Blending Multiple Views

Ramesh Raskar
Mitsubishi Electric Research Labs
raskar@merl.com

Kok-Lim Low
University of North Carolina at Chapel Hill
lowk@cs.unc.edu

Abstract

Current blending methods in image-based rendering use local information such as "deviations from the closest views" to find blending weights. They include approaches such as view-dependent texture mapping and blending fields used in unstructured lumigraph rendering. However, in the presence of depth discontinuities, these techniques do not provide smooth transitions in the target image if the intensities of corresponding pixels in the source images are significantly different (e.g. due to specular highlights).

In this paper, we present an image blending technique that allows the use of global visibility and occlusion constraints. Each blending weight now has a global component and a local component, which, respectively, are due to the view-independent and the view-dependent contributions of the source images. Being view-independent, the global components can be computed in a pre-processing stage. Traditional graphics hardware is exploited to accelerate the computation of the global blending weights.

Keywords: *Image-Based Rendering, Image Blending*.

1. Introduction

Image-based rendering (IBR) has become a popular alternative to traditional three-dimensional graphics. Two examples of effective IBR methods are the view-dependent texture mapping (VDTM) [Debevec98] and the light field/lumigraph [Levoy96, Gortler96] approaches. Light field, lumigraph and concentric mosaic [Shum99] require a large collection of input images from cameras, but they make few, if any, assumptions about the geometry of the scene. In contrast, VDTM assumes a relatively accurate geometric model, but requires only a small number of images from input cameras that can be in general positions. Both methods interpolate color values for a desired ray as some weighted combination of input rays. In VDTM this interpolation is performed using a geometric proxy model to determine which pixel from each input image

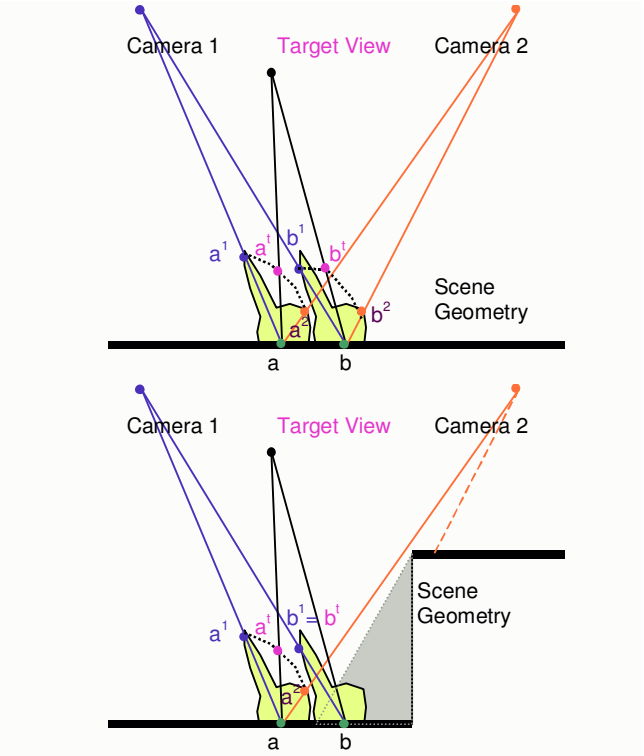


Figure 1. (Top) The blending function at each scene point can be expressed as a weighted combination using local information such as the angles between the target ray and the corresponding source rays. The yellow region indicates local radiance. (Bottom) In the case of view-dependent surface reflectance, and in the presence of depth discontinuities, the neighboring scene points may have very different contributions from the source images.

"corresponds" to the desired ray in the output image. Of these corresponding rays, those that are closest in angle to the desired ray are weighted to make the greatest contribution to the interpolated result.

The blending operation ensures that the influence of a single source image on the final rendering is a smoothly varying function across the target image plane (or, equivalently, across the geometry representing the scene). These smooth weighting functions combine to form a "blending field" that specifies how much contribution each input image makes to each pixel in

the output image. The reconstructed blending field is then used to blend pixels from the input images to form the output image.

The main reasons for blending images are (i) lack of photometric agreement in the source images (caused by change in camera response or view-dependent appearance of scene), and (ii) small errors in registration and depth maps. Blending may ruin crisp imagery due to blurring, but is necessary to mask the unavoidable errors due to the lack of agreement in the source images. Blending methods can be image-space, acting on pixel fragments, or they can be object-space, handling each polygon in the geometric proxy.

Current blending methods use only local information such as "deviations from the closest views" to find blending weights. They include approaches such as view-dependent texture mapping [Debevec98] and blending fields used in unstructured lumigraph rendering [Buehler01]. Both these and other methods (e.g. [Pulli97]) create smooth spatial and temporal transitions in blended source images. In this paper, we focus on the specific problem of blending sparse views in the presence of depth discontinuities. Many previous papers have mentioned the problem of handling feathering across depth discontinuities and proposed solutions based on local computations. We instead use a global image-space approach to create smooth blending weights across the scene geometry.

1.1. Contributions

In this paper, we identify the problem in traditional blending methods and re-state the blending goals to consider global constraints. We present a general algorithm that works in the presence of depth discontinuities. Then we present a fast technique to implement the global blending algorithm using traditional graphics hardware.

2. Previous Work

The blending methods in this paper are inspired by very useful feathering techniques presented by Debevec et al in their paper about view-dependent texture mapping [Debevec98] and by Buehler et al in their paper about unstructured lumigraph rendering [Buehler01]. We are also influenced by a technique devised for projector-based augmented reality to fill in shadows of one projector by other projectors in Shader Lamps [Raskar01].

The appearance of a scene can be described through all light rays (2D) that are emitted from every 3D scene point, generating a 5D radiance function, called the plenoptic function. In a transparent medium the plenoptic function is reduced to four dimensions. In practice, the plenoptic function is sampled from discrete calibrated camera views, and those radiance values that are not sampled have to be represented by interpolating the recorded ones, sometimes with additional information on physical restrictions.

Often, real objects are assumed to be Lambertian, meaning that each point on the object has the same radiance value in all possible directions. This implies that two viewing rays have the same color value if they intersect at a surface point. If specular effects occur, this is no longer true. Two viewing rays then have similar color values, only if their directions are similar and their point of intersection is near the real scene point. To reconstruct an image for a virtual camera, we have to determine those source rays that are closest, in the above sense, to those of this camera. The closer a source ray is to a desired ray, the greater is its contribution to the target color value.

The basic approach to view-dependent texture mapping (VDTM) is put forth by Debevec et al in their Facade image-based modeling and rendering system [Debevec96]. Facade is designed to estimate geometric models consistent with a small set of source images. As part of this system, a rendering algorithm was developed where pixels from all relevant cameras were combined and weighted to determine a view-dependent texture for the derived geometric models. In later work, Debevec et al describe a real-time VDTM algorithm [Debevec98]. In this algorithm, each polygon in the geometric model maintains a "view map" data structure that is used to quickly determine a set of three input cameras that should be used to texture it. Like most real-time VDTM algorithms, this algorithm uses hardware supported projective texture mapping for efficiency.

Buehler et al listed a set of desirable goals that an ideal image-based rendering algorithm should satisfy when blending multiple views [Buehler01]. It includes, among others, the following two main goals:

1. **Near-view fidelity**

   Epipole consistency. When a desired ray passes through the center of projection of a source camera, it can be trivially reconstructed from the ray database (assuming a sufficiently high-resolution input image and the ray falls within the camera's field-of-view). In this case, an ideal algorithm should return a ray from the source image. An algorithm with epipole consistency will reconstruct this ray correctly without any geometric information.

   Minimal angular deviation: In general, the choice of which input images are used to reconstruct

a desired ray should be based on a natural and consistent measure of closeness. In particular, source image rays with similar angles to the desired ray should be used when possible.

2. Continuity

When one requests a ray with infinitesimal small distance from a previous ray intersecting a nearby point on the geometric proxy, the reconstructed ray should have a color value that is correspondingly close to the previously reconstructed color. Reconstruction continuity is important to avoid both temporal and spatial artifacts. For example, the contribution due to any particular camera should fall to zero as one approaches the boundary of its field-of-view [Pulli97], or as one approaches a part of a surface that is not seen by a camera due to visibility [Raskar01].

In this paper we focus on the goal of achieving continuity. Both authors [Debevec98, Buehler01] accept that spatial continuity is not provided due to visibility changes. As described below, algorithms based only on local information cannot achieve smoothness across depth boundaries. The VDTM algorithm of [Debevec98] uses a triangulation of the directions to source cameras to pick the "closest three". Even if the proxy is highly tessellated, nearby points on it can have very different triangulations of the "source camera view map", resulting in very different reconstructions. While this objective is subtle, it is nonetheless important, since lack of such continuity can introduce noticeable artifacts.

In [Buehler01, Heigl99], the authors use a very large number of views, and pixels are blended using views that are very similar and captured by the same (video) camera. Such dense sampling avoids most of the artifacts even if the target pixel is blended from very few source pixels. In later work [Matusik01], in relation to the sparse unstructured light field blending, the authors attempt to handle visibility changes. However, since all the computations are based locally on individual vertices (or triangles), global smoothness on the scene geometry cannot be achieved.

In Shader Lamps [Raskar01], multiple projectors are used to project imagery onto real-world 3D surfaces to simulate interesting lighting effects. Shadows from one projector due to occlusion are "filled in" by other projectors, with feathering around the shadowed region. Since the illuminated surfaces are diffuse, the blending weights are computed during pre-processing and do not need to be updated. Furthermore, the final blending is automatically done in the "real world" and hence there is no need to merge the source images.

3. Local and Global Constraints

Despite the formalization of the blending problems, the previous IBR algorithms attempt to solve the problem by considering one-fragment at a time. This only works well when

(i) the surface is diffuse so that radiance is the same in all directions and corresponding pixels have very similar intensities and
(ii) there are no occlusion boundaries so that the relative ordering of corresponding pixels in any local neighborhood is the same, resulting in continuous functions without gaps.

Consider the example in Figure 1. In the top part of the figure, the target value a^t is a weighted combination of radiance in the directions of Camera 1 and Camera 2, i.e. a^1 and a^2. Similarly, b^t is a weighted combination of b^1 and b^2. Thus, nearby points a and b in the scene combine in a very similar fashion.

In the presence of a depth discontinuity (bottom part

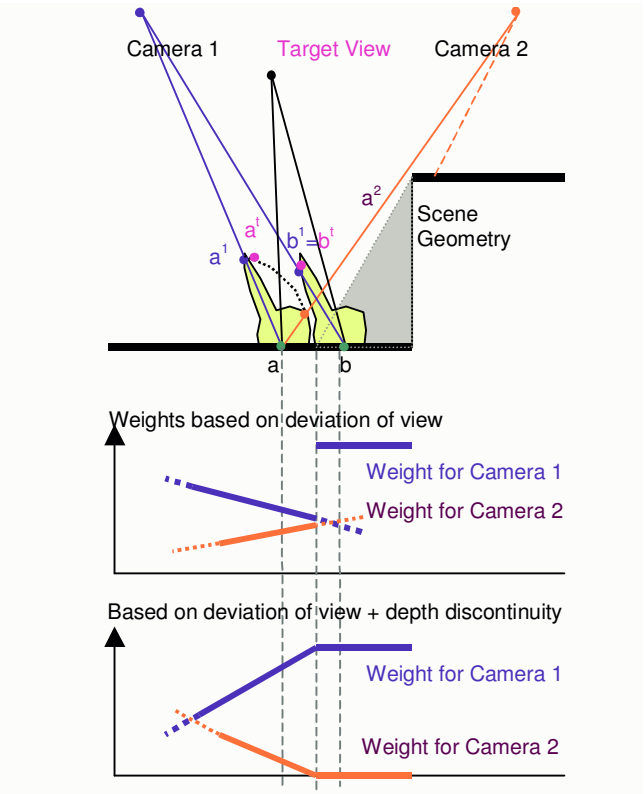


Figure 2. (Top) The weighted combination a^t is very similar to a^1 if feathering across depth discontinuity is considered. (Middle) The traditional feathering based on local information has a gap in the assigned weights. (Bottom) Our method considers depth boundaries to ensure smoothness.

of Figure 1) with respect to Camera 2, a^t is combined as before but $b^t = b^1$ and there is no contribution from Camera 2 at b. Thus nearby points a and b combine very differently. This usually results in an obvious jump in computed intensities in the target view. It is also clear that any computation based on local information cannot detect the variation in the contribution at neighboring points such as a and b.

We propose a global solution, which considers the contribution not just at a fragment but also in the neighborhood of the fragment. As shown in Figure 2, the blending weights near depth discontinuity (in any single source view) are modified to maintain local smoothness. The global processing may not be very critical for reconstruction from very dense set of views, but is essential for view-dependent reconstruction from a sparse set of source views.

For n source images each with m pixels, time complexity of all blending algorithms is $O(n^2 + nm)$, due to the need to find corresponding pixels between all source images and the need to process each source pixel.

4. Computing Blending Fields

Spatial smoothness relates to variation of weights of source images within a target image. Neighboring pixels in target image should have similar weights if there is no depth discontinuity. Temporal smoothness relates to variation of weights of source images at a 3D feature point in the scene in nearby novel views. The weights at a scene feature should change smoothly if the views change smoothly. The guidelines for achieving spatial and temporal smoothness of contributing weights can be stated as follows:

1. **Normalization.** The sum of the intensity weights of the corresponding source image pixels is one so that the intensities in the target image are normalized.
2. **Continuity:**
   Scene smoothness. The weights of a source image along a physical surface should change smoothly in and near overlaps so that the inter-camera intensity differences do not create visible discontinuity in target image.
   Intra-image smoothness. The distribution of intensity weights within a source image should be smooth if there is no depth discontinuity.
3. **Near-view fidelity:** Epipole consistency and minimal angle deviation;
4. **Localization.** Unnecessary blending should be avoided by limiting the number of transition regions to reduce blurring.

The guidelines suggest solving the feathering problem without violating the weight constraints at depth discontinuities and shadow boundaries. It is important to note that, under certain conditions, some of the guidelines may be in conflict and it may not be possible to satisfy all of them. For example, when the boundaries between overlap regions and non-overlap regions meet at a singularity, the blending weights in the local neighborhood of the singularity are not continuous.

4.1. Approach

We use an image-space approach and divide our blending field calculation into two parts: view-independent and view-dependent. The view-independent contributions of the source images are due to the global relationships among the source images and the scene features. They primarily include field-of-view and visibility. One may also include other parameters such as resolution, i.e. sampling density.

View-dependent contribution is due to relationship between the target view and the available sampled source rays. As described in Section 4.4, the final blending field at a scene feature is a normalized dot product of the view-independent and the view-dependent components. The view-independent calculations take into consideration the global visibility constraints while view-dependent calculations are performed per fragment of the target image. A fragment can be an image pixel or a polygon of the geometric proxy.

Typically, in scenes with significant depth discontinuities, view-independent calculations are more complex than view-dependent calculations. For static scenes, the view-independent contribution can be pre-processed. However, the algorithm described below is sufficiently fast to be implemented for dynamic scenes.

4.1.1. View-independent Contributions

Traditional feathering methods use the distance to the nearest invisible or boundary pixel to find the weight [Szeliski97]. In [Debevec98, Buehler01], the contribution due to any particular source view falls to zero as one approaches the boundary of its field-of-view. We, instead use the notion of "distance to the nearest image boundary or depth discontinuity". In this paper, we also refer to depth discontinuity as depth boundary. We first find pixels corresponding to regions seen in only a single source image and assign them intensity weights of 1. Then, for each remaining pixel in the source image, the basic idea behind our technique is to find the shortest path to an image boundary or depth

boundary pixel, *ignoring paths that cross overlap boundaries*. The assigned weight is proportional to this distance. Figure 2 shows the result of this feathering algorithm in flatland for two source images. Even under view-dependent intensities, the algorithm generates smooth transitions corresponding to image of a continuous surface in the presence of shadows and fragmented overlaps. The algorithm can be used for three or more source images without modification.

4.1.2. View-dependent Contributions

The view-dependent component is dependent on the angular deviation of a source ray from the target ray. We use weights based on the traditional approach of using the angular deviations with respect to the *k*-nearest neighbors [Buehler01]. For each target pixel, we choose the corresponding pixels in the *k*-nearest camera views in which the pixel fragment is visible. The follwoing pseudo-code shows how the view-depedent weights are computed:

```
For each target pixel x,
   For each source image i,
      angle_i = deviation of source ray from target ray x
   Find k smallest angles
   For each source image i in k-nearest neighbors,
      w_i(x) = max( 0, (1/angle_i )*(1 – angle_i / angle_k) )
```

We do not need to normalize the view-dependent blending weights, $w_i(x)$, across source images because they will be normalized after the dot product with the view-independent weights. This weighting scheme also ensures the epipole consistency. If the deviation is zero (i.e. the target ray is an epipole) then weight is close to one. The weight gradually drops to zero when the camera drops out of the first *k*-nearest views. $angle_k$ denotes the largest angle among the *k* neighbors.

4.2. View-Independent Weights

Let us consider the view-independent computation in more detail. For a practical real time implementation, we use four buffers for each source image: in addition to the image buffer, $\mathrm{Image}(x)$ and depth buffer $\mathrm{Depth}(x)$, we use a overlap buffer $v(x)$ to record overlaps, a distance buffer $d(x)$ and weight buffer $w(x)$ to store values for pixel x.

A depth buffer is a 2D array of depth values of a source images. If a geometric proxy is used, the depth buffer can be calculated by rendering the geometric proxy from the source camera's viewpoint and reading back the rendered depth buffer. The overlap buffer contains integer values to indicate the number of overlapping camera views for each pixel. The overlap regions (i.e. overlap count of two or more) are computed using the traditional shadow-buffer technique. The following is an outline of the subsequent steps to compute the view-independent weights:

```
For each source image,
   Compute depth discontinuities in depth buffer
   For each pixel in overlap region,
      Update shortest paths to image or depth boundary

For each source image,
   For each pixel in overlap region,
      Find all corresponding pixels in other source images
      Assign weights proportional to the shortest distance
```

The following shows how to find the depth boundaries in a depth buffer.

```
For each depth buffer Depth_i(.)
   For each pixel x with overlap count v_i(x) >= 1
      Mark pixel if depth of any pixel in
      8-neighbor is > ( Depth_i(x)+ threshold )
```

Due to the inequality, the depth boundary pixels are only on the nearer surface and do not result in double boundaries at depth discontinuity (as noticed in traditional edge detectors). The threshold is used to eliminate errors due to lack of depth precision.

The algorithm for weight computation basically creates a smooth function based on the "shortest path to image or depth boundary". In the absence of depth discontinuities, this function will be smooth across the source image. By ignoring paths that cross overlap boundaries, we enforce the *scene smoothness* condition. Hence, the blending weights are discontinuous only at the depth boundaries in image space but are continuous along the scene geometry. During run time, the view-independent weight $d(x)$ is multiplied by the view-dependent blending weight, $w(x)$, which is also smooth. Naturally, the product of these two weighting functions is also smooth except at the depth boundaries [Borgefors86].

The time-consuming part of this weight assignment algorithm is, however, the computation of the distances along the shortest paths, since the paths may not be along straight lines.

4.3. Finding Nearest Nodes

The shortest path problem is well-studied in the field of computational geometry. [Latombe91] suggest many approaches based on generalized Voronoi diagrams. They are commonly used for applications such as path planning in potential field and for avoidance of obstacle nodes. Motion planning is a fundamental problem, with

applications to the animation of digital actors, maintainability studies in virtual prototyping, and robot-assisted medical surgery. The classic Piano Mover's problem involves finding a collision-free path for a robot moving from one location (and orientation) to another in an environment filled with obstacles. The underlying idea is to treat the obstacles as repulsion nodes. The Voronoi boundaries then provide paths of maximal clearance between the obstacles. Due to the practical complexity of computing generalized Voronoi diagrams, the applications of such planners have been limited to environments composed of a few simple obstacles.

Recently, hardware based approaches have been used for greedy path planning algorithms with gradient approach [Hoff99]. They are relatively fast. However, encoding of shortest path for all points with respect to attraction nodes and repulsion nodes is difficult. In our case, attraction nodes are the overlap boundaries and repulsion nodes are the image and depth boundaries.

Instead, we propose an approximation of the distance along shortest path. We try to satisfy the following three goals:

(i) Weights near overlap boundary in overlap region are close to one.
(ii) Weights near depth boundary in overlap region are close to zero.
(iii) Weights for pixels in regions in between transition smoothly.

4.3.1. Voronoi Diagrams Using Graphics Hardware

Our method is based on [Hoff99] to compute Voronoi diagrams using graphics hardware. The idea is to render a cone to approximate each node's distance function. A node can be a pixel on a image boundary or a depth boundary. Each node is assigned a unique color ID, and the corresponding cone is rendered in that color using a parallel projection. When rendering a polygonized cone, the polygon rasterization in the graphics hardware reconstructs all distances across the image plane using linear interpolation of depth across polygons and the Z-buffer depth comparison operation. The Z-buffer depth test compares the new depth value to the previously stored value. If the new value is less, the Z-buffer records the new distance, and the color buffer records the site's ID. In this way, each pixel in the frame buffer will have a color corresponding to the site to which it is closest, and the depth-buffer will have the distance to that site. In order to maintain a single-pixel-accurate Voronoi diagram, a finely tessellated cone needs to be rendered.

Let $Z(x)$ and $z(x)$ denote the nearest depth boundary node and distance to that node, for a pixel x. Note that these Euclidean distances are along straight lines, as defined by Voronoi diagrams. (We convert the depth-buffer values to object-space depth distances, which in this case correspond to distances from boundary pixels in pixel-units). We need to update $z(x)$ to avoid the depth boundary nodes. Let $d(x)$ denote the updated $z(x)$, i.e. the approximate distance that is proportional to the distance to the nearest image or depth boundary.

The pseudo-code for computing the view-independent weights for each source image is as follows:

```
For each pixel j on image and depth boundary,
    Draw a cone with color(j) using parallel projection
Read buffers: Zcolor = color buffer, and Zdepth = depth buffer
For each pixel x,
    d(x) = Zdepth(x)
```

4.4. Rendering Target Image

Let x_i denote the corresponding pixel for a target pixel x in source image i. For a given target view, the following pseudo-code computes the target image using the view-dependent weights and the view-independent weights:

```
For each source image i,
    For each pixel j in image i,
        Splat in target view
        and store d_i(j), view vector and view depth
For each pixel x in target view
        Eliminate invisible contributions
        Find w(x_i) for k-nearest views
        Find normalized weight
            W_i(x_i) = (w_i(x_i) * d_i(x_i)) / Σ_i (w_i(x_i) * d_i(x_i)) for i = 1..k
        TargetImage(x) = Σ_i Image_i(x_i)* W_i(x_i)
```

As mentioned earlier, the procedure can be modified to use any type of fragments. Instead of pixels, one can use polygons from geometric proxy [Buehler01, Debevec98]. The procedure is somewhat simpler than methods where multiple weights per pixel have to be assigned and later distinguished depending on depth values. In our case, to eliminate invisible contributions, before splatting, we simply render the geometric proxy to update the depth buffer in the target view. The summation after dot product on the last line of the algorithm above is achieved with an accumulation buffer in the graphics hardware.

5. Results

Figures 3 and 4 show the results for a simple example. Our objective is to verify our blending method and compare its results with those of traditional local approaches. We have also tested the algorithms with more complex scenes. The multiple source images are blended in real time using alpha blending and an accumulation buffer.

5.1. Issues

For sparse views, the depth values (or proxy geometry) are expected to be sufficiently accurate to avoid aliasing. However, for blending, stricter requirement is needed to ensure that the weights computed using global constraints remain normalized after re-projection. During implementation, we had to handle depth precision and image re-sampling problems and had to use fast morphology (erosion and dilation) to get rid of isolated regions and the resultant artifacts in overlap buffers.

Further analysis is required to understand the trade-off between blurring and limited interpolation of source samples, similar to the sampling issues discussed in [Chai00].

6. Conclusion

We have described a new blending method to influence pixel intensities for merging multiple source views into a target view. We have presented the need to use global constraints for handling feathering across depth discontinuities. In our method, each blending weight is made up of a view-independent and a view-dependent component. We have detailed how the view-independent components can be computed by considering global information such as depth discontinuities and visibility. Using traditional graphics hardware, approximation of the desired view-independent weights can be quickly computed.

The technique presented in this paper is sufficiently general to be used for blending operations in many types of interpolation. They include synthesis from surface light fields, re-lighting of image-based scenes, and novel view generation from sampled BRDF values. We are also investigating adaptive blending techniques for a given image-content such as in [Burt83] and extend them to non-planar scene geometry.

More details and images can be found at
http://www.cs.unc.edu/~raskar/Blending/

7. References

[Borgefors86] G. Borgefors. Distance transformations in digital images. Computer Vision, Graphics and Image Processing, 34:344–371, 1986.

[Buehler01] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen, Unstructured lumigraph rendering, Proc. ACM Conference on Computer Graphics (SIGGRAPH'01), pp. 425–432, Aug. 2001.

[Burt83] Burt, Peter J, and Adelson, Edward H. A Multiresolution Spline With Application to Image Mosaics. ACM Transactions on Graphics 2(4):217-236, October, 1983.

[Chai00] Jin-Xiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum. Plenoptic sampling. SIGGRAPH 00, pages 307–318.

[Debevec96] P. Debevec, C. Taylor, and J. Malik. Modeling and rendering architecture from photographs. SIGGRAPH 96, pages 11–20.

[Debevec98] Paul E. Debevec, Yizhou Yu, and George D. Borshukov. Efficient viewdependent image-based rendering with projective texture-mapping. Eurographics Rendering Workshop 1998.

[Gortler96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. SIGGRAPH 96, pages 43–54.

[Shum99] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. SIGGRAPH 99, pages 299–306.

[Heigl99] B. Heigl, R.Koch, M. Pollefeys, J. Denzler, and L.Van Gool. Plenoptic modeling and rendering from image sequences taken by hand-held camera. Proc. DAGM 99, pages 94–101.

[Hoff99] K. Hoff III, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast Computation of Generalized Voronoi Diagrams using Graphics Hardware. In Computer Graphics (SIGGRAPH '99), pp. 277–286, 1999.

[Latombe91] J.C. Latombe. Robot Motion Planning. Kluwer Academic Publishers, 1991.

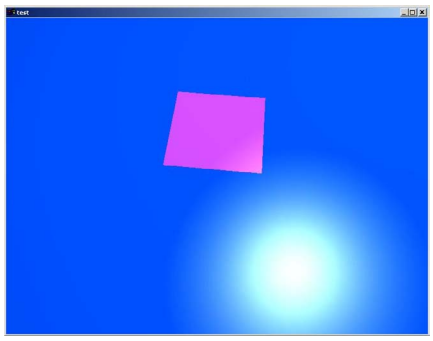
[Levoy96] M. Levoy and P. Hanrahan. Light field rendering. SIGGRAPH 96, pages 31–42.

[Matusik01] Matusik, W., Buehler, C., and McMillan, L., Polyhedral Visual Hulls for Real-Time Rendering, In Proceedings of Eurographics Workshop on Rendering 2001.

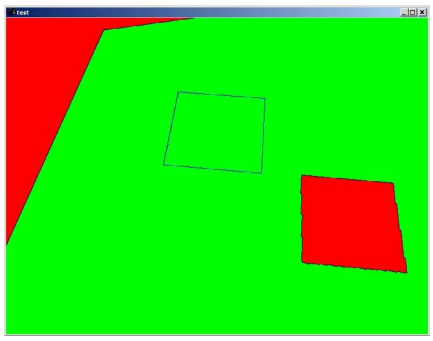
[Pulli97] Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. Eurographics Rendering Workshop 1997, pages 23–34.

[Raskar01] Ramesh Raskar, Greg Welch, Kok-Lim Low and Deepak Bandyopadhyay. Shader lamps : Animating real objects with image-based illumination. In Eurographics Rendering Workshop 2001, August 2001.

[Szeliski97] R. Szeliski and H. Shum. Creating Full View Panoramic Mosaics and Environment Maps. SIGGRAPH ’97, August 1997.

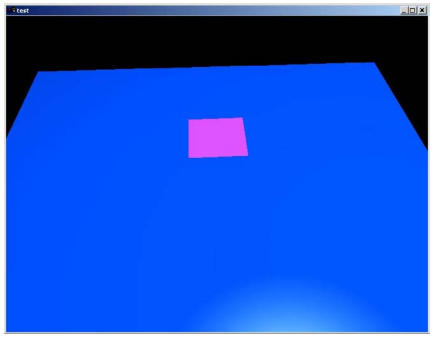


Source Image 1

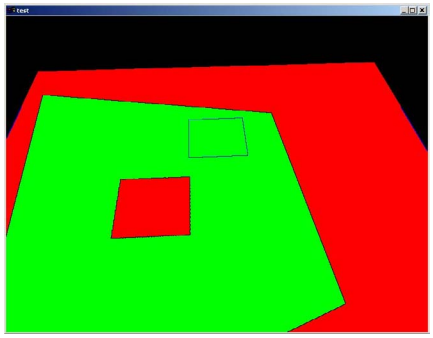


Overlap Map 1

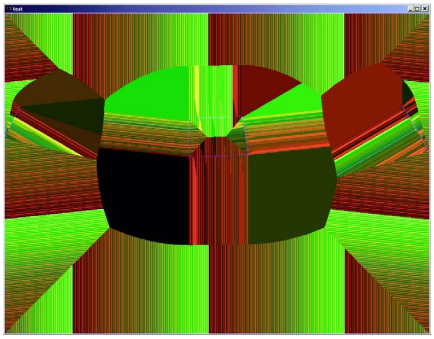
Voronoi Diagram 1



Source Image 2



Overlap Map 2

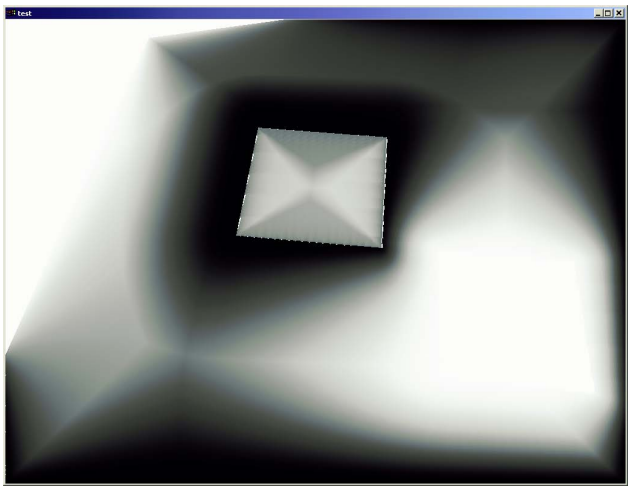


Voronoi Diagram 2

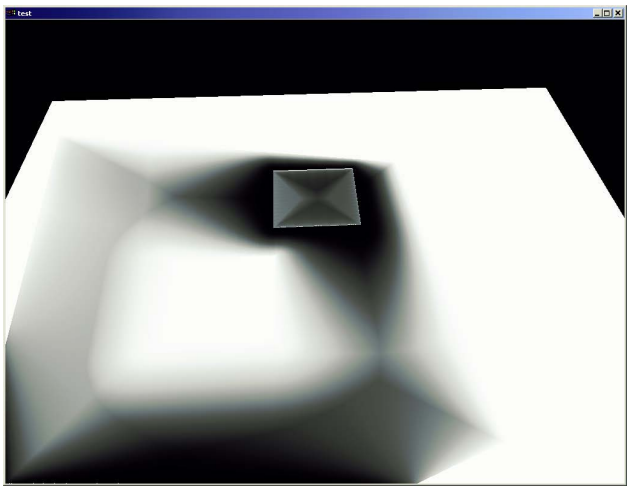
The simple scene consists of a small square floating high above a bigger one. Note the view-dependent specular highlights in the source images.

The overlap map of each source view shows the overlap regions (green), the non-overlap regions (red), overlap boundaries (black) and depth boundaries (blue).

Each pixel in the above voronoi diagrams is colored with the color ID of the nearest node on the image or depth boundaries. The depth boundaries are shown in blue.



View-independent weights for Source View 1.



View-independent weights for Source View 2.

View-independent weights. Note that the weights are one in the non-overlap regions, close to one for pixels near the non-overlap regions and close to zero near depth boundaries.

Figure 3. Results for a simple example.

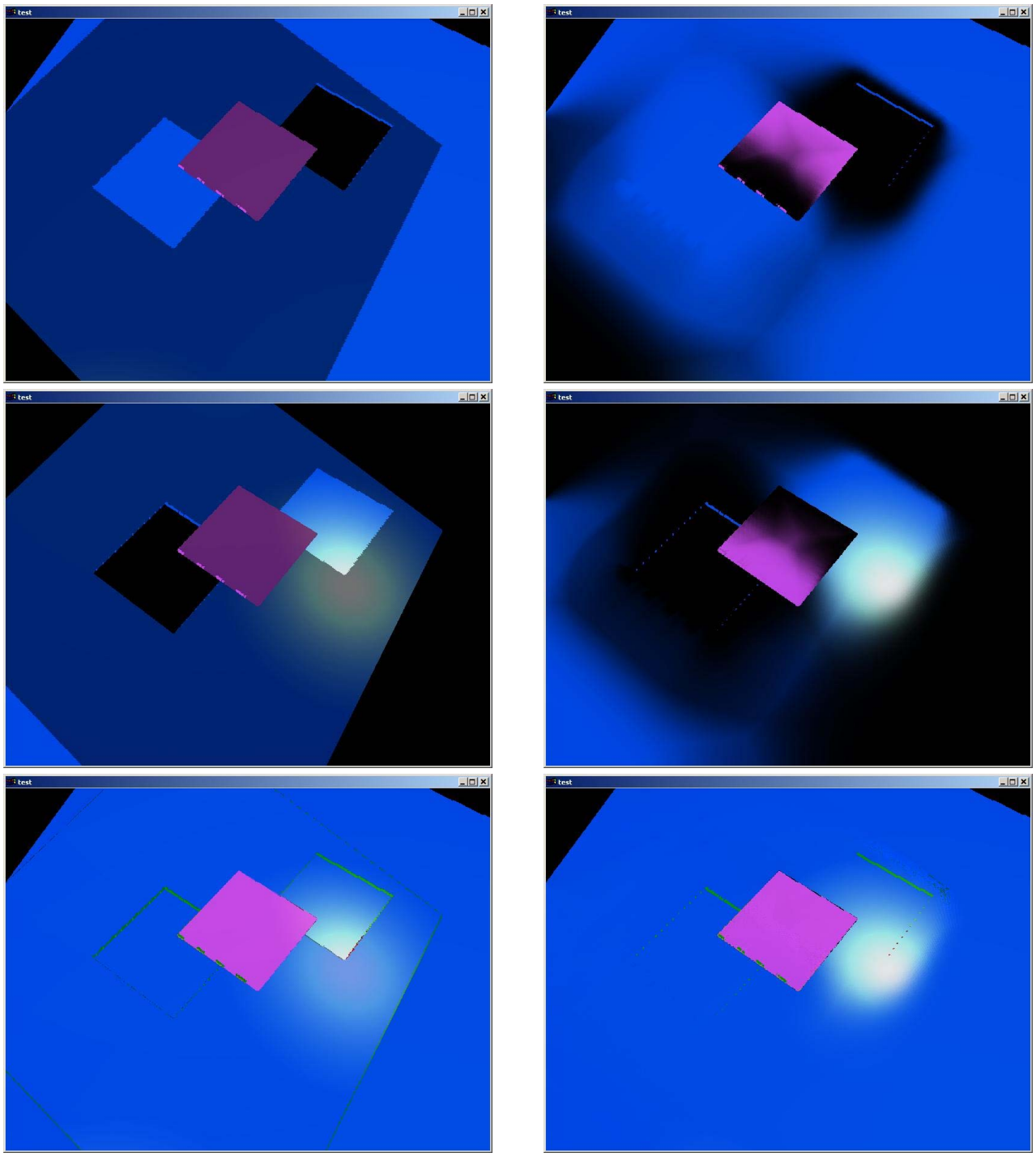


Figure 4. Constructing a target image. (Left column) Using only “deviation from the nearest views” method. (Right column) Our method, which considers depth discontinuities. (Top and middle rows) Contribution from individual source views. (Bottom) Combined sum. Note how in the bottom-left image, the intensities jump near the shadow boundary. In our method (bottom-right image), due to view-independent feathering, the intensities are smooth. (Please ignore the green colored artifacts due to unclamped accumulation buffer.)