User-guided Pedestrian and Object Removal

Antonio Haro
Nokia
425 West Randolph Street
Chicago, IL 60606, USA
antonio.haro@nokia.com

Abstract

Street-level image collections now exist for a number of locations around the world. While the focus is on capturing streets and buildings, pedestrians are commonly captured. Pedestrian faces are identified and blurred in different systems to address privacy concerns but leave their possibly identifiable bodies intact. In this paper, we present a user-guided approach to fully remove pedestrians and undesirable scene objects. Both are fully removed given a user created removal mask with our system's gui and using successive image frames with an image inpainting-inspired approach.

1. Introduction

Street-level image collections capture an increasing amount of data at high resolutions. These collections have a wide range of applications including real estate and tourism, among many others. While image capture vehicles focus on buildings and roads, pedestrians are inevitably captured presenting privacy concerns in some countries.

Privacy can be increased by finding human faces in street-level image collections and blurring them [9]. However, pedestrian bodies are not processed which can be identifiable even without a face. In addition, temporary or undesirable objects captured during data collection can lower the final user experience or usability of the imagery.

In this paper, we present an approach for completely removing pedestrians and undesirable objects with minimal user interaction with a simple gui. The gui helps users create selection layers comprised of simple shapes to quickly and accurately label removal regions. Users are not required to have image editing experience to remove pedestrians or objects. The graphical operations to actually remove pedestrians and objects from selected regions are fully automatic.

Multiple source images are captured by a vehicle (Section 3.1) with high overlap between successive images. The selection regions provided by a user with our simple gui are used along with the prior and next image frames to remove pedestrians and objects. The approach samples from regions behind pedestrians and objects after performing image registration (Section 3.3).

Separate algorithms are used to remove pedestrians and objects (Sections 3.4 and 3.6) from scenes since pedestrian image regions are typically comprised of multiple image patches with high intensity gradients and large color differences, unlike objects which can be more aggressively segmented. Both algorithms use graphcut-based image segmentation to minimize the size of the user's removal region by copying patches from the prior/next frames in a visually coherent manner. Once the removal region is minimized, an image inpainting algorithm (Section 3.5) is used to fill the remaining uncopied regions.

The removal algorithms and selection gui comprise a system that removes pedestrians and objects with results comparable to skilled graphic artists in a fraction of the time and without requiring prior skills. Our use of manual selections, graphcut-based image segmentation, inpainting and multiple frames allows our approach to handle challenging cases such as crowds of pedestrians at traffic intersections.

2. Previous work

Flores *et al*. [8] remove pedestrians from street-level imagery automatically. Their approach differs from ours in the use of an automatic pedestrian detection algorithm and in-place image patch copying. Our removal process is instead based on graphcut-driven image segmentation using multiple frames. In addition, our method uses a minimal inpainting and context aware-fill to ensure removal in scenes without large planar objects to aid in registration and avoids copying incoherent image patches. Also, their approach is fully automatic whereas ours uses manually defined removal regions. These take additional time to define but help handle challenging situations such as crosswalks and jaywalking pedestrians as well as undesirable scene objects.

Image editing programs provide tools to help users manually remove objects from images. However, these require

978-1-4673-5674-9/12/$31.00 ©2012 IEEE

user expertise and may be time intensive. Criminisi *et al*. [5] combine texture synthesis and inpainting approaches to automatically remove image regions within a user's selection area. The fill region is successively reduced by sampling image patches according to a visual coherence priority ordering along the fill boundary.

Graphcut-based approaches have been presented for various domains including texture synthesis (Kwatra *et al*. [10]) and image segmentation (Rother *et al*. [13], Li *et al*. [11]). These approaches sample image patches directly instead of infilling to preserve detail present in the original image. Our pedestrian and object removal algorithms similarly use graphcuts during removal to minimize infilling.

3. Approach

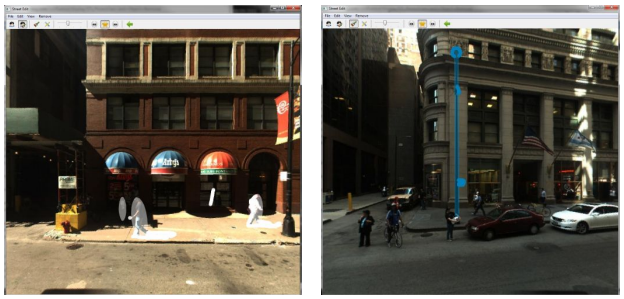
Figure 2 shows an overview of our approach. Users manually select pedestrian and unwanted object regions in individual frames from a previously captured sequence using a custom gui. Multiple frames are registered and merged to maximize image patch copying from nearby frames while shrinking the user's selection regions. Finally, an infilling algorithm is used to fill the small noise-like mask regions that remain in the merged frame. The resulting image is free of pedestrians and objects in regions selected by the user.

3.1. Image capture

Data is collected from a mobile system composed of a 360° LiDAR sensor (Velodyne HDL-64E), six high-resolution cameras, a Ladybug 3 camera, GPS, Inertial Measurement Unit (IMU), and Distance Measurement Instrument (DMI). The Ladybug 3 covers more than 80 percent of a full sphere, with six high quality 1600x1200 Sony CCD sensors, and provides up to 12 MP images at 15 Frames Per Second (FPS). All of these sensors are georeferenced through a GPS and IMU. Spherical images are captured at 4 meter intervals which provides significant image overlap for multi-frame registration.

3.2. Removal region selection

We created a single user gui for pedestrian and object removal (Figure 1) using simple shape selections. Users select pedestrian regions (Figure 1a) using different brush tools including constrained ellipsoidal selection or freehand brushes mimicking pedestrian dimensions. Undesirable objects can be selected in the same user interface (Figure 1b) and users can toggle selections on/off to ensure that all regions of interest are covered before the two removal algorithms are run in parallel on the user's selections. The gui tools help users accurately select removal regions without overselection for best results.



(a) Pedestrians (b) Street light

Figure 1. Interactive removal region selection using ellipsoidal, line and freehand brushes.

3.3. Image registration

Improved pedestrian and object removal is possible using multiple frames. This enables sampling from overlapping scene regions before infilling. Our data capture (Section 3.1) provides a small baseline between images enabling multiframe registration. For a particular frame n, two homographies are computed, one relating frame $n-1$ to n and another from $n+1$ to n. Each homography is computed using SURF [2] features matched between frames using multiple RANSAC [7] iterations with increasingly tighter bounds on the homography.

3.4. Pedestrian removal

After registering frames $n-1$ and $n+1$, pedestrians are separately removed in the user's selection region in frame n. A list of pedestrian regions is created using connected components [4]. The corresponding selection region from frames $n-1$ and $n+1$ is copied into new sub-images $(n-1)'$ and $(n+1)'$. GrabCut [13] segmentation is used on each sub-image with the user's selection region as its mask to remove extraneous or incorrect regions that do not align with frame n. However, this only identifies overcopied image regions such as pedestrian body parts, possibly keeping regions from frame n.

Frame n is processed separately to create a fallback result. User selected pedestrian removal regions are filled using a coarse-to-fine infilling algorithm (Section 3.5). Pixels from the resulting output are used to determine when pixels from frames $(n-1)'$ and $(n+1)'$ are visually incoherent. This is possible if frames $n-1$ or $n+1$ are misregistered or in the case of high scene complexity.

The infilled result is combined with frames $(n-1)'$ and $(n+1)'$ to generate the "merged frame" in Figure 2. The merged frame consists of a complete removal of all pedestrian regions maximizing large patch copying from neighboring frames and minimizing the infill region. The infill region is minimized since infilled regions have lower detail than those from copied image patches.

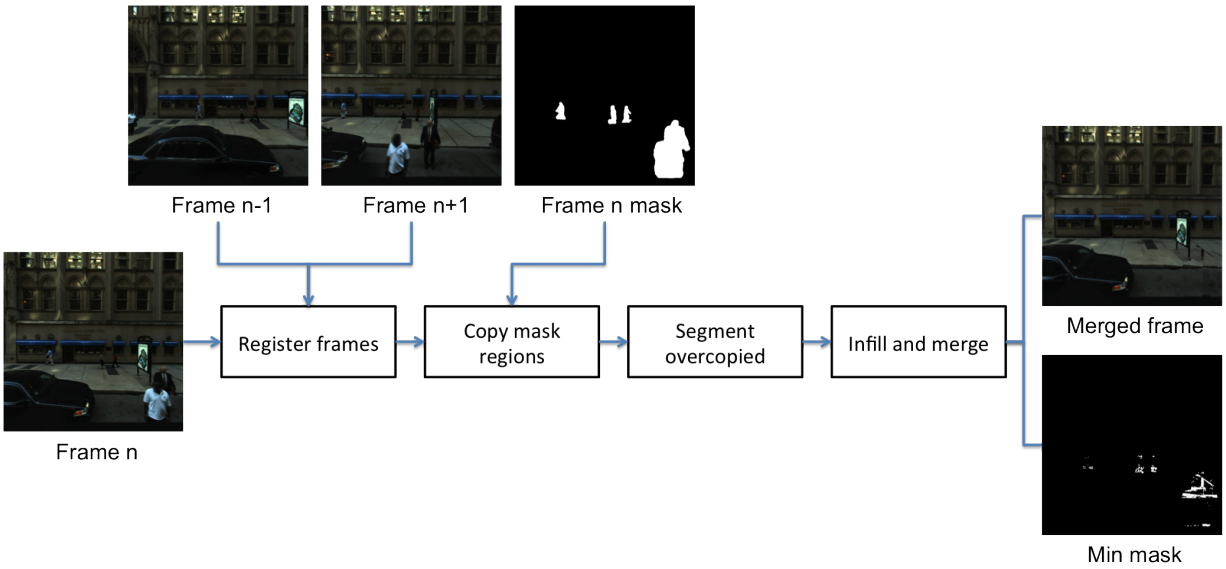


Figure 2. Pedestrian removal overview: Frames $n-1$, n, $n+1$ and mask for frame n are provided as input. The resulting merged frame and minimal fill mask are used to produce the final processed frame n output.

The merged frame and minimal infill mask are generated by individually comparing pixels from $(n-1)'$, $(n+1)'$ and the infilled-only result. The absolute difference between pixel s in the infilled-only result and pixel s in each of the two frames is calculated. The pixel with minimum difference is used in the merged frame, unless its color value differs from the infilled result by more than a threshold (50 in our implementation). If the threshold is exceeded, it is marked as a pixel that needs to be infilled.

Figure 2 shows the user's original selection mask and the resulting minimal infill mask ("min mask"). The majority of the pixels in the user's mask region are sampled and not infilled since the method favors copying entire image patches from registered neighboring frames. Finally, the merged frame is used as input to our infill algorithm (Section 3.5) using the minimized mask as the fill mask. Reducing the fill mask before infilling to resemble lines and noise also has the advantage of improving the performance of the infill algorithm since it performs best with smaller noise-like fill regions.

3.5. Infilling

Our infilling algorithm is an extension of Criminisi *et al*. [5] to better handle pedestrian/object regions at different scales in a single frame. Separate Gaussian pyramids are created for the merged frame, minimal infill mask and the output image. At the top level of the output image's pyramid, the merged frame's fill regions are horizontally interpolated to minimize interpolation artifacts. At all other levels of the output image pyramid, the algorithm proceeds in a coarse-to-fine manner propagating each level's output downward. Infilling is performed for each pyramid level using the corresponding input image and mask from the other pyramids.

3.6. Object removal

Object removal is identical to pedestrian removal up to the computation of the merged frame and minimal infill mask. The algorithm copies even larger image patches than possible for pedestrians using graphcut-based techniques to find patches to copy from frames $n-1$ and $n+1$ into frame n. Using this approach to remove pedestrian regions would cause body portions from multiple frames to be copied into frame n which is undesirable.

In each object removal region, the largest image patches possible are first copied from the previously registered frames $n-1$ and $n+1$ into n in a visually coherent manner. This is framed as a graphcut problem where pixels are assigned as belonging to a source or sink using the max-flow algorithm [3]. Merging successive frames is similar to the patch compositing operation of Kwatra *et al*. [10]. For each object removal region, we create a graph comprised of overlapping pixels from frames $n-1$ and $n+1$. Terminal weights sized ∞ are attached from each graph node on the left side of the removal region to the source. Similarly, terminal weights sized ∞ are attached from each graph node on the right side of the removal region to the sink.

Horizontal and vertical gradients are computed for pixels within the current object removal region. Gradients are used to set the edge weights in the graph. We use the enhanced edge weights from [10]:

$$D(G,s,t) = ||G(s)|| + ||G(t)|| \quad (1)$$

(a) (b)

(c)

Figure 3. Object removal results.

$$W(s,t,A,B) = \frac{||A(s) - B(s)|| + ||(A(t) - B(t)||}{D(G_A^d, s, t) + D(G_B^d, s, t)} \quad (2)$$

where A and B represent registered frames $n-1$ and $n+1$ respectively and W is the edge weight between pixels s and t between the two frames. $A(s)$ and $B(s)$ are the color values for pixel s in each frame and G_A^d and G_B^d are the gradients in direction d in each frame. Max-flow is computed for this graph to find a seam boundary between pixels labeled source and sink in the overlapping removal region.

The computed seam shows how to merge the two adjacent frames but not which frame should be considered the left or right side of the seam. In texture synthesis, there is no ambiguity because the two sides of the seam overlap only along edges. In this multi-frame case, the two sides of the seam completely overlap since they have been previously registered. The approach determines left and right frame assignment by finding which configuration has the minimum summed gradient magnitude along the computed seam boundary.

Once the frames for the current object selection region are merged into a new image M, the original object removal selection region is used as input to automatically identify any object parts that remain in the selection region. Any remaining parts will be much smaller than the original user selection so only this minimal area is infilled. The intuition is that the pixels in M along the user's selection boundary are probably background pixels if the prior step computed a good seam. This process uses a similar graph and edge weights as in Li *et al.* [11] again followed by max-flow. Computing max-flow on this graph has the effect of pushing the mask inward where possible by identifying patches on the mask boundary that can be directly copied from the background.

K-means clustering [6] is used with 64 clusters for foreground regions (F) and background regions (B) with cluster mean colors denoted by $\{K_n^F\}$ and $\{K_n^B\}$ respectively. For each node s, the minimum distance from its color $M(s)$ to the foreground clusters is computed with $d_s^F = \min_n ||M(s) - K_n^F||$ and $d_s^B = \min_n ||M(s) - K_n^B||$ for background clusters. The likelihood energy L at each pixel's graph node is then:

$$L(s=F) = \frac{d_s^F}{d_s^F + d_s^B}, L(s=B) = \frac{d_s^B}{d_s^F + d_s^B} \quad (3)$$

Edge weight function W is similar to Li *et al.*'s except gradient magnitude is used instead of color differences and an additional factor λ is used for additional smoothness:

$$W(s,t,M) = \lambda \frac{1}{||G_M^d(s)|| + \epsilon} \quad (4)$$

with $\epsilon = 0.01$ and $\lambda = 150$. G_M^d is the gradient in the merged frame M with d set to x or y depending on whether the edge between s and t is horizontal or vertical. The resulting minimal infill mask for the current object is analogous to the merged infill mask in the pedestrian removal case. The final steps are to run the same infilling algorithm as for pedestrians on the minimal object selection mask and to repeat this process for all other remaining object selection regions.

4. Results

We used the described system to remove pedestrians and objects from various scenes. Figure 4 shows results varying in pedestrian density, lighting, time of day, and closeness to the camera demonstrating the effectiveness of the approach. In contrast to prior work, pedestrians and objects are always fully removed with no portion of the original region left in the resulting image.

The approach can remove pedestrians under challenging conditions including dense crowds at intersections (Figure 4b). Objects in untextured regions (Figure 3a) are removed by sampling and infilling between frames. Highly detailed regions such as the building ornamentation in Figure 3b are sampled from other frames for full removal instead of infilled to preserve detail. Figure 3c shows full removal of a large object with limited parallax.

Our system was mostly written in Python. The implementation is unoptimized though parallelized to take advantage of multicore hardware. The inpainting portion of the algorithm is in C++ and exposed to Python. During inpainting, image patches are matched using approximate k-d trees with FLANN [12]. Additional speedups might be possible using PatchMatch [1] as well.

Typical scenes with multiple pedestrians and objects take ~1.5 minutes to process on 2.53Ghz Intel Xeon E5630, not counting region selection time which depends on the user. Inpainting is the slowest portion, taking almost half of the total time. The approach benefits from having extremely small inpainting regions as a result of the earlier graphcut segmentation-based steps.

5. Conclusions and future work

We presented an approach to remove pedestrians and objects from street-level image collections. Collection owners use a gui to select pedestrian and object regions using high-level operations and without requiring image editing experience. Multiple frames are used in addition to the selected regions to remove pedestrians and objects completely from scenes using a graphcut-driven inpainting algorithm. The approach falls back to pure infilling in large low parallax removal regions and scenes with misregistration or poor graphcut segmentation.

Users can iteratively experiment with differently shaped removal regions in the current implementation, though further speed improvements are needed to achieve real-time removal. Frame registration and k-d trees can be precalculated or computed while users select image removal regions. Registered frames can be used to suggest removal regions based on image differences. Pedestrian and object selection can be sped up using Lazy Snapping [11] to guide mask selection. Frame matched rotoscoping can be used to propagate selected regions across multiple frames with minimal additional work from users when removing pedestrians and objects from sequences.

References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), Aug. 2009. 5

[2] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008. 2

[3] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *EMMCVPR*, pages 359–374, 2001. 3

[4] F. Chang, C. J. Chen, and C. J. Lu. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93:206–220, 2004. 2

[5] A. Criminisi, P. Perez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *Trans. Img. Proc.*, 13(9):1200–1212, Sept. 2004. 2, 3

[6] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley Press, New York, 2001. 4

[7] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. 2

[8] A. Flores and S. Belongie. Removing pedestrians from Google Street View images. In *IEEE International Workshop on Mobile Vision*, San Francisco, CA, June 2010. 1

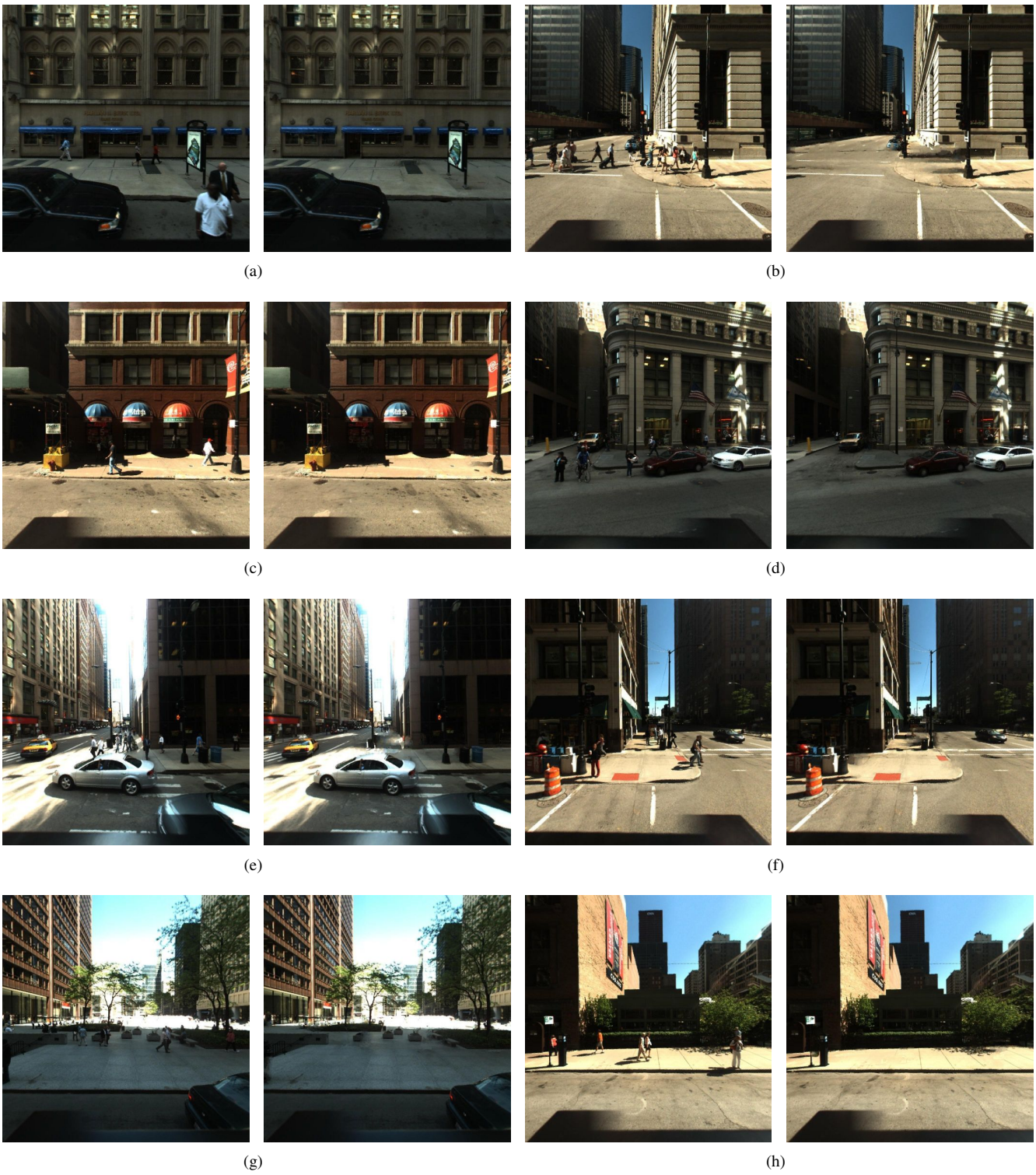
[9] A. Frome, G. Cheung, A. Abdulkader, M. Zennaro, B. Wu, A. Bissacco, H. Adam, H. Neven, and L. Vincent. Large-scale privacy protection in Google Street View. In *IEEE International Conference on Computer Vision*, 2009. 1

[10] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22(3):277–286, July 2003. 2, 3

[11] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum. Lazy snapping. *ACM Trans. Graph.*, 23(3):303–308, Aug. 2004. 2, 4, 5

[12] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application (VISSAPP'09)*, pages 331–340. INSTICC Press, 2009. 5

[13] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, Aug. 2004. 2



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 4. Pedestrian removal results.