bitwise XOR of the masks is calculated, so that the penalty is increased by 1 for each differing pixel. While this distance measure is strict, it is also very fast to compute and follows the same reasoning that the desired contour c_{f+1} should not be significantly different from c_f. In sum, the contour in each frame can be calculated as follows:

$$c_{f+1} = \begin{cases} \max\limits_{c \in C} A(c), & \text{if } c_f = \text{null}. \\ \min\limits_{c \in C} d(o_c, o_{c_f}) + M_c \oplus M_{c_f}, & \text{otherwise}. \end{cases} \quad (1)$$

Here C is the set of all detected contours in frame $f+1$ that lie within the ROI that envelops c_f. The function d calculates the euclidean distance. After the update is completed, the ROI is set to the bounding box of c_{f+1} and the procedure is repeated. As a final consideration, our implementation retains the previous contour if no good result was found in the current frame (e.g., due to motion blur). A new match is typically found within a few frames.

3.4 Inpainting

Inpainting was chosen amongst the methods for image completion detailed in Section 2, because it is the only technique that does not require additional data or hardware to operate. Several papers were reviewed to derive a suitable implementation. The work of Herling and Broll [4, 21] was particularly insightful. The PatchMatch (PM) algorithm [23] provided the foundation for the implementation, as it appeared to be most prominent in research on real-time DR, and provided sufficient accuracy given the performance constraints of this project.

PM enables inpainting by finding so-called "patches" - square image regions - and copying them into the ROI. The intuition is that lost information in the ROI can often be reproduced by reusing information from other parts of the image, assuming that no unique information lies within. The use of square patches results in sharp features and a much simpler representation of mappings between patches. However, it also introduces visible segmentation between patch groups, and variation is more limited overall.

The input to PM is the current frame f and a binary mask b which indicates the area to inpaint (as given by the selection and tracking algorithm). The output is the inpainted frame. First, f is converted to grayscale, and an image pyramid is constructed for f and b. On the highest level (the lowest resolution), an initial guess is made using a fast marching approach to inpainting [33] provided in OpenCV. This estimate helps PM to converge quicker, and it is cheap to compute at this resolution.

For each level in the image pyramid, the algorithm proceeds as follows (we refer to masked pixels as pixels that need to be inpainted):

(1) Initialization: The nearest neighbor field (NNF) is constructed. It stores for each masked pixel p the position and cost of the lowest cost patch outside the masked region. This cost is obtained by applying a distance measure to the target patch and the equally sized patch which has p as its top-left pixel. At first, mappings are assigned randomly. Unmasked pixels are mapped to themselves with zero cost, indicating that no changes should be made to these areas. If an NNF exists from the previous pyramid level, the initialization is completed by scaling the previous NNF up by a factor of two, simply copying the value of each pixel four times.
(2) Propagation: Each masked pixel p tries to improve its mapping by checking if its immediate neighbors point to cheaper patches. For example, if the left neighbor p' of p located at $(p.x-1, p.y)$ maps to pixel m, then p checks the cost of mapping to pixel m' at $(m.x+1, m.y)$. The propagation step is performed in scan-line order, forwards on even levels and backwards on odd levels.

(3) Random search: Each masked pixel p tries to improve its mapping by checking for a random unmasked pixel if its patch would reduce the mapping cost of p. For each masked pixel, multiple searches are performed, and the search area is decreased exponentially.
(4) Transfer and voting: The mappings found on level i need to be applied to the image on level $i-1$, which is twice as large. Transferring the patch of each masked pixel p directly gives a poor result, because it is likely that the patches containing p all have slightly different mappings, and thus artifacts are created. Instead, the patch with the cheapest mapping out of all patches containing p is selected. To avoid overlap of patches in the image on level $i-1$, the patch size is set to 2×2.

After 5 iterations (1 iteration per pyramid level), the result is returned as a grayscale image g. To recover the color, the NNF is used together with f to copy all patch correspondences to their correct location (which works since f and g have the same dimensions). Figure 1 showcases the process of inpainting.

Distance Measure. The distance measure gives the cost of mapping a patch of masked pixels to an equally sized patch of unmasked pixels. In our implementation, the spatial distance (SD) and visual similarity (VS) based on the L^2 norm are used. An implementation using only the SD will see that the pixels at the boundary of masked and unmasked pixels are simply copied inwards, whereas using only VS will result in many artifacts, since most pixels will use their own mapping found through random search instead of accepting a patch from their neighbors. Equation 2 describes the cost for each mapping.

$$d(A,B) = \sqrt{(r_A/h - r_B/h)^2 + (c_A/w - c_B/w)^2} \\ * \ln(\ln(L^2(A,B)+1)+1) \tag{2}$$

Here r_A and c_A are the row and column of the top left pixel of patch A, and similarly for B. The image width and height are denoted by w and h, respectively. The row and column values are therefore normalized w.r.t. the image size, meaning that the SD is independent of the image resolution, and always between 0 and $\sqrt{2}$.

Intuitively, image data that is closest to the area to inpaint is very likely to also be contained within it, which is why the VS is multiplied by the SD. Applying the natural logarithm twice to the VS is done to emphasize small improvements to a VS that is already very small, and to limit the visual dissimilarity beyond high orders of magnitude. This encourages pixels to also select patches that are visually somewhat different, but over time may yield very good VS for neighbors of this pixel. Lastly, by adding 1 to the result of each logarithm, the VS is positive for any output of the L^2 norm (which is greater or equal to 0).

Coherence and Frame Propagation. Recomputing patch correspondences in each frame is expensive, but also leads to coherence issues and flickering of the inpainted region. Our implementation copies the image region inpainted in frame f to the masked region in frame $f+1$. This ensures that PM starts with a better guess, and promotes convergence towards a result more similar to the previous one.

3.5 Implementation details

Setup and Hardware. Development took place in the Android Studio IDE using Java 11, on a Samsung Galaxy Note20 Ultra (SM-N986B) with Android API level 31. Only the main camera was used to receive RGB frames as input to the application. Frames were processed at a rate of 30 frames per second, with a resolution of 1280x720 pixels (720p). However, algorithms for object selection and tracking used a down-scaled image of 200x112 pixels, to remove unnecessary detail and improve performance.

Use of ARCore. Implementing AR functionality was achieved using Google's ARCore framework. It provides several relevant features such as plane detection, scene light estimation, and camera pose (position and orientation) estimation [34]. Feature points in the scene are detected using SLAM (simultaneous localization and mapping) together with the information which is provided through the phone's sensors, such as the gyroscope and accelerometer. The `hello_ar` sample project provided by Google [8] facilitates these features, and was used as foundation for the application. Appendix A showcases this application.

4 EVALUATION

Evaluating image algorithms for video when the camera has six degrees of freedom (6DoF), meaning no restrictions are placed on the position and rotation during the interaction, is challenging [5]. Misalignment, shaking of the hands and so on, make it difficult to reproduce a video sequence and obtain ground truths for each frame. A qualitative analysis is provided for the reliability of object selection and tracking, and the quality of inpainting. The performance of all algorithms is evaluated quantitatively, by considering latency under various input sizes. Finally, the quality of inpainting is analyzed over still images using the peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) [35].

4.1 Qualitative Evaluation

The implementation was applied to scenarios with various lighting conditions and backgrounds. The objects to select, track, or remove were not always symmetrical, and of varying texture.

Selection and tracking by extracting contours worked most reliably in settings without strong incidents of light from nearby windows or lamps. The object contour repeatedly merged with the background when illuminated from certain angles. Also, the contrast between the object and background had to be strong for the selection to be found in the initial step (such as in Figure 6a), and instances with partial occlusion by other objects with similar texture were not solved by the algorithm. The latter is illustrated by Figure 6h, in which the left object is to be selected, but the right object is erroneously included as well. Overall, the algorithm was rather limited in its applicability, and not suitable for cases like those shown in Appendix B. However, when it did work it selected and tracked the object very reliably.

Selection with GrabCut performed well in almost all settings. Inaccuracies due to lighting or asymmetric object shape were easily addressed by providing a few adjustment strokes. Even challenging cases as illustrated in Figures 6i and Appendix B were consistently solved by the algorithm. Occasionally, the adjustment strokes by the

user were not correctly incorporated, which may be due to the lower resolution (200x112) that it operated on. Nevertheless, selection with GrabCut effectively selected objects in the majority of settings.

Tracking with CSRT was surprisingly accurate in all test scenarios, even at very low resolutions. However, the performance was far from real-time, therefore leaving almost no computational headroom for inpainting. For this reason, the CSRT tracker was neither considered further nor evaluated quantitatively.

Tracking with KCF performed notably better than the CSRT variant, but sometimes lost the ROI when moving the camera excessively. Asymmetric objects or those with varying textures on each side were not always properly tracked or lost after a large rotation around the object. As long as the tracking was functional, the returned bounding box for each frame was sufficiently accurate. Figures 5 and 12 are examples in which the KCF tracker was used to provide the bounding box to the GrabCut algorithm.

Inpainting was evaluated by considering both two-dimensional settings by applying the algorithm to partially destroyed textures, and three-dimensional settings in live interactions over various scenes with differing lighting and background geometry. The coherence [36], and subsequently the overall similarity of the inpainted and original background were considered.

Figures 1 and 7 show the inpainting process for various scenes. The algorithm performed best in scenarios with planar backgrounds when inpainting unstructured textures. This is partially due to the lack of composition (blending) between patches, which can leave visible outlines around larger patch clusters, such as in Figure 1e. Additionally, the algorithm currently does not reward patches for reproducing existing patterns in a texture, which, in combination with a steep viewing angle, leads to a lack of coherence in structured textures and at plane intersections (such as in Figure 7i). These artifacts become increasingly visible the larger the area to inpaint; for smaller problems such as in Figure 7c the inpainted result appears seamless. In sum, given the hardware and performance constraints for this research, our algorithm provides reasonable results. However, the quality of inpainting will vary depending on the use case.

4.2 Quantitative Evaluation

Quantitative evaluation was conducted using still frames, and measuring metrics such as average latency for object selection and tracking, as well as pixel fill rate and image similarity for inpainting. Figure 8 plots the performance of the various algorithms to execute on ROIs of different size, averaged over 100 runs.

Selection and tracking by extracting contours turned out to be extremely efficient. As is also the case with most of the following algorithms, an exponential relationship between the required computation time and input size could be observed (cf. Figure 8a). Note that despite the additional computation required to track a contour from the previous frame, the latency hardly differed from the initial step of simply selecting the largest contour.

Selection with GrabCut provided suitable performance only for very small inputs of up to 160x120 (cf. Figure 8b). At this resolution, the reliability of GrabCut was notably reduced. While an update with an existing mask is again exponential w.r.t. the input size, the initialization step seemed to scale linearly. It was also reassuring to see that the time for an update was significantly lower than for the initialization. In practice, with an input of 200x112 the GrabCut algorithm allowed for interactive performance.

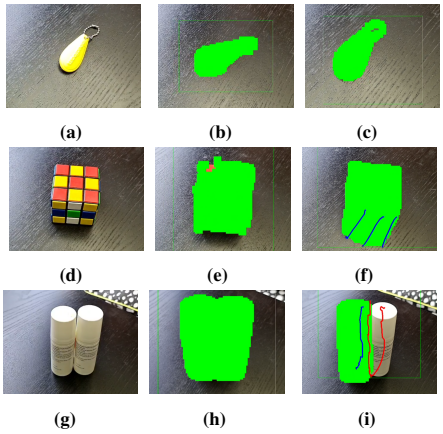


Figure 6: Comparison of object selection algorithms. For each row, left to right: input, selection by extracting contours, selection through GrabCut. Increasingly complex objects or scenes where the object contour is not clearly defined are difficult for the contour extraction algorithm. GrabCut yields good results in all cases, but requires increasing amounts of manual adjustments.

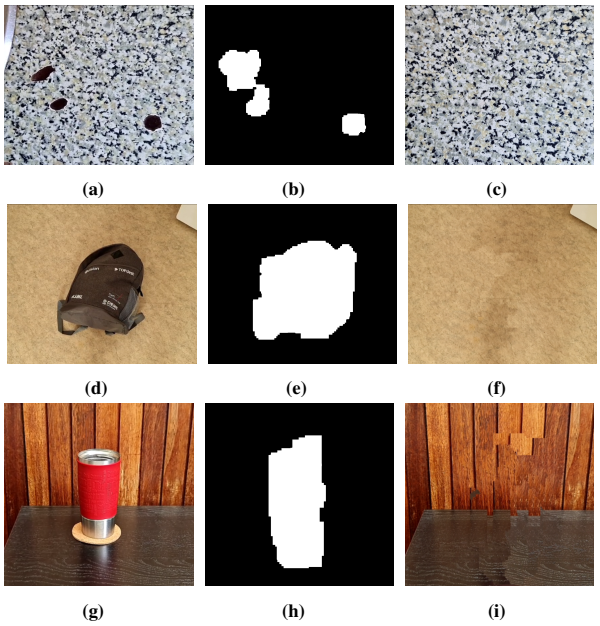


Figure 7: Application of the inpainting algorithm to various scenes. From left to right: the input image, the removal mask (initially generated from the user's selection), the inpainted result. (c) The algorithm performs best with planar backgrounds and small fill regions; (f) larger areas are inpainted convincingly over unstructured textures; and (i) structured textures and plane intersections lead to a loss of coherence in some regions.

Tracking with KCF performed well in the initialization step, which was expected since the first call is used to simply set the target

to track, whereas each update call needs to actually localize the new ROI. Surprisingly, the initialization step started to take significantly longer for resolutions above 1024x768, and the time required for initializing at 720p (which is not plotted) exceeded 10 seconds. Using the input of 200x112, interactive performance was achieved. However, since KCF was used in combination with GrabCut, the performance was impaired further.

Inpainting was evaluated by measuring the pixel fill rate using a 5-level and 2-level image pyramid, as well as the peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) [35] when inpainting destroyed regions of textures. The pixel fill rate initially saw an increase, arguably due to the multi-threading optimization which adds a constant overhead and is less performant over few iterations. For higher resolutions, it converged to ca. 30 pixels/millisecond. It remained unclear why this rate was not more stable at lower resolutions, given that it should be largely independent of the input size, and initialization cost of the image pyramid with at most 5ms per frame was negligible even at high resolutions.

Surprising was also the only marginal increase when using additional pyramid levels, which was meant to accelerate convergence for PM. Both [4] and [16] reported performance gains when using this technique. Since only 1 PM iteration is performed per level due to performance constraints, the algorithm in our implementation may not benefit enough from the faster convergence to make up for the overhead of constructing and iterating through the pyramid levels.

The PSNR and SSIM were calculated for five examples and are given in Table 1. A PSNR above 40 is typically considered acceptable, for the SSIM the score is 1 for identical images, and reduces gradually to 0 the more dissimilar two images are.

The scores confirmed the findings of the qualitative evaluation, namely that the visual similarity is notably reduced for inpainting of structured textures, especially evident in Figure 13e (Example V). Surprisingly, the scores for the texture in Figure 13b (Example II) were comparatively low, even though the inpainted result is convincing. This emphasizes that there can be many possible solutions to inpaint an area, and although they may differ strongly from the ground truth, the results can still be coherent and convincing.

	I	II	III	IV	V
PSNR	48.65	36.27	52.59	42.28	34.64
SSIM	0.9887	0.8434	0.9948	0.9492	0.7747

Table 1: Peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) for 5 inpainted image regions compared to their ground truths. Examples I to V are given in Appendix C.

5 DISCUSSION

In this section the implementation is compared to other relevant works, and limitations are outlined. Note that the proposed DR pipeline is meant to run in real-time, and on very restricted hardware. Therefore, it is expected that results are far less accurate.

5.1 Comparison to other works

The inpainting algorithm was based on PatchMatch, but had to be adapted for efficient use on a smartphone. Although several optimizations were implemented, the obtained results remained similar to

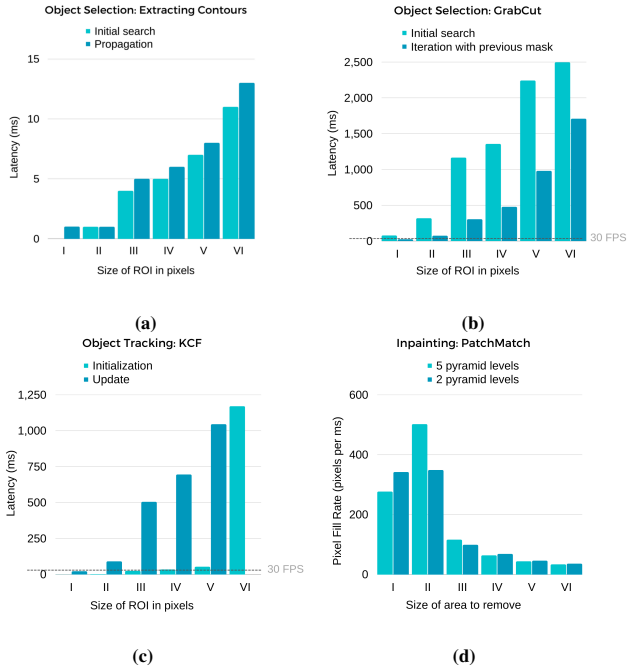


Figure 8: Performance of algorithms for object selection, tracking, and inpainting in terms of latency and pixel fill rate, averaged over 100 runs. In the application, the input frame for object selection and tracking is always downsampled to 200x112, whereas for inpainting the size depends on the ROI given by the tracking algorithm. Size of ROI: I - 160x120, II - 320x240, III - 640x480, IV - 800x600, V - 1024x768, VI - 1280x720.

those of the original PatchMatch algorithm [23]. Figure 9 illustrates this with an example.

We also compared our work to the real-time inpainting approach of Herling and Broll [4], since it is the most similar to ours. Figure 10 shows multiple scenarios from their paper which were run with our inpainting pipeline. The results in the first two examples are similar, however, it appears that in the work of [4], a type of averaging or blending was applied between patches to achieve a smoother result, which was not done in this research. Figures 10b and 10c illustrate this well, since the patch regions are very similar, but the transitions between them are far more visible in our implementation.

The third example was taken from [21], in which the same authors refined their inpainting approach to achieve better coherence. Here it is clear that our patch based approach is limited by the distance function which considers only spatial distance and visual similarity, without regards to continuity of textures.

5.2 Limitations and future work

Performance. The DR techniques implemented in this research provide adequate results, but most are not real-time capable. The GrabCut algorithm allows selection of almost any object in diverse scenery, but it has to be optimized not just to achieve real-time performance, but to also leave enough headroom for the inpainting algorithm. The same reasoning applies to the KCF tracker that GrabCut relies on. Alternative approaches to object selection and tracking, such as super pixels [37] and CNNs such as [38], could be

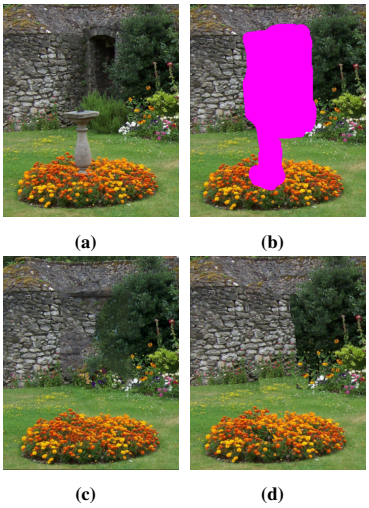


Figure 9: Comparison to an inpainting example in the original paper on PatchMatch [23]. (a) Input image; (b) removal mask; (c) result of [23]; and (d) our result. All images were enlarged to focus on the region to inpaint. As expected, the visual quality of inpainting is very similar, but fewer iterations in our approach lead to some artifacts.

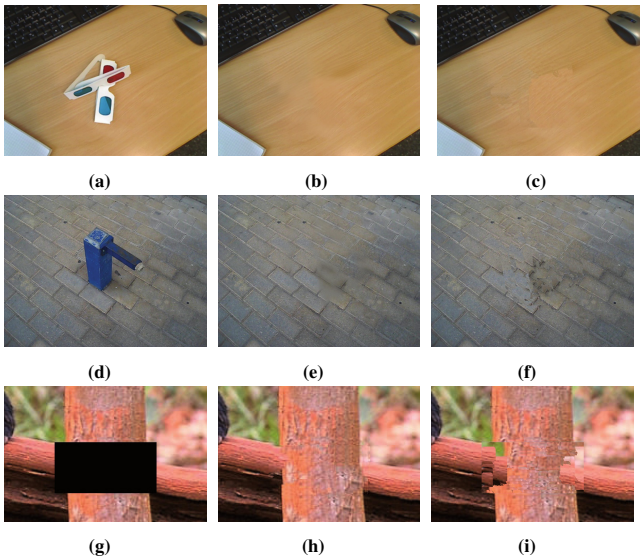


Figure 10: Comparison to examples presented by Herling and Broll [4, 21]. For each row, from left to right: input, their result, result using our optimized inpainting algorithm on a smartphone.

investigated and may provide better performance. Tracking could also be achieved by using SLAM, which is already implemented in ARCore.

Inpainting performance could be improved by reducing the NNF to store only patch correspondences for masked pixels, since the mapping of unmasked pixels to themselves is essentially redundant. A more compact representation could reduce memory usage significantly, and accelerate iterations. Additionally, transferring image data from level i to $i-1$ is currently separate from the expansion of the NNF to twice its size. Combining these operations would reduce the overall transfer cost notably.

It should also be investigated why the performance was not improved by using additional pyramid levels, as outlined in Section 4. This is important, since the convergence speed of PM is the dominant factor for latency and visual quality. If more iterations can be performed in the same or less time, the current visual artifacts may be eliminated.

The smartphone's CPU may also be a limiting factor, not to mention several related considerations such as battery life, lack of active cooling, constrained system memory, and so forth. Performance may however be aided by implementing the DR pipeline on the phone's GPU.

Visual quality of inpainting. The visible division between groups of patches as addressed in Section 4 can be reduced by either averaging the pixel color during the transfer and voting process, or implementing composition as fourth step in the DR pipeline [5]. Color correction could be used to make inpainting more convincing when the scene light or camera exposure changes, which currently leads to coherence issues during propagation of inpainting results between frames.

Lastly, the coherence of inpainting over structured textures could be improved. Generalized patch match [24] considers scaling and rotation of patches, and image melding [39] expands on this approach further to achieve highly convincing inpainting results. Although more computationally expensive, such approaches may be suitable for mobile DR in the future. Lastly, the use of machine learning for image completion may provide a compelling alternative to inpainting, if it can be implemented efficiently.

6 CONCLUSION

In this paper we have presented a diminished reality application for commodity smartphones that allows removing objects at interactive speed from live video while using only data from a single RGB camera. Due to the lack of constraints, the approach is much more widely applicable compared to many DR solutions presented in the literature. Two object selection algorithms were investigated, which present trade-offs between accuracy and performance in different scenarios: extracting contours, and GrabCut [27]. To track an object, the KCF [31] and CSRT [32] trackers were investigated, and a far simpler algorithm for contour tracking was proposed. Lastly, the PatchMatch algorithm [23] was adapted to achieve interactive inpainting with good coherence for planar backgrounds and unstructured textures. The evaluation showed that both object selection and tracking, as well as inpainting, need to be optimized further to achieve real-time DR on a smartphone. The limited hardware of the mobile platform also presented a limitation. Future work in mobile DR may look into how the presented algorithms can be optimized, and which alternative approaches could be feasible to realize a real-time DR pipeline. It would also be of interest to see whether the visual quality of inpainting can still be improved, considering the performance constraints of the mobile platform.

7 RESPONSIBLE RESEARCH

Practicing research responsibly is essential as it ensures that the work upholds scientific integrity, and is free of ethical issues or that the authors maintain awareness and appropriately address such issues as best they can. Further, the research should be reproducible, to verify that results have not been cherry-picked, and that the presented findings are legitimate overall. It also enables other researchers to more easily continue and improve on the work of a previous paper.

Scientific integrity. This research is supported by a vast collection of sources, all of which are appropriately referenced in this paper. Paraphrased sentences, direct quotes, and information taken from other works have been clearly marked as such. All other content has been produced by the author.

Reproducibility. The source code for this research will be made available upon acceptance of the paper, in accordance with the supervisors. By publishing the source code, other researchers can recreate and verify the results presented in section 4. Moreover, the research is entirely reproducible by simply running the codebase on an Android smartphone, and development can be continued in less than a day. To this end, the code has been extensively documented.

Ethical considerations. Diminished reality is not free of ethical issues. In theory, the technology could be used by governments or corporations to censor or replace pieces of content from live video. When DR becomes increasingly convincing and real-time capable, it may be difficult to discern an edited signal from the original.

In this research we have proposed a smartphone prototype to lay the foundation for mobile DR development. In its current form, it does not facilitate any of the aforementioned abuse. As this is a starting point for further work, it will be the responsibility of future developers to ensure that increasingly sophisticated DR implementations are not exploited. As with all emerging technologies, a residual risk will always remain.

REFERENCES

[1] T. Olsson and M. Salo, "Online user survey on current mobile augmented reality applications," Nov. 2011, pp. 75–84. DOI: 10.1109/ISMAR.2011.6092372.

[2] M. Mekni and A. Lemieux, "Augmented reality: Applications, challenges and future trends," 2014.

[3] H. Chen, Y. Dai, H. Meng, Y. Chen, and T. Li, "Understanding the characteristics of mobile augmented reality applications," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018, pp. 128–138. DOI: 10.1109/ISPASS.2018.00026.

[4] J. Herling and W. Broll, "Advanced self-contained object removal for realizing real-time diminished reality in unconstrained environments," in *2010 IEEE International Symposium on Mixed and Augmented Reality*, 2010, pp. 207–212. DOI: 10.1109/ISMAR.2010.5643572.

[5] S. Mori, S. Ikeda, and H. Saito, "A survey of diminished reality: Techniques for visually concealing, eliminating, and seeing through real objects," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, p. 17, 2017. DOI: 10.1186/s41074-017-0028-1.

[6] E. Zhang, M. F. Cohen, and B. Curless, "Emptying, refurnishing, and relighting indoor spaces," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2016)*, vol. 35, no. 6, 2016.

[7] F. Rameau, H. Ha, K. Joo, J. Choi, K. Park, and I. S. Kweon, "A real-time augmented reality system to see-through cars," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 11, pp. 2395–2404, 2016. DOI: 10.1109/TVCG.2016.2593768.

[8] Google, *Arcore sdk for android studio*, 2022. [Online]. Available: https://github.com/google-ar/arcore-android-sdk.

[9] OpenCV, *Open source computer vision library*, 2015. [Online]. Available: https://opencv.org/ (visited on 06/19/2022).

[10] S. Mann and J. Fung, "Eyetap devices for augmented, deliberately diminished, or otherwise altered visual perception of rigid planar patches of real-world scenes," *Presence: Teleoperators & Virtual Environments*, vol. 11, pp. 158–175, 2002.

[11] J. Wang and E. Adelson, "Representing moving images with layers," *IEEE Transactions on Image Processing*, vol. 3, no. 5, pp. 625–638, 1994. DOI: 10.1109/83.334981.

[12] S. Zokai, J. Esteve, Y. Genc, and N. Navab, "Multiview paraperspective projection model for diminished reality," in *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.*, 2003, pp. 217–226. DOI: 10.1109/ISMAR.2003.1240705.

[13] S. Jarusirisawad and H. Saito, "Diminished reality via multiple hand-held cameras," in *2007 First ACM/IEEE International Conference on Distributed Smart Cameras*, 2007, pp. 251–258. DOI: 10.1109/ICDSC.2007.4357531.

[14] T. Hosokawa, S. Jarusirisawad, and H. Saito, "Online video synthesis for removing occluding objects using multiple uncalibrated cameras via plane sweep algorithm," in *2009 Third ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, 2009, pp. 1–8. DOI: 10.1109/ICDSC.2009.5289380.

[15] D. Lindlbauer and A. D. Wilson, "Remixed reality: Manipulating space and time in augmented reality," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–13.

[16] Y. Wexler, E. Shechtman, and M. Irani, "Space-time video completion," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, 2004, pp. I–I. DOI: 10.1109/CVPR.2004.1315022.

[17] S. Meerits and H. Saito, "Real-time diminished reality for dynamic scenes," in *2015 IEEE International Symposium on Mixed and Augmented Reality Workshops*, 2015, pp. 53–59. DOI: 10.1109/ISMARW.2015.19.

[18] G. Queguiner, M. Fradet, and M. Rouhani, "Towards mobile diminished reality," Oct. 2018. DOI: 10.1109/ISMAR-Adjunct.2018.00073.

[19] J. Carter, K. Schmid, K. Waters, L. Betzhold, B. Hadley, R. Mataosky, and J. Halleran, *Lidar 101: An introduction to lidar technology, data, and applications*, 2012. [Online]. Available: https://coast.noaa.gov/data/digitalcoast/pdf/lidar-101.pdf (visited on 06/19/2022).

[20] N. Kawai, T. Sato, and N. Yokoya, "Diminished reality based on image inpainting considering background geometry," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 3, pp. 1236–1247, 2016. DOI: 10.1109/TVCG.2015.2462368.

[21] J. Herling and W. Broll, "Pixmix: A real-time approach to high-quality diminished reality," *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 141–150, 2012.

[22] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988. DOI: 10.1007/BF00133570.

[23] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "Patchmatch: A randomized correspondence algorithm for structural image editing," *ACM Trans. Graph.*, vol. 28, no. 3, p. 11, 2009. DOI: 10.1145/1531326.1531330.

[24] C. Barnes, E. Shechtman, D. Goldman, and A. Finkelstein, "The generalized patchmatch correspondence algorithm," Sep. 2010, pp. 29–43, ISBN: 978-3-642-15557-4. DOI: 10.1007/978-3-642-15558-1_3.

[25] E. Adelson, C. Anderson, J. Bergen, P. Burt, and J. Ogden, "Pyramid methods in image processing," *RCA Eng.*, vol. 29, Nov. 1983.

[26] J. Herling and W. Broll, "High-quality real-time video inpainting with pixmix," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 6, pp. 866–879, 2014. DOI: 10.1109/TVCG.2014.2298016.

[27] C. Rother, V. Kolmogorov, and A. Blake, ""grabcut": Interactive foreground extraction using iterated graph cuts," in *ACM SIGGRAPH 2004 Papers*, Association for Computing Machinery, 2004, pp. 309–314, ISBN: 9781450378239. DOI: 10.1145/1186562.1015720.

[28] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand. 2009.

[29] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986. DOI: 10.1109/TPAMI.1986.4767851.

[30] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985. DOI: https://doi.org/10.1016/0734-189X(85)90016-7.

[31] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2015. DOI: 10.1109/tpami.2014.2345390.

[32] A. LukeźIăź, T. Vojıř̌, L. Čehovin Zajc, J. Matas, and M. Kristan, "Discriminative correlation filter tracker with channel and spatial reliability," *Int. J. Comput. Vision*, vol. 126, no. 7, pp. 671–688, 2018. DOI: 10.1007/s11263-017-1061-3.

[33] A. Telea, "An image inpainting technique based on the fast marching method," *Journal of Graphics Tools*, vol. 9, no. 1, pp. 23–34, 2004. DOI: 10.1080/10867651.2004.10487596.

[34] Google, *Fundamental concepts of arcore*, Feb. 2022. [Online]. Available: https://developers.google.com/ar/develop/fundamentals.

[35] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

[36] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani, "Summarizing visual data using bidirectional similarity," *CVPR*, Jan. 2008.

[37] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012. DOI: 10.1109/TPAMI.2012.120.

[38] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. DOI: https://doi.org/10.48550/arXiv.1506.02640.

[39] S. Darabi, E. Shechtman, C. Barnes, D. B. Goldman, and P. Sen, "Image Melding: Combining inconsistent images using patch-based synthesis," *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2012)*, vol. 31, no. 4, 82:1–82:10, 2012.

A ADDING VIRTUAL OBJECTS WITH ARCORE

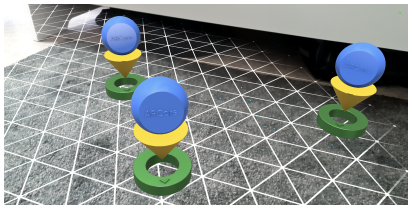


Figure 11: Adding virtual objects with ARCore. The estimated floor plane is visualized, together with feature points (small blue dots) that are used for SLAM. Based on scene light estimation, the specular highlights slightly differ on each object.

B CHALLENGING APPLICATIONS OF GRABCUT

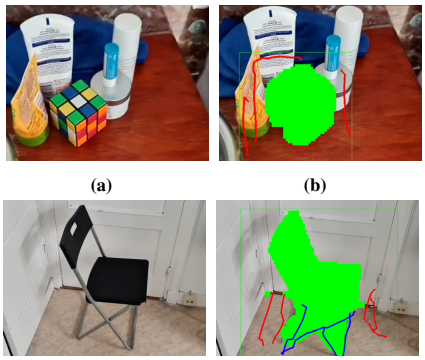


Figure 12: Challenging cases of object selection solved by GrabCut: (a) The algorithm can select an object amidst a diverse collection; and (b) large, heterogeneous objects can be selected, enabling many practical applications in combination with inpainting.

C INPAINTING OF DESTROYED TEXTURES

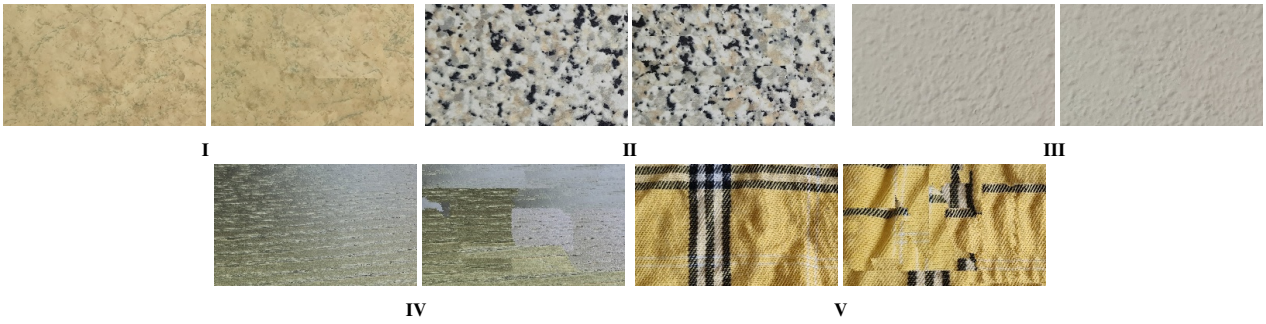


Figure 13: Closeups of inpainting a rectangular subregion of various textures. The shown region has been destroyed in the input to the inpainting algorithm. For each pair, the left image is the ground truth (before destruction), and the right image is the result of inpainting.