Multi-View Alpha Matte for Free Viewpoint Rendering

Daniel Herrera C., Juho Kannala, and Janne Heikkilä

Machine Vision Group, University of Oulu, Finland,
{dherrera,jkannala,jth}@ee.oulu.fi

Abstract. We present a multi-view alpha matting method that requires no user input and is able to deal with any arbitrary scene geometry through the use of depth maps. The algorithm uses multiple observations of the same point to construct constraints on the true foreground color and estimate its transparency. A novel free viewpoint rendering pipeline is also presented that takes advantage of the generated alpha maps to improve the quality of synthesized views over state-of-the-art methods. The results show a clear improvement on image quality by implicitly correcting depth map errors, providing more natural boundaries on transparent regions, and removing artifacts.

1 Introduction

Transparency in a scene is often desirable and usually unavoidable. It can be the result of hair, semi-transparent materials, motion blur, or even aliasing. It gives the objects in the scene a realistic look as boundaries in real scenes are usually not pixel sharp.

A very popular application that suffers greatly from transparency artifacts is free viewpoint rendering. The goal is to render new views by interpolating from nearby cameras [1]. New view synthesis is particularly useful for 3D TV. Mixed boundary pixels produce ghosting artifacts in the synthesized view that significantly reduce its quality.

This paper deals with estimating the transparency of objects in an image, also known as an alpha map, using multiple views of the scene. It is a challenging problem since the transparency can have different causes and the object boundaries are often complex (e.g. hair). We also address the application of the obtained alpha maps to the free viewpoint rendering problem to improve the quality of novel views.

1.1 Alpha matting background

The literature contains many single view alpha matte estimation algorithms, including a comprehensive online benchmark [2]. Due to the under constrained nature of the problem they all require user input to identify some pure foreground and background regions to be used as examples. Some accept a sparse labeling

in the form of strokes, while others require dense labeling in the form of a tri-map. This is a considerable limitation since the need for user input limits the applicability of the algorithms, particularly for video sequences.

Single view alpha matting algorithms can be divided into three categories according to the assumptions made for an image: color sampling, alpha propagation, and optimization methods. Color sampling methods (e.g. shared sampling [3]) take samples from nearby labeled regions. They assume color smoothness to interpolate the alpha values between the labeled regions, usually requiring a dense tri-map. Alpha propagation methods assume that the alpha values are correlated to some local image statistics and use this to interpolate the alpha values (e.g. Closed form matting [4]). These methods often allow sparse user input. Optimization methods combine the previous two approaches to exploit their strengths (e.g. Robust matting [5]). Although very impressive results have been shown for single view alpha matting [2] it is expected that a multi-view approach would improve the existing methods since more information is available and several observations of the same point can be used.

Zitnick et al. presented a free viewpoint rendering system that estimates the alpha matte along depth discontinuities [6]. It uses a variant of Bayesian matting [7] to estimate colors and opacities for mixed boundary pixels. Although the stereo and rendering is multi-view, the matting is performed using a single view. Moreover, they assume a fixed width boundary which limits the applicability in scenes with large semi-transparent regions.

Hasinoff et al. [8] propose a method to estimate transparency at object boundaries using boundary curves in 3D space. They use a multi-view approach but limit theirselves to mixed boundary pixel transparency. Intrinsic material transparency is not addressed and objects are assumed to be opaque. Joshi et al. [9] suggest a multi-view variance measure to estimate transparency. The approach computes a tri-map and propagates color statistics. This imposes limitations on the color statistics of the scene. Moreover, it does not use all available information by using only the variance of the samples.

Wexler et al. [10] present a multi-view environment matting algorithm to estimate the light transfer function of a foreground object (e.g. a magnifying glass). They include alpha estimation in their algorithm but only handle planar backgrounds in their paper. Moreover, they assume an alpha value independent of viewpoint, which limits the algorithm to planar foregrounds as well. The most closely related work is that of Wexler et al. in [11]. They developed a multi-view approach to alpha matte estimation under a Bayesian framework. They show very good results but limit their model to planar layers. Moreover, their model has an alpha value independent of view, which is not suitable for mixed boundary pixels.

The goal of our matting stage is to generate a layered depth image [12] from each input camera. However, we focus on the construction of this LDI from real world images while estimating transparency. Even modern LDI approaches like [13] suffer from artifacts due to mixed pixel boundaries and transparency.

1.2 Free viewpoint rendering background

A review of the latest free viewpoint rendering methods [1] shows that one of the dominant approaches is to calculate a depth map for each image and then warp the pixel colors to the new view using camera calibration information. The problem with this approach is that traditional depth maps have only a single depth per pixel and do not take transparency into account. This results in ghosting artifacts. Recent methods like Müller et al. [14] attempt to discard mixed pixels to remove the artifacts. Yet, this approach discards information, suffers from unnaturally sharp boundaries, and still produces artifacts for complicated semi-transparent regions.

Our approach also shares a strong similarity with Fitzgibbon et al. [15]. We use a similar scanning of the optical rays to find matching colors amongst the images. Our approach is novel in that it uses linear constraints on RGB space to estimate the true color of semi-transparent points while their approach ignores transparency issues.

2 Modeling transparency

When the transparency and color of an object are unknown the observed color of a pixel can be the result of different situations. As mentioned in [10], if the background is known to be white and the pixel is a 50% combination of red and white, this can be due to any of the following:

1. Object is pink.
2. Object is red with a transparency of 50%.
3. Object is red but covers only 50% of the pixel (mixed boundary pixel).

Both transparency types are view dependent. The former because light rays will traverse different paths through the object, and the latter because a 3D point observed from a different view might not be a 2D boundary pixel any more.

Our model for a semi-transparent pixel p in image i is described by the following matting equation:

$$M_i = \alpha_i F + (1 - \alpha_i) B_i \tag{1}$$

The observed color M_i is a mixture of the foreground and background colors. It assumes a Lambertian surface, which results in a single foreground color F shared by all images. Yet, the background color B_i and alpha value α_i are view dependent. Because most of the work is done individually for each pixel, the index p is omitted.

3 Multi-view alpha estimation

Our algorithm requires the camera projection matrix and depth map for each input image. Using this information all pixels can be back-projected into 3D

Algorithm 1 Multi-view alpha algorithm

for all $i \in$ Images **do** ▷ Sample collection
  for all $p \in$ Image(i) **do**
    object_cluster$_i(p) \leftarrow FindCluster\,(p, depth_{min})$
    depth$_{obj} \leftarrow depth$(object_cluster)
    background_cluster$_i(p) \leftarrow FindCluster\,\left(p, \text{depth}_{obj} + \epsilon_f\right)$
    $B_i(p) \leftarrow ref\_color$(background_cluster)
  end for
end for
for all $i \in$ Images **do**
  for all $p \in$ Image(i) **do**
    sample_set $\leftarrow \emptyset$ ▷ Sample assembly
    for all pixel $\in$ object_cluster$_i(p)$ **do**
      $j \leftarrow$ pixel.image
      sample.M $\leftarrow$ pixel.color
      sample.B $\leftarrow B_j$(pixel)
      If sample is stable **Then** add to sample_set
    end for
    Project samples to main constraint ▷ Alpha estimation
    $F_i(p) \leftarrow$ farthest color along RGB ray
    $\alpha_i^*(p) \leftarrow \frac{\|M_i(p) - B_i(p)\|}{\|F_i(p) - B_i(p)\|}$
  end for
end for
Minimize energy using graphcut ▷ Alpha smoothing

world space and several observations of the same scene point can be grouped together. The main objective of the algorithm is to estimate B_i, F, and α_i for each pixel. Because we can obtain several samples for a scene point and its background, alpha estimation can be done pixel-wise and no tri-map or user input is needed.

Our method is summarized in Algorithm 1. It can be divided into four stages. First, color samples are collected for each pixel and its background. Second, the samples are assembled together into geometric constraints. Then, using these constraints the true color and alpha value are estimated. These first three stages treat each pixel individually. The final stage uses a graph cut minimization to enforce spatial smoothness in the alpha map. Each stage is described in detail in the following sections.

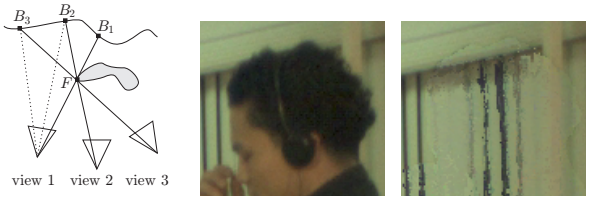
3.1 Sample collection

Instead of using neighbor pixels from the same image, as is common in most alpha matting algorithms, our approach takes advantage of the fact that multi-view systems observe a point in the scene several times from different angles. Because of parallax the point is observed each time with a different background. The background itself can often be directly observed in a different image, as illustrated in Figure 1.

The colors observed for the same point are grouped together in clusters. Each cluster can have as many samples as there are cameras in the system. The algorithm scans the pixel's optical ray to find the first two distinct clusters in space. The first is denoted the foreground object cluster and contains the observed colors from all the views where the corresponding space point is visible. The second

Algorithm 2 Find color cluster algorithm

```
function FindCluster(pixel, depth_0)
    c* ← ∅                                              ▷ Best cluster found
    for d = depth_0 ... depth_max do
        c_d ← ∅                                         ▷ Cluster at depth d
        p_ref(x, y, z) ← back-project pixel using d
        for j ∈ Views do
            p_j(u, v, w) ← project p_ref to view j
            w_map ← nearest_neighbor(depth_j, u, v)
            if |w − w_map| ≤ ε_w then
                rgb ← bilinear_interpolation(j, u, v)
                Add rgb to c_d
            end if
        end for
        if score(c_d) > score(c*) then
            c* ← c_d                                    ▷ New best cluster
        end if
        if d − depth(c*) ≥ ε_f then
            break                                       ▷ No cluster found for a while, end search
        end if
    end for
    return c*
end function
```



(a) (b) Original (c) Background

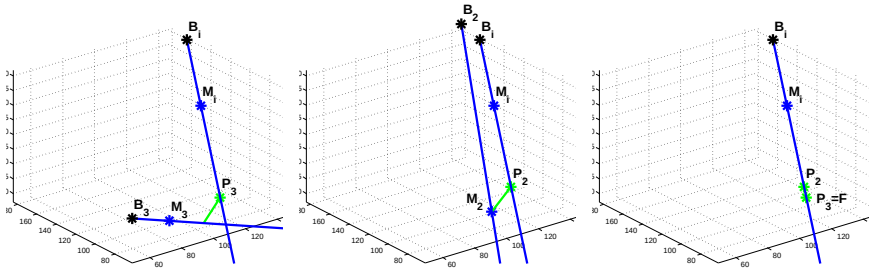
Fig. 1: Background recovery. (a) The background color for views 2 and 3 is directly observed by view 1. Image taken from [8]. (b) and (c) show an example of the recovered background.

is the background cluster and contains observed colors for the background. The method of collecting samples is detailed in Algorithm 2.

Each discretized depth d along the pixel's optical ray is projected onto the epipolar line of the other images. If the expected depth and the pixel's depth are similar, the pixel is added to a cluster at this position d. Due to noise and necessary tolerances, this procedure will obtain many similar clusters at nearby depths. These are essentially the same cluster at slightly different displacements. To select the best cluster for a point in space the candidate clusters are ranked according to the following formula:

$$score = \underset{M_k \in \text{cluster}}{median} \left(\| M_{\text{ref}} - M_k \| \right) \tag{2}$$

where the median is over all the samples in the cluster. The choice of reference color M_{ref} differs for the object and background clusters. For the object cluster it is the pixel color of the current view M_i. This creates a bias towards higher alpha values as it tries to find similar colors, but maximizes the chances of finding a match with the same foreground color. For the background cluster, the sample



(a) Different background (b) Similar background (c) Final constraints

Fig. 2: Projection of samples onto main constraint. According to the final result in (c), P_3 is selected as $F_i(p)$ and M_i is assigned an alpha of 42%.

M_j is selected whose camera j is closest to i because camera calibration and depth map noise have a smaller impact on nearby cameras.

The object cluster has the observed colors for this point $\{M_j | j \in$ [views where point is not occluded]$\}$. The background cluster is discarded and only the reference color is stored for the following stages. This reference color becomes the background color for the current pixel B_i. If only one cluster is found then no estimation is performed for this pixel (i.e. $F_i(p) = B_i(p) = M_i(p), \alpha_i^*(p) = 0$).

3.2 Sample assembly

From (1) it can be seen that M_i lies on the line segment between B_i and F in RGB space. Because we do not know F we can use each sample to create a ray in RGB space that starts from B_i and passes through M_i. One ray is constructed for each entry in the object cluster. For an entry from image j, the background is obtained from the corresponding B_j.

3.3 Alpha estimation

Since M_i is directly observable and B_i was estimated in the previous stage, the remaining task is to estimate F in Eq. (1). However, in order to facilitate new view synthesis, we would like to have an image-based representation for the color of the foreground objects (i.e. pre-rendered per source view) and hence we estimate the foreground layers $F_i(p)$ in a view dependant manner. To this effect, using the assembled rays in RGB space one can derive two types of constraints, as shown in Figure 2 and detailed in this section.

The first type of constraint is derived from the fact that all pixels belonging to the same foreground object cluster should share the same F, thus all rays should intersect at F (Fig. 2a). This is the underlying idea used for triangulation in standard blue screen matting [16]. However, rays originating from backgrounds with very similar color have very unstable intersection points, demonstrated in

Figure 2b. In the case where the backgrounds are exactly the same color the rays are collinear.

The second type of constraint captures the idea that M_i lies in the line segment $\overline{B_iF}$. This means that F must lie on the $\overrightarrow{B_iM_i}$ ray at least as far as M_i. This gives an upper bound to the observed alpha values. This is specially useful for samples which are always observed with similar background colors and their intersection is therefore unreliable. If at least one image sees the true color of the point (i.e. $M_i = F$), which is a common case for mixed pixels at object boundaries, then we can still recover the true alpha value even if the background is non-textured.

To estimate $F_i(p)$ for a pixel p in image i, we first consider the ray defined by B_i and M_i to be the main ray. Each M_j from the foreground object cluster of the pixel p is then projected onto this ray in one of two ways, depending on the intersection angle between the rays:

$$P_j = \begin{cases} \left((M_j - B_i) \cdot \hat{d}_i\right) \hat{d}_i + B_i & \text{if } \angle ij \leq \epsilon_\angle \\ \text{Ray intersection} & \text{else} \end{cases} \tag{3}$$

where $\hat{d}_i$ is the ray direction from B_i to M_i.

If the angle between rays is lower than the threshold, the intersection is considered unreliable. Because this is caused by similar backgrounds, M_j can be directly projected onto the main ray (Fig. 2b). If the intersection angle is above the threshold, the point on $\overrightarrow{B_iM_i}$ closest to $\overrightarrow{B_jM_j}$ is used as the sample's projection P_j (Fig. 2a).

Once all samples have been projected onto the main ray (Fig. 2c), $F_i(p)$ is taken as the farthest P along the ray. The alpha value is then calculated as the distance of M_i to B_i relative to $F_i(p)$:

$$\alpha_i^*(p) = \frac{\|M_i(p) - B_i(p)\|}{\|F_i(p) - B_i(p)\|} \tag{4}$$

3.4 Alpha smoothing

The previous stage estimates the foreground and background colors as well as the alpha value for each pixel. However, this is done independently for each pixel and is noisy. We can improve this estimate by taking spatial information into account. Since the alpha gradient directly contributes to the total gradient, we assume that regions with low color variation imply low alpha variation. This is exploited by applying a graph cut algorithm [17] to the obtained alpha values. The continuous interval $[0, 1]$ of alpha values is discretized into 100 labels with constant separation. The energy to be minimized is of the standard form:

$$E = \sum_{p \in I} E_d(p) + \lambda \sum_{p,q \in I} E_s(p, q) \tag{5}$$

where λ controls the weight of the spatial term relative to the data term.

The data term controls how much the new alpha deviates from the previous estimation. A truncated L1 norm is used as a robust cost measure:

$$E_d(p) = min\left(\left|\alpha(p) - \alpha^*(p)\right|, \epsilon_\alpha\right) \tag{6}$$

The spatial term penalizes variations in alpha value where the image gradient is low. However, if the depth of the recovered clusters differs, the spatial term is set to zero because the pixels belong to different objects.

$$E_s(p,q) = \begin{cases} \frac{min(|\alpha(p)-\alpha(q)|,\epsilon_\alpha)}{|\nabla M(p,q)|+1} & \text{if } |Z_i - Z_j| < \epsilon_z \\ 0 & \text{else} \end{cases} \tag{7}$$

3.5 Noise considerations

There are three sources of noise for the algorithm: camera calibration parameters, depth map, and RGB noise. Each of these was analyzed to determine their impact in the estimation.

Camera calibration errors lead to an inaccurate optical ray for each pixel. The effect is directly visible in the plot of the epipolar line. We tested this effect in our datasets [6] using both the provided camera calibration and estimating the parameters using off-the-shelf structure from motion techniques. In both cases the epipolar line's inaccuracy was visibly less than half a pixel. We therefore assume sufficiently accurate calibration parameters.

The depth map on the other hand, presents considerable errors. Even though the quality of the depth map can be improved by using better stereo methods, it will still likely contain inaccuracies. This is taken into account in the sample collection stage. The clustering of the backprojected points and ranking of the clusters provides robustness against some depth map errors.

RGB noise has a stronger impact on pixels where the observed and background colors are similar. This can be measured by the length of each sample constraint (i.e. $\|M_j - B_j\|$). Constraints with a small length have an unstable direction and its projection is unreliable. Therefore, if the length is smaller than a given threshold the constraint is ignored. If the main constraint is to be ignored then no alpha value is calculated for the pixel (i.e. $F_i(p) = M_i(p), \alpha_i^*(p) = 1$).

4 Free viewpoint rendering

As an application for the obtained alpha maps, a free viewpoint rendering system was developed that handles transparent layers appropriately. The algorithm takes four layers as input: left background, left foreground, right background, and right foreground. Each layer has a depth map, an RGB texture, and an

alpha map. The layer components are obtained directly from the output of the alpha estimation output for the left and right views. Areas where no alpha estimation could be performed have an empty foreground, with the original image color and depth used for the background layer. Left and right layers are merged to produce the final background color B_n, foreground color F_n, and alpha value α_n.

Each layer is first warped to the novel viewpoint independently. Small cracks that appear due to the forward warping are filled using the same crack-filling algorithm presented in [14]. Cracks are found by looking for depth values that are significantly larger than both neighboring values in horizontal, vertical, or diagonal directions. The median color of neighboring pixels is then used to fill in the cracks. Warped background layers are then combined pixel by pixel using a soft z threshold:

$$B_n = \begin{cases} \frac{d_l B_l + d_r B_r}{d_l + d_r} & \text{if } \left|Z_l^b - Z_r^b\right| < \epsilon_z \\ B_l & \text{else if } Z_l^b < Z_r^b \\ B_r & \text{else} \end{cases} \tag{8}$$

where d_l and d_r are the distances from the novel view's camera center to the left and right views' camera centers respectively. Merging of the foreground layers must take transparency into account. First the left and right foreground colors are combined. If both foreground pixels are close to each other, the final foreground color is interpolated between the two. If they are far apart, it is assumed that they represent different transparent layers and are thus combined using (1):

$$F_n = \begin{cases} \frac{d_l F_l + d_r F_r}{d_l + d_r} & \text{if } \left|Z_l^f - Z_r^f\right| < \epsilon_z \\ \alpha_l F_l + (1 - \alpha_l) F_r & \text{else if } Z_l^f < Z_r^f \\ \alpha_r F_r + (1 - \alpha_r) F_l & \text{else} \end{cases} \tag{9}$$

$$\alpha_n = \begin{cases} max\left(\alpha_l, \alpha_r\right) & \text{if } \left|Z_l^f - Z_r^f\right| < \epsilon_z \\ 1 - (1 - \alpha_l)(1 - \alpha_r) & \text{else} \end{cases} \tag{10}$$

Finally, (1) is applied to produce the final output color using F_n, α_n, and B_n. Because the foreground layers already have an alpha channel no extra processing is necessary for the transparent regions or the boundary mixed pixels.

5 Results

5.1 Alpha maps

Figure 3 shows the obtained alpha maps for the well known ballet and breakdancers datasets [6]. A close up of two relevant regions is presented in Figure 4. The dancers in the scene have a mixed pixel boundary several pixels wide, as seen in 4. The alpha values for these mixed pixels were succesfully recovered without any user input. Hair presents a challenge for alpha estimation and even

though the semi-transparent region of Figure 4 has an uneven width, its alpha matte was also extracted properly. The central region of the breakdancer has no alpha values because the background could not be observed in any of the images. Yet the mixed pixel boundary was also detected.

Fig. 3: Extracted alpha maps for the characters in the scenes.

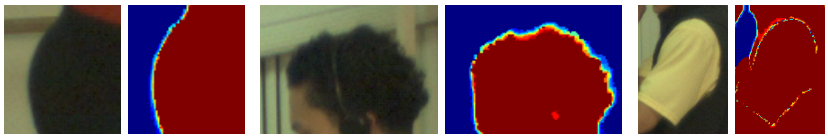


Fig. 4: Close up of semi-transparent regions. **Left:** Boundary pixels. **Middle:** Semi-transparent hair. **Right:** Incorrect estimation.

Figure 4 shows how the algorithm labels as semi-transparent the area where the yellow sleeve and black vest meet. These pixels are indeed mixed pixels as can be observed by the mixture of yellow and black on the border. However, when the algorithm classifies them as transparent, it incorrectly assumes that they are mixed with the wall behind.

5.2 Free viewpoint rendering

A novel view generated using our method is presented in Figure 5. Müller et al.'s state-of-the-art method presented in [14] was implemented and used as a comparison. At a broad scale, both algorithms produce novel views of similar quality. Close ups of the most relevant differences are presented in Figure 6.

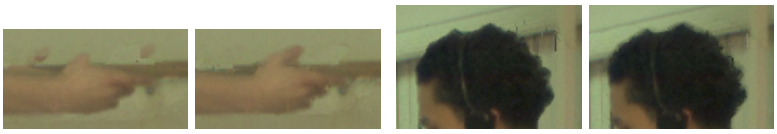
On Figure 6a it can be observed how an error in the depth map causes the thumb to be warped incorrectly by Müller et al.'s method. The alpha matte estimation stage of our algorithm succesfully recovers from this error in the depth map and assigns the thumb to the proper place.

A semi-transparent region made of hair is presented on Figure 6b. Müller et al.'s method produces an unnaturally sharp and even boundary for the hair. The alpha map obtained with our method allows a more natural look of the hair.

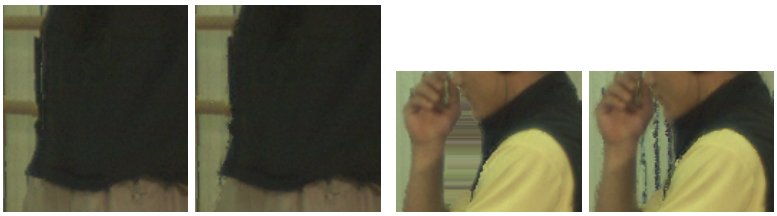
Figure 6c shows an artifact present in Müller et al.'s approach due to the wall being incorrectly assigned to the foreground, similar to a ghosting artifact. Our

(a) Müller et al.'s method [14] (b) Our method using alpha matte

Fig. 5: Comparison of synthesized views from a novel viewpoint.



(a) Correction of depth innacuracies. (b) Improved transparency handling.



(c) Removal of line artifact on left border. (d) Naïve hole filling vs. recovered background.

Fig. 6: Comparison of synthesized views from a novel viewpoint.
Left column: Müller et al.'s method. **Right column:** our proposed method.

method does not suffer from this type of artifacts. However, our method presents more noise on the border. The noise suggests that the alpha smoothing stage could be improved. The current algorithm enforces spatial smoothness only on the alpha map and not in the foreground or background color maps.

Finally, Figure 6d shows that the naïve hole filling approach used in [14] is not suited to big holes in the background. Because our method uses the entire dataset for the alpha estimation stage, the background can be recovered from other images and no hole filling is necessary.

6 Conclusions

We presented a multi-view alpha estimation algorithm that requires no user interaction. It handles arbitrary scene geometry using pre-computed depth maps.

It automatically detects semi-transparent pixels in the images. The algorithm handles mixed boundary pixels and hair regions correctly estimating their transparency and true colors.

Using the results of the alpha estimation algorithm, a novel free viewpoint rendering pipeline was developed and compared to the state of the art. The alpha estimation stage allowed the free viewpoint rendering algorithm to correct some depth map errors. The obtained results are of high quality and removed several artifacts found in the state-of-the-art methods. Future research can focus on better use of spatial information during alpha estimation and in simultaneous depth and transparency estimation.

References

1. A. Smolic, "3d video and free viewpoint video-from capture to display," *Pattern Recognition*, 2010.
2. C. Rhemann, C. Rother, J. Wang, M. Gelautz, P. Kohli, and P. Rott, "A perceptually motivated online benchmark for image matting," in *CVPR*, 2009.
3. E. Gastal and M. Oliveira, "Shared sampling for real-time alpha matting," *Computer Graphics Forum*, vol. 29, no. 2, 2010.
4. A. Levin, D. Lischinski, and Y. Weiss, "A closed-form solution to natural image matting," *PAMI*, 2007.
5. J. Wang and M. Cohen, "Optimized color sampling for robust matting," in *CVPR*, 2007.
6. C. Zitnick, S. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," *ACM Trans. on Graph. (Proc. SIGGRAPH)*, 2004.
7. Y. Chuang, B. Curless, D. Salesin, and R. Szeliski, "A bayesian approach to digital matting," in *CVPR*, 2001.
8. S. Hasinoff, S. Kang, and R. Szeliski, "Boundary matting for view synthesis," *Computer Vision and Image Understanding*, vol. 103, no. 1, 2006.
9. N. Joshi, W. Matusik, and S. Avidan, "Natural video matting using camera arrays," *ACM Trans. Graph.*, vol. 25, 2006.
10. Y. Wexler, A. Fitzgibbon, and A. Zisserman, "Image-based environment matting," in *Proceedings of the 13th Eurographics workshop on Rendering*, 2002.
11. Y. Wexler, A. Fitzgibbon, and A. Zisserman, "Bayesian estimation of layers from multiple images," in *ECCV*, 2002.
12. J. Shade, S. Gortler, L. He, and R. Szeliski, "Layered depth images," *SIGGRAPH*, 1998.
13. A. Frick, F. Kellner, B. Bartczak, and R. Koch, "Generation of 3d-tv ldv-content with time-of-flight camera," in *3DTV Con*, 2009.
14. K. Müller, A. Smolic, K. Dix, P. Merkle, P. Kauff, and T. Wiegand, "View synthesis for advanced 3d video systems," *EURASIP Journal on Image and Video Processing*, 2008.
15. A. Fitzgibbon, Y. Wexler, and A. Zisserman, "Image-based rendering using image-based priors," in *ICCV*, 2003.
16. A. Smith and J. Blinn, "Blue screen matting," in *Proc. of ACM SIGGRAPH*, 1996.
17. Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *PAMI*, vol. 23, no. 11, 2002.