Eurographics Symposium on Rendering 2017
P. Sander and M. Zwicker
(Guest Editors)

Volume 36 (2017), Number 4

Minimal Warping: Planning Incremental Novel-view Synthesis

Thomas Leimkühler[1] Hans-Peter Seidel[1] Tobias Ritschel[2]

[1]MPI Informatik [2]University College London

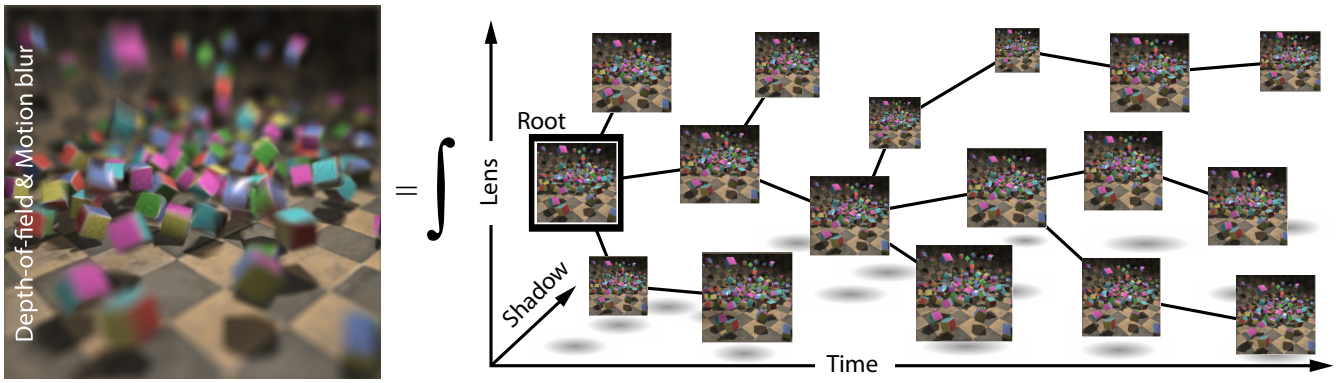


Figure 1: *We produce complex distribution effects such as motion blur and depth-of-field at high quality by warping RGB-D images* (left). *In this example the time, lens, and shadow domain are sampled by warping a reference image* (thick boundary) *into many novel views and averaging them* (left). *Different from previous work which warps the reference into each new sample individually, we organize samples into a tree such that warps* (edges) *become minimal, i. e., they can be produced at high speed by deforming another sample using minimal effort.*

Abstract

Observing that many visual effects (depth-of-field, motion blur, soft shadows, spectral effects) and several sampling modalities (time, stereo or light fields) can be expressed as a sum of many pinhole camera images, we suggest a novel efficient image synthesis framework that exploits coherency among those images. We introduce the notion of "distribution flow" that represents the 2D image deformation in response to changes in the high-dimensional time-, lens-, area light-, spectral-, etc. coordinates. Our approach plans the optimal traversal of the distribution space of all required pinhole images, such that starting from one representative root image, which is incrementally changed (warped) in a minimal fashion, pixels move at most by one pixel, if at all. The incremental warping allows extremely simple warping code, typically requiring half a millisecond on an Nvidia Geforce GTX 980Ti GPU per pinhole image. We show, how the bounded sampling does introduce very little errors in comparison to re-rendering or a common warping-based solution. Our approach allows efficient previews for arbitrary combinations of distribution effects and imaging modalities with little noise and high visual fidelity.

1. Introduction

Rendering images is time-consuming, in particular when complex visual effects such as motion blur or depth of field are needed, for spectral rendering, soft shadow or caustics, chromatic aberrations or when many images need to be produced, such as for light fields or high refresh rates. We observe, that images which such effects or those modalities are the sum of many pinhole images covering a "distribution domain" and that the information across this domain at the same time is highly redundant, presenting an opportunity for our method to exploit it.

One method to exploit this coherence is image warping: after an initial image is shaded, it is warped into several novel ones across the distribution domain and finally averaged. Warping is faster than re-rendering, mostly because geometric transformations and shading costs become decoupled from sampling the distribution effect visibility and is even shared over time or the angular domain. Still, warping itself has two main limitations: First, it can have limited speed in practice, especially when many and very different novel views are required. Second, it can only represent object or camera motion, and ignores other effects such as moving shadows

or caustics or changes of wavelength. Our approach addresses these two shortcomings.

Addressing the first issue, our main observation is that small warps are easier than large warps. In particular, warps by a single pixel can be achieved by an effort not bigger than a conditional move from a tiny pixel neighborhood. Warps that do not change any pixels in a spatial or sampling-domain region are even better: they do not cost any time at all. To this end, instead of performing a large warp from the input image into each new view, we propose a system to plan the traversal of the warping space such that the warps become small. The resulting warping only needs to be done incrementally, i. e., by moving pixels by an amount that is spatially bounded or not at all.

Addressing the second issue, we generalize camera and object motion into a general notion of distribution flow that captures arbitrary reactions of the pixel position inside an image in response to changing any distribution coordinate, including area light sampling position or motion or animated caustics. To this end, we make two contributions: First we introduce shadow and spectral flow, that describe motions of pixels in response to motions of lights, occluders or receivers as well as changes in the spectral band. Second, to represent any arbitrary combination of flows, including shadow and spectral flow, we sample the flow at a low number of representative locations and extrapolate it to all other sample locations using radial basis functions.

The simplicity of our approach allows producing content that fuses arbitrary combinations of distribution effects (lens, time, area lights, spectrum, including shadow or caustic motion) into arbitrary combinations of outputs (plain images, stereo pairs, light fields, higher temporal resolution). In particular, scenes with complex geometry and high shading cost benefit from this decoupling, producing images virtually free of sampling noise. Yet, we show how our approach, when using the sampling bounds we devise, does not introduce more error in comparison to a common warping-based solution while being much faster. At negligible planning cost, our approach is roughly three to four times as fast as drawing points or a forward-warping grid. For a one-megapixel resolution, minimal warping typically requires less than a millisecond, which is only four times slower than the theoretical optimum for any warping algorithm: Reading the forward flow from a texture and directly using it as the inverse flow for a lookup in another texture.

2. Previous Work

Our approach computes distribution effects (Sec. 2.1) by means of a novel multi-image warping procedure (Sec. 2.2). Works related to those two sub-problems are presented next. Furthermore, we review related concepts dealing with incremental or differential changes in images (Sec. 2.3).

2.1. Rendering distribution effects

Our fast warping allows for efficient production of many visual effects, such as depth-of-field, motion blur, temporal up-sampling (generally, known as “distribution effects”) and many others that are otherwise hard to come by. Here we will briefly survey how those distribution effects are commonly produced.

Ray-tracing The canonical way of simulating these is distribution ray-tracing [CPC84]. Here, for every lens, time, etc. sample, a ray is sent. The final image is obtained by averaging the contribution of all rays. This is a general and easily implemented solution, but inherits the difficulties of ray-tracing that does not deliver the same performance as rasterization. Despite its theoretical properties, ray-tracing does not scale favorably with increasing geometry. Also shading complexity directly affects the cost of distribution effects as it is usually not able to exploit the coherence in shading, which is very similar in many cases. However, efficient reuse of samples is possible by considering anisotropic footprints [LAC*11], allowing to reduce the sampling rate significantly.

Micro-polygons An alternative to ray-tracing for high-quality images is REYES [CCC87]. Here, the scene is decomposed into micro-polygons, shaded and rasterized from many views. The shading of vertices decouples it from visibility determination, allowing to exploit coherence. A limitation we share with REYES is to assume shading does not change across distribution effects. While GPU implementations of REYES are available [ZHR*09], the complexity of the process does not map as well to GPUs as plain rasterization does. Rasterizing micro-polygons still requires more computational effort than what is necessary to achieve a very similar result using minimal warping.

Accumulation Buffering Accumulation buffering [HA90] is a simple alternative to this: A GPU is used to render many slightly different pinhole images, which is a highly optimized process taking full advantage of common graphics hardware. The resulting images are accumulated to produce an image with distribution effects. While this is a fairly simple and general method, like every super-sampling it can be slow and the (shading) coherence between similar images is not exploited.

Stochastic rasterization Instead of averaging many pinhole images, an alternative solution is to blend the random nature of distribution ray-tracing and rasterization in what is called stochastic rasterization [AMMH07]. Here, an output image is still rasterized, but contains pixels from many views which are then aggregated to support many distribution effects. Using layered rendering improves the quality here [MVH*14]. Other specialized rasterizations have directly produced multiple views [HAM06]. Finally, direct samples of a light field rendering, i. e., multiple views, can be produced and aggregated using optimization for natural image (or light field) sparsity [HWRH13]. The optimization and the required ray-tracing have not been demonstrated with interactive performance and are limited to the lens domain.

Warping-based Finally, our approach belongs to the category of warping-based solutions. They are similar to accumulation buffering, just that the pinhole images are not actually rasterized, but produced using warping, i. e., by deforming a single RGB-D image. This deformation should account for effects such as parallax and respect occlusions. While the information in a single image is limited, the approach is arguably faster than plain accumulation buffering, as it performs rasterization only once. Warping-based distribution effects have been explicitly demonstrated for depth-of-field [YWY10], but most warping-based work considers dis-

tribution effects as a common test case for the performance they achieve [MMB97, YTS*11, BMS*12, DRE*10].

Many more applications of warping are possible, such as light field pre-filtering [ZMD*06], multi-view expansion [DSAF*13], temporal up-sampling, specular rendering or even spectral rendering [EBR*14].

Specialized solutions While warping is a general solution to all distribution effects and beyond, specialized solutions, such as [LES09] for depth-of-field do perform better. However, it is not clear how those approaches should be combined, and what their resulting performance would be. Also they often make assumptions on motion, scene configurations, sensor shapes etc. that might not apply in general. Our approach does not make any assumptions other than that pixels move in a bounded way in response to sampling the distribution domain.

In particular, solutions based on light transport Fourier analysis [DHS*05] are often limited to special geometric relations, distance assumptions, diffuse surfaces, etc. [BSS*13, YMRD15]. Such limitations are typically not a problem for warping-based approaches that can simulate even the most esoteric effects to be imagined (e. g., temporal up-sampling of a light-field showing a dispersive caustic). In special cases they might be slower, but they share the flexibility with the original distribution ray-tracing of sampling arbitrary domains.

Summary Out of several options to compute distribution effects, warping-based solutions seem to show the best quality and performance trade-off. They are conceptually simple and support all effects that result in image warping using a single code base.

The applicability of warping-based multi-view effects stands and falls with, first, the number of views that can be produced, and second, the speed at which the warping can be performed. Our approach improves speed by designing a warping approach that, for the first time, is tailored towards producing multiple images, instead of previous approaches, that have considered warping of a single source to another one.

2.2. Warping

Deforming images to produce novel views has its origin in interactive exploration frameworks [Lip80], where images undergo simple transformations for more appealing transitions. These basic ideas were further developed in the vision community, where extrapolation from given real images is often required as no mechanism exists to "render" arbitrary novel views [CW93]. Later, the opposite was exploited: what is possible for real images is possible for synthetic ones, too. Mark et al. [MMB97] were the first to present efficient image deformation to speed up rendering, where it is often referred to as "warping".

When used in interactive applications such as games, computational efficiency of warping has become important. Early graphics systems [TK96] have warped entire tiles instead of re-rendering using a linear mapping. If tiles cover non-planar geometry, artifacts arise. Plain drawing of points is neither fast nor does it give high quality. Drawing connected quads is an option, but due to the high number of primitives, does not deliver a high-enough performance required for many warps. To reduce the number of primitives, quad trees were used [DRE*10]. In all cases, using layered depth images [SGHS98] can improve quality [WPS*15].

The best performance can be achieved when warping is expressed as a gathering instead of a scattering operation [McM97, YTS*11, BMS*12]. Here, instead of mapping pixels to a new location along a forward map, the inverse of the map is found and used to decide from which position a pixel in the output image is read. In practice, such approaches can require many iterations and diverge if not initialized properly, such as by using a forward warp.

2.3. Incremental and differential image changes

Capture Epsilon photography [Ras09] refers to techniques in computational photography that rely on multiple images acquired by incrementally varying camera parameters like aperture, exposure, or viewpoint. Observing that there is much redundancy in the acquired images, compressive epsilon photography [ITM*14] only acquires a subset of the images and uses techniques from compressive sensing to infer the missing information. Our approach follows similar ideas as it also utilizes the redundancy of a densely sampled space of images.

Synthesis Ray and path differentials [Ige99, SW01] describe the spatial change of a ray under a differential change of the associated sensor coordinate. This quantity essentially corresponds to the ray's footprint and is therefore an effective means for anti-aliasing. This concept was extended to also include the spectral domain [EBR*14]. The differentials are analytic models for computing the change of ray properties with respect to a limited set of parameters. Our approach is different as it is a non-analytic way of considering screen space flow in respect to a multitude of parameters.

In gradient-domain path tracing [KMA*15] pairs of paths are traced for explicitly estimating image gradients. In conjunction with a Poisson reconstruction step this method yields superior results compared to standard path tracing. While gradient-domain path tracing considers the change of color in respect to pixel position, our approach looks at the change of pixel position in respect to distribution parameters.

3. Overview

We will here give an overview of our approach, starting from the input and output (Sec. 3.1), motivating the need for minimal warping (Sec. 3.2) and giving the big picture of our pipeline (Sec. 3.3).

3.1. Input and output

Input to our approach is a *root image* and a continuous distribution sample domain, e. g., lens, time and area light. Output is a small discrete set (e. g., a stereo pair) of images that are each convex combinations of discrete images in the sample domain.

The *root image* can be any RGB-D image, be it synthetic or acquired, taken from a representative sample, typically in the center of the domain. Highest quality results are obtained by combining our approach with layered depth image (LDI) rendering [SGHS98].

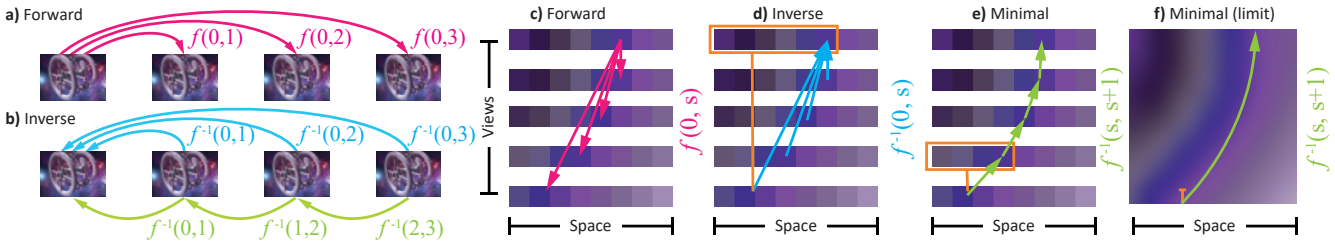


Figure 2: *Forward and inverse warping.* (a) *Starting from the original image at sample* $\mathbf{s} = 0$ *the warp* f *maps to new samples, here* 1, 2 *and* 3 (pink arrow). (b) *Commonly, inverse image warping seeks to create the inverse map* f^{-1} (blue arrow) *from the original image to all other images by inverting the forward warp. We show how the minimal warp* (green arrow) *between a known and a new inverse map is easier and faster to compute. The right part shows a comparison of different strategies for inverting many flows on a 1D image in a 1D sample space. The original image is shown in the top row. The forward flow is show in* (c). *Different samples are on the vertical axis, space is on the horizontal axis. Inverting the flow* (d) *using common approaches requires identifying the correct solution in a set of candidates* (orange box). *This can be accelerated using an iterative search [YTS*11, BMS*12], but in practice compute time grows while quality is decreased when the warp distance grows. Our approach* (e) *only needs to search a small neighborhood to find the candidate. In the limit* (f) *this reduces to a single pixel.*

3.2. Minimal warping

A *forward warp* from a source image to another view defines where every pixel is written to (Fig. 2, a). Such a forward warp is easily applied by drawing points [ZPVBG01, SSLK13], triangles [MMB97] or a quad tree [DRE*10]. Forward warping can produce holes or requires sophisticated filtering and sampling and does not parallelize well.

Our approach instead finds the *inverse warp* [YTS*11, BMS*12], i. e., where every pixel is to be read from (Fig. 2, b). Why is finding an inverse flow hard? It is easy, if the mapping is constant over the image: When all pixels move five pixels to the right in the forward flow, every pixel just needs to look five pixels to the left in an inverse flow. If pixels move differently – and they do if the distribution space coordinates change in an arbitrary scene – the inversion is very hard. When found, however, it is used for sampling the texture and averaging the result per pixel [HA90]. This avoids holes and fits modern GPUs well.

The differences of forward and inverse mappings are best visualized in 2D such as on the right of Fig. 2. The input image (top row) is mapped forward (pink arrows in Fig. 2, c) or backward (blue arrows in Fig. 2, d) to novel views (other rows). Instead of previous work that uses iterative search to find the inverse map between the input image and a new view, we proceed incrementally, allowing to work with minimal warps (Fig. 2, e). In the limit, the search radius becomes a single pixel (Fig. 2, f). For this to work, the approach requires planning.

3.3. Pipeline

Our approach has several steps (blocks in Fig. 3) and two main parts (colors in Fig. 3). The first part (yellow in Fig. 3) estimates distribution flow (Sec. 4) and plans the warping (Sec. 5.1 and Sec. 5.2), the second one executes this plan (Sec. 5.3, blue in Fig. 3). The first part is "intelligent", involving function fitting, graph operations, optimizations, etc. and mostly CPU-based. The second part is very simple and executed on the GPU by no more than massive conditional memcopies.

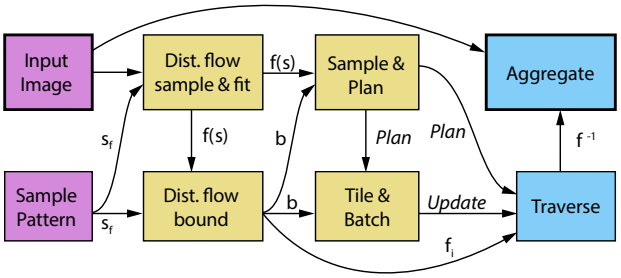
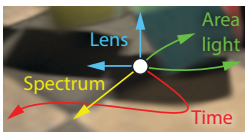


Figure 3: *Overview of our pipeline (see Sec. 3.3).*

Distribution flow model Our approach proceeds in respect to a certain distribution flow model, which defines how a 3D scene position changes as a function of quantities such as, time, lens position, area light coordinate, wavelength, or index in the stereo image pair of light field image array. Before planning the minimal warps, we therefore first need to know how 3D distribution flow responds to changes in the distribution domain.



A simplified version of distribution flow is shown on the right: the arrows show motion of the central pixel in response to changes in the distribution sample domain. The image is a simplification, as we are interested in the *combined* effect of all dimensions, while the image only shows flow with respect to changes in a *single* variable. We opt to sample the domain using a low-discrepancy pattern of pilot samples and fit a radial basis function (RBF) model to the pixel motion.

Output of this step is a model of how each pixel moves in 3D when any of its distribution parameter changes.

Planning The planning phase uses the distribution flow model to sample the distribution domain into images where pixels move at most by a single-pixel distance when moving from one image to its

neighbor. The images are arranged into a *sample tree*, in which an edge is an inverse warp we want to find between two images. The tree is organized such that its traversal will always produce *minimal warps*. We define a warp from image A to image B to be *minimal*, if for every pixel in the output image B, it has not moved more than one pixel from its position in A.

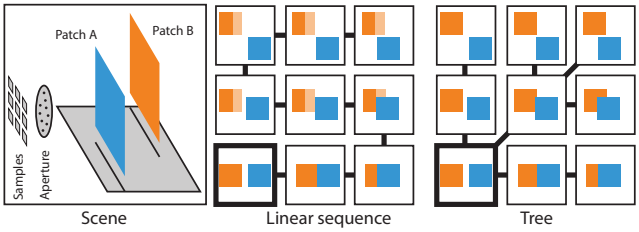


Figure 4: *"Wiping" effects in a linear sequence and a tree.*

An example is Fig. 4, rendering depth-of-field of two patches at different depth, sampled at 9 views for depth-of-field. Consider comparing two alternative sample graphs in the second and third column. The most simple sample graph could be to find the shortest path to connect all samples (Fig. 4, second column). Here, the long overall path can result in disocclusion (transparent in patch B that is behind patch A) propagating over the entire solution (referred to as "Wiping"). Instead, we arrange samples in a tree (Fig. 4, third column). Now, the disocclusions can only affect a single branch, or as in this case, no pixels at all. Intuitively, combining a tree-shaped flow that always expands and never shrinks with small warp lengths creates an efficient solution without occlusion problems. "Wiping" is the only reason, why our solution is not identical to the ground-truth inverse warp, i. e., an exhaustive search for the best matching pixel in the source image.

The ideal traversal would minimize disocclusions directly instead of only minimizing the path length. However, no obvious method exists to predict disocclusions in a complex forward flow in an LDI without executing it.

4. Distribution Flow

We call flow which arises from general distributions and their mutual combinations *distribution flow*, as it is a key component for our solution of the distribution rendering problem [CCC87]. In this section we give details on this notion. First, we formally define the domain and the mapping (Sec. 4.1). Then, we explain how individual flow components are computed (Sec. 4.2). Finally, we show how complex distribution flow is efficiently and compactly represented by using radial basis functions (Sec. 4.3).

4.1. Domain and Mapping

The *sample domain* $\mathbb{S}$ is a subset of the n_d-dimensional hypercube. Typically n_d is 2 or 3, but can also be higher. A *sample* is a position in the sample space and corresponds to an image (Fig. 5). The 3D motion in camera space is defined as a composed mapping $\mathbf{y} = f(\mathbf{s}_1,\mathbf{s}_2)(\mathbf{x}) \in \mathbb{S}\times\mathbb{S} \to (\mathbb{R}^2 \to \mathbb{R}^3)$ from the source and target sample $\mathbf{s}_1$ and $\mathbf{s}_2$ to a mapping of positions $\mathbf{x}$, visible in the original image, to a new 3D camera space position $\mathbf{y}$. The original image sample is denoted as $\mathbf{s} = 0$.

While the above notation describes the flow between two arbitrary samples $\mathbf{s}_1$ and $\mathbf{s}_2$ in distribution space, for the remainder of this section it is best thought of as the warping from one source image at 0 to another image with coordinate $\mathbf{s}$.

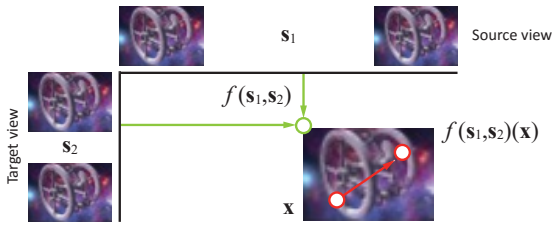


Figure 5: *Common forward flow* (red) *is a mapping from each image location* $\mathbf{x}$ *to a new location* $f(\mathbf{x})$. *We operate on a family of flows* $f(\mathbf{s}_1,\mathbf{s}_2)(\mathbf{x})$ *that depend on the source sample* $\mathbf{s}_1$ *and the target sample* $\mathbf{s}_2$ (green). *In this illustration, the sample space, which is high-dimensional, is depicted two-dimensionally for simplicity.*

4.2. Flow Components

Taking a novel-view sample $\mathbf{s}$ induces a forward flow $f(0,\mathbf{s})$. In the following sections, we describe how to compute specific flow components individually. While object motion flow (Sec. 4.2.1), camera flow (Sec. 4.2.4), and aberration flow (Sec. 4.2.5) correspond to traditional published effects and are therefore only listed to make the paper more self-contained, we give more details on two novel, very specific types of flow: The first one is *shadow flow* (Sec. 4.2.2) that predicts, how an image of a shadow changes when light, occluder or receiver move. The second is *caustic flow* (Sec. 4.2.3) that is applicable to photon-mapped caustics.

The process of flow estimation is best imagined as capturing `gl_Position` of a vertex shader that takes the distribution coordinate $\mathbf{s}$ as parameter. Therefore, no re-rendering is necessary at any point during flow estimation. Furthermore, occlusions are not considered at this stage, since the warping step will take care of visibility configurations. The flow field is stored in a texture at full input image resolution.

In order to combine different flow components, the individual mappings are simply concatenated in light transport order, i. e., in the order they are listed below.

4.2.1. Object motion flow

Moving and deforming geometry causes a scene flow field [VBR*99] that can be used to produce motion blur. In order to obtain the 3D position $\mathbf{x}(t)$ of the surface under a pixel, the rigid or non-rigid mesh deformation is sampled across time.

4.2.2. Shadow flow

Shadow flow predicts the motion of a 3D shadow point $\mathbf{x}$ for a point light source at position $\mathbf{l}$, blocked by an occluder at position $\mathbf{o}$ cast into a receiver position $\mathbf{r}$. To this end, we make the assumption, that

the occluder does not change (the visibility graph does not change, just deforms) and that the receiver is locally planar with normal $\mathbf{n}$. Under this condition, the new position $\mathbf{x}'$ can be computed by ray-casting an "updated" ray from the new light position $\mathbf{l}'$ through the new occluder position $\mathbf{o}'$ to the receiver plane with the new position $\mathbf{r}'$ and new normal $\mathbf{n}'$ (Fig. 6, a). This procedure is simply an analytical ray-plane intersection:

$$\mathbf{x}' = \mathbf{l}' + \mathbf{d}\frac{\langle \mathbf{l}' - \mathbf{r}', \mathbf{n}' \rangle}{\langle \mathbf{d}, \mathbf{n}' \rangle} \quad \text{with} \quad \mathbf{d} = \frac{\mathbf{o}' - \mathbf{l}'}{\|\mathbf{o}' - \mathbf{l}'\|_2}.$$

Since non-silhouette pixels do not have a unique occluder, the mapping from old to new shadow position is well defined only at shadow boundaries (Fig. 6, b). Therefore, we first compute shadow flow at the shadow silhouettes and then densify the resulting sparse flow field using pull-push.

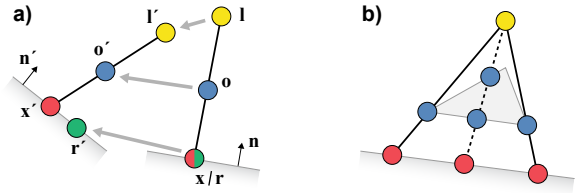


Figure 6: *The geometry of shadow flow.* (a) *A moving light source* $\mathbf{l}$*, occluder* $\mathbf{o}$*, and receiver* $\mathbf{r}$ *lead to a moving shadow position* $\mathbf{x}$*.* (b) *Only shadow silhouettes have a unique occluder.*

Note, that rendering twice is no solution to the shadow flow problem, as it would explain how both shadows look like, but not which point has moved where. Shadow flow is applicable to temporal upsampling of moving lights, occluders or receivers as well as to sampling of area lights or the combination of both.

4.2.3. Caustic flow

Caustic flow predicts where a single caustic $\mathbf{x}$ moves, if the wavelength or the position of the light $\mathbf{l}$, the dispersive interface $\mathbf{i}$, or the receiver $\mathbf{r}$ is changed (Fig. 7). Different from shadows at a single point that are caused by a single occluder, a caustic at a world position is caused by multiple reflectors. We will therefore have to resort to discrete derivatives and assume we use photon mapping to compute caustics: we simply send the same photon for a different light, time and spectral component and reproject it. Note, that in our approach this has to be done for a very low number of distribution pilot samples only, as described in Sec. 4.3. To reconstruct per-pixel flow, we use density estimation of distribution flows i. e., every photon does not splat its color, but the change of position, relative to the distribution coordinate, e. g., time.

Since this produces noisy estimates, we robustify it by, first, setting flow with a magnitude larger than a threshold to zero, and second, by performing median filtering. The flow is finally densified using pull-push.

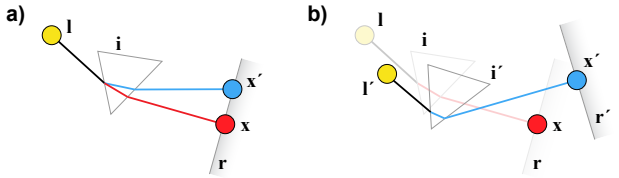


Figure 7: *The geometry of caustic flow.* (a) *A change of wavelength leads to caustic motion.* (b) *This can be combined with a moving light source* $\mathbf{l}$*, dispersive interface* $\mathbf{i}$*, and receiver* $\mathbf{r}$*. As in the case of shadow flow, dashed symbols denote the new configuration.*

4.2.4. Camera flow

Camera flow produces motion blur induced by camera motion as well as depth of field from a thin lens. The 3D object position, which may be altered by the previously described flows, is projected using a lens-time-sampled camera matrix $A(u,v,t)$ [HA90].

4.2.5. Aberration flow

For transverse chromatic aberrations (Fig. 13) a typical forward flow is given by the second-order radial lens distortion model

$$f(0,\mathbf{s})(\mathbf{x}) = \begin{pmatrix} \mathbf{x}_x \left(1 + \lambda_\mathbf{s}\left(\alpha r + \beta r^2\right)\right) \\ \mathbf{x}_y \left(1 + \lambda_\mathbf{s}\left(\alpha r + \beta r^2\right)\right) \\ \text{depth}(\mathbf{x}) \end{pmatrix},$$

where $\lambda_\mathbf{s}$ denotes the wavelength of sample $\mathbf{s}$, $r = \sqrt{\mathbf{x}_x^2 + \mathbf{x}_y^2}$ denotes the distance from the image space position $\mathbf{x}$ to the image center, and α, β are free parameters to control the amount and shape of the radial distortion.

4.3. Representing Distribution Flow

The distribution flow is potentially high-dimensional and partly expensive to evaluate, making it impractical to sample for any cubature-type approach. In contrast, we will perform a cubature-like sampling to produce individual view samples in Sec. 5.1. This is feasible, since the amount of work needed for creating a view sample via minimal warping is in the order of magnitude of a texture copy operation.

To find the distribution flow for every pixel, we assume it to be smooth across the sample domain $\mathbb{S}$. Note, that this does not assume spatial smoothness of either the image itself or the flow field or the inverse flow field, which need to tackle occlusions.

Inspired by quasi-Monte Carlo rendering we create a low number K of pilot samples $\mathbf{s}_f$ with low discrepancy (Halton pattern with leaping [RA99]) and sample the flow at these pilot locations, avoiding the curse of dimensionality and still covering the domain well. We include an additional deterministic sample at $\mathbf{s} = 0$ to ensure that the original image will be interpolated exactly. We did not see evidence that more than $K = 12$ samples provide a change in the scenes we explored. Next, we fit a radial basis function (RBF) model

$$f(0,\mathbf{s})(\mathbf{x}) = \sum_i^K \mathbf{w}_i(\mathbf{x})\phi(\|\mathbf{s} - \mathbf{s}_{f,i}\|_2) \tag{1}$$

with $\mathbf{s}_{f,i}$ being the i'th pilot sample. We use the inverse multiquadric radial basis function $\phi(r) = \left(r^2 + a^2\right)^{-\frac{1}{2}}$ with scale factor $a = 2n_d$. The weights $\mathbf{w}_i(\mathbf{x}) \in \mathbb{R}^3$ are determined by solving the linear system

$$\sum_i^K \mathbf{w}_i(\mathbf{x})\phi(\|\mathbf{s}_{f,i} - \mathbf{s}_{f,j}\|_2) = f(0, \mathbf{s}_{f,j})(\mathbf{x}), \quad \text{for } 1 \leq j \leq K.$$

This is done by first performing a QR decomposition of the $K \times K$ system matrix (which is the same for all pixels) and with this then cheaply and robustly solving the system for each pixel in parallel.

After this step, the flow from the root to any sample can be approximated by K RBF evaluations and a dot product.

5. Minimal Warping

The central idea of this work, is to efficiently find the inverse flow $f^{-1}(0, \mathbf{s}_3)$ from the original image to many $\mathbf{s}_3$ by recursively re-using solutions $f^{-1}(\mathbf{s}_1, \mathbf{s}_2)$ computed previously. The inverse flow is not the inversion of the entire mapping. Instead, it can only explain where a 2D position in the novel images was in the original image.

We will now explain the four main steps of our algorithm in detail. Planning (Sec. 5.1) produces a set of samples and a traversal order that allows for minimal warps. Tiling and Batching (Sec. 5.2) determines which image regions for which samples can be skipped without introducing errors. Warping (Sec. 5.3) performs the flow inversion, and Aggregation (Sec. 5.4) produces the final output.

5.1. Sample planning

Input to the planning is the sample domain $\mathbb{S}$ and an RBF distribution flow. Output is a discrete set of samples S and a view traversal order. Planning proceeds in seven steps (Fig. 8): Bounding distribution flow *(i)* and sampling *(ii)* produce a set of samples with a carefully determined density that allows for minimal warps. A bounded distance graph *(iii)* in conjunction with a shortest-path tree algorithm *(iv)* and connection sampling *(v)* augment the samples with a connectivity structure. Tree linearization *(vi)* produces the final traversal order. Optionally, a traversal stack *(vii)* can be created to significantly reduce the memory requirements of the Warping step.

Bounding distribution flow In order for minimal warping to work we need to know how a change of a sample domain coordinate translates into pixel motion in image space. More specifically, we want to know how many samples are needed per sampling dimension so that f moves at most p pixels in the image when moving from one sample to the next. We denote this quantity $\mathbf{b} \in \mathbb{R}^{n_\mathrm{d}}$. The total number of samples is then $n_\mathrm{s} = \prod \mathbf{b}_i$. A usual value for p is 1 or 2 pixels.

For determining a conservative estimate of $\mathbf{b}$ we are interested in the maximum rate of change of f with respect to each sampling dimension. Therefore,

$$\mathbf{b}_i = \frac{1}{p} \max_{\mathbf{x}} \max_{\mathbf{s} \in \mathbb{S}} \left\| \begin{pmatrix} \frac{\partial f^{(1)}(0,\mathbf{s})}{\partial \mathbf{s}^{(i)}}(\mathbf{x}) \\ \frac{\partial f^{(2)}(0,\mathbf{s})}{\partial \mathbf{s}^{(i)}}(\mathbf{x}) \end{pmatrix} \right\|_2 \tag{2}$$

where $\mathbf{s}^{(i)}$ denotes the i'th component of $\mathbf{s}$ and $f^{(1)}$ and $f^{(2)}$ are the horizontal and vertical component of the flow, respectively. The depth component of f is not necessary at this point, because it does not give any information about image space motion. The maximum over $\mathbf{s}$ is found using gradient ascent, starting at every pilot sample $\mathbf{s}_f$, in parallel for all pixels. The maximum over all pixels $\mathbf{x}$ is found using a max MIP map.

Sampling Sampling (Fig. 8, a) considers the entire continuous domain $\mathbb{S}$ and produces a discrete sample set $S = \{\mathbf{s}_i\}$. To this end, the sample domain is discretized into a grid of size $\mathbf{1}/\mathbf{b}$ (where $/$ denotes element-wise division) and one sample is produced per grid cell. Following the above, samples are placed exactly such that pixels move only p pixel distances when going from one sample to the next for the entire sampling domain. Additionally, every sample is subject to correlated multi-jittering [Ken13]. Our approach naturally lends to such a sampling pattern as it exhaustively covers the domain in all dimensions by design to not miss any feature.

Bounded distance graph Key to our planning is to arrange the samples into a graph (Fig. 8, b). In this graph, the samples $\mathbf{s}_i$ are vertices. Edges are created between two vertices if their sample space distance corresponds to less than p pixels in image space. The distance between samples i and j is measured according to a $\mathbf{b}$-weighted norm that converts the Euclidean sample distance into units of pixel motion, so $d_{ij} = ||\mathsf{B}(\mathbf{s}_i - \mathbf{s}_j)||_2$, where B is a diagonal matrix scaling dimension k by $\mathbf{b}_k$. The graph is computed in parallel for all $n_s \times n_s$ candidate edges und is very sparse with an average out-degree of 1.8.

Shortest path tree Next, the graph is converted into a shortest-path tree, with the input sample $\mathbf{s} = 0$ as a root, using a breadth-first search in linear time (Fig. 8, c). For each node during search we select the child being explored next at random, which leads to a smaller traversal stack (explained below).

Connection Sampling Occasionally, due to the jittered sampling pattern, tree edges correspond to a pixel motion slightly larger than p. Therefore, we simply add a sample in the middle of each such edge (Fig. 8, d).

Tree linearization Tree linearization turns the tree into a sequence of edges (indices of parent and child) as encountered in a depth-first, pre-order traversal (Fig. 8, e). Depth-first warping is preferred over breadth-first warping that would also allow for minimal warping, but requires to store a much larger number of intermediate inverse warps when using a traversal stack as explained below. No particular order is necessary among children. However, to optimize progressiveness, optionally, when more than two children are present, we employ a farthest-first traversal of the children. As the representative position of a child we choose the mean position of all nodes in the child's sub-tree. The farthest-first traversal avoids exploring similar sub-trees sequentially and encourages the exploration of diverse solutions first. The end-outcome is not affected by this operation, only when executing parts of S, e. g., for progressive preview.

Traversal stack While the procedure described above is sufficient to execute a minimal warping plan, it still requires to have all previously visited nodes in memory as they can appear as the parent

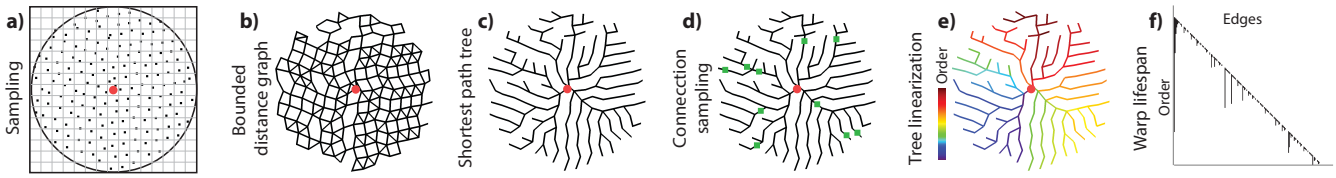


Figure 8: *Overview of six steps for sample planning, here for a two-dimensional lens sample space example.*

of any child. If we are interested in creating distribution effects, there is no need to store all warped images. Instead, we can perform a moving average. In this step, which is optional, the traversal is augmented with additional information, such that only a small stack of active inverse warps needs to be remembered when executing the minimal warping plan. Therefore, the space complexity of our approach depends on the depth of the traversal tree and not on the total number of its vertices.

A warped sample needs to be remembered only, if it will be needed as a source sample for a minimal warp at a later point in the traversal (Fig. 8, f). Whenever a new sample $\mathbf{s}_i$ has been created from a source (i. e., parent) sample $\mathbf{s}_{i-1}$, two configurations may occur: *a)* If $\mathbf{s}_i$ is the last child of $\mathbf{s}_{i-1}$ in the traversal order, $\mathbf{s}_{i-1}$ can be forgotten. *b)* If $\mathbf{s}_i$ is not a leaf, $\mathbf{s}_i$ needs to be stored as a source sample for one or more later warps. Since the sample ordering originates from a pre-ordered, depth-first traversal of a tree, a simple stack data structure is sufficient to encode this behavior. After the last occurrence of a sample $\mathbf{s}_{i-1}$ as the source for a minimal warp, the sample can be popped from the stack. When a newly created sample $\mathbf{s}_i$ will be needed as a source for a later warp, it is pushed to the stack. Following this procedure, the source sample for the current minimal warp will always be the top of the stack while executing the traversal plan.

5.2. Tiling & Batching

Depending on the distribution domain and its associated flow, different image regions move at different speeds. We exploit this fact by subdividing the image into tiles (for execution coherence) and determining a distinct update pattern for each tile.

When using the max MIP map for solving Equation 2, we can simply read the MIP map at a lower level to get a vector $\mathbf{b}^{(i)}$ per tile. A typical tile size is 16×16 pixels. The values of $\mathbf{b}^{(i)}$ cannot be used as a representative sampling density measure for the tiles, since they are only estimating forward flow. A tile in an inverse warping framework can only be skipped, if nothing maps to it. In order to get an estimate of the inverse flow bounds $\widetilde{\mathbf{b}}^{(i)}$, we first compute a per-pixel axis-aligned bounding box (AABB) of the flow (Fig. 9, a). The four sides of the AABB are determined as

$$\text{left/right} : \min_{\mathbf{s}\in\mathbb{S}} / \max_{\mathbf{s}\in\mathbb{S}} f^{(1)}(0,\mathbf{s})(\mathbf{x}),$$
$$\text{bottom/top} : \min_{\mathbf{s}\in\mathbb{S}} / \max_{\mathbf{s}\in\mathbb{S}} f^{(2)}(0,\mathbf{s})(\mathbf{x}),$$

using gradient descent and ascent, respectively, in parallel for all pixels. Then, we intersect each tile with the AABBs of all other tiles and compute, in parallel for all tiles, $\widetilde{\mathbf{b}}^{(i)} = \max_{j\in\Omega_i} \mathbf{b}^{(j)}$, where Ω_i is the set of all indices corresponding to tiles whose AABB intersect tile i. The maximum is performed component-wise. Tiles with $\widetilde{\mathbf{b}}^{(i)} = \mathbf{0}$ are not affected by the distribution sampling. Consequently, they do not need to be processed at all. As a side effect, this achieves scalability in regard to scenes with high depth complexity by avoiding spending to much work on almost-empty LDI layers.

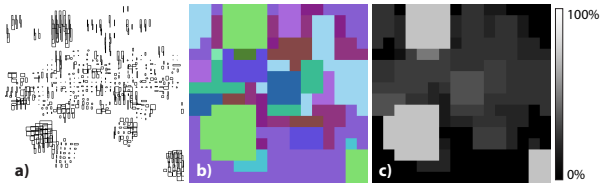


Figure 9: *Tiling and batching is done by computing axis-aligned bounding boxes of the flow* (a) *to perform clustering of tiles* (b)*. In* (c) *the update frequency for individual tiles during sample space traversal is shown. In this motion blur example most tiles need an update only at every third sample or less often.*

Since we cannot afford to compute and store an update pattern for each tile, all remaining active tiles are clustered by their $\widetilde{\mathbf{b}}$ vectors (Fig. 9, b). We use k-means clustering using scattering and blending [DGR*09] ($k = 20$ in all our experiments). To produce the final update pattern per cluster (Fig. 9, c), we iterate over all samples $\mathbf{s}$ for all tiles in parallel: If in any sampling dimension the distance between the current sample and the sample at which the cluster had its last update is larger than the cluster's value $\mathbf{1}/\widetilde{\mathbf{b}}$, the cluster needs an update at the current sample's parent in the traversal tree. Once the update pattern is created, all that needs to be done for each tile during the sample space traversal is to look up if it needs an update based on its cluster ID. During aggregation, this update pattern will be taken into account.

5.3. Warping

The warp itself is executed for every sample in two steps: Requesting (Sec. 5.3.1) evaluates the forward flow and computes necessary auxiliary information. Searching (Sec. 5.3.2) inverts this forward flow based on a previously warped sample.

5.3.1. Requesting

Requesting computes $f(0,\mathbf{s}_i)$, the camera-space 3D forward flow from the original image to sample $\mathbf{s}_i$ of every pixel seen in the original image. This is done by evaluating Equation 1.

From the forward flow field we additionally compute a magnification field $m(0,\mathbf{s}_i)(\mathbf{x})$ at sample $\mathbf{s}_i$. This field contains the larger absolute eigenvalue of the Jacobian

$$J(0,\mathbf{s}_i)(\mathbf{x}) = \begin{pmatrix} \frac{\partial f^{(1)}}{\partial \mathbf{x}^{(1)}} & \frac{\partial f^{(1)}}{\partial \mathbf{x}^{(2)}} \\ \frac{\partial f^{(2)}}{\partial \mathbf{x}^{(1)}} & \frac{\partial f^{(2)}}{\partial \mathbf{x}^{(2)}} \end{pmatrix} (0,\mathbf{s}_i)(\mathbf{x}),$$

where, again, $\cdot^{(1)}$ and $\cdot^{(2)}$ denote horizontal and vertical components, respectively. The entries of the Jacobian are computed numerically from neighboring pixels using central differences. A magnification field value of 1 indicates that in image space, locally, the forward flow merely induces translation or rotation. In a minification or magnification condition, this value is smaller or larger than 1, respectively. The magnification field is later used to appropriately deal with those two conditions. Note that neither condition will increase the number of elements required to search for flow inversion, but will only adapt the thresholds for deciding which candidate is better than a different one.

5.3.2. Searching

Next, we explain how the plan can be executed by performing minimal warps. A minimal warp is essentially a small search. As we will explain, the search can be performed *directly* or in an *occlusion-aware* fashion. Finally we will show, how, while we work on an integer nearest neighbor field (NNF), an optional sub-pixel accurate search allows to read the input image at fractional coordinates.

Direct search The direct search is illustrated in Fig. 10, b. Searching will produce the inverse warp of the current sample $f^{-1}(0,\mathbf{s}_i)$ from the inverse warp of the previous sample $f^{-1}(0,\mathbf{s}_{i-1})$. The planning step has assured that for every $\mathbf{x}$, the solution $f^{-1}(0,\mathbf{s}_i)(\mathbf{x})$ is in the set $\{f^{-1}(0,\mathbf{s}_{i-1})(\mathbf{x} \oplus [-p,\dots,p]^2)\}$, where $\oplus$ denotes Minkowski summation i. e., the set of all points in a two-dimensional integer lattice of side length $2p+1$ around that point. Intuitively, this says that the correct solution for the current sample is known to be a very nearby other pixel in a previous sample. Therefore, for every pixel in the output, all that is required is a two-dimensional loop over its $(2p+1)^2$ neighboring pixels.

Input : Inverse flow $f^{-1}(0,\mathbf{s}_{i-1})$ at previous sample, Forward flow $f(0,\mathbf{s}_i)$ for new sample.
Output : Inverse flow $f^{-1}(0,\mathbf{s}_i)$ and depth at new sample.
forall *pixels* $\mathbf{x}$ *in* $f^{-1}(0,\mathbf{s}_i)$ **do**
  result($\mathbf{x}$) := vec3(0, 0, ∞);
  vec2 $\mathbf{u}$:= $f^{-1}(0,\mathbf{s}_{i-1})(\mathbf{x})$
  forall $\mathbf{y} \in [-p,\dots,p]^2$ **do**
    vec3 $\mathbf{t}$:= $f(0,\mathbf{s_i})(\mathbf{u}+\mathbf{y})$;
    if $\|\mathbf{x}_{xy} - \mathbf{t}_{xy}\|_\infty < m(0,\mathbf{s}_i)(\mathbf{x})$ *and* $\mathbf{t}_z \le \text{result}_z(\mathbf{x})$ **then**
      result($\mathbf{x}$) := vec3($\mathbf{u}+\mathbf{y}$, $\mathbf{t}_z$);
    end
  end
end
return result;

Algorithm 1: Direct search.

This can be implemented in a very simple and shader-friendly way as seen in Alg. 1. The outer loop is parallel over all pixels. The small inner loop is performed by a fragment shader. It merely requires visiting the neighboring pixels and performing two tests: If the forward flow maps to the current pixel, and if yes, if it is closer to the camera than any other pixel that previously also mapped there. The winner is the new inverse flow. The acceptance criterion for the decision whether a pixel maps to the current one depends on the minification or magnification m. In a magnification condition ($m > 1$), the forward flow is allowed to map to a location further away from the pixel center to be accepted, avoiding holes in the warped image. In a minification condition ($m < 1$), the tighter threshold leads to a more accurate selection of the best match.

Input : Inverse flow $f^{-1}(0,\mathbf{s}_{i-1})$ at previous sample, Forward flow $f(0,\mathbf{s}_i)$ for new sample.
Output : Inverse flow $f^{-1}(0,\mathbf{s}_i)$ and depth at new sample.
forall *pixels* $\mathbf{x}$ *in* $f^{-1}(0,\mathbf{s}_i)$ **do**
  result($\mathbf{x}$) := vec3(0, 0, ∞);
  vec3 $\mathbf{u}$:= vec3(0, 0, ∞);
  forall $\mathbf{y} \in [-p,\dots,p]^2$ **do**
    vec2 $\mathbf{r}$:= $f^{-1}(0,\mathbf{s}_{i-1})(\mathbf{x}+\mathbf{y})$;
    vec3 $\mathbf{q}$:= $f(0,\mathbf{s_i})(\mathbf{r})$;
    if $\|\mathbf{x}_{xy} - \mathbf{q}_{xy}\|_\infty < m(0,\mathbf{s}_i)(\mathbf{x})$ *and* $\mathbf{q}_z \le \mathbf{p}_z$ **then**
      $\mathbf{u}$:= vec3($\mathbf{r}$, $\mathbf{q}_z$);
    end
  end
  forall $\mathbf{y} \in [-1,0,1]^2$ **do**
    vec3 $\mathbf{t}$:= $f(0,\mathbf{s_i})(\mathbf{u}_{xy}+\mathbf{y})$;
    if $\|\mathbf{x}_{xy} - \mathbf{t}_{xy}\|_\infty < m(0,\mathbf{s}_i)(\mathbf{x})$ *and* $\mathbf{t}_z \le \text{result}_z(\mathbf{x})$ **then**
      result($\mathbf{x}$) := vec3($\mathbf{u}_{xy}+\mathbf{y}$, $\mathbf{t}_z$);
    end
  end
end
return result;

Algorithm 2: Occlusion-aware search. The gray part is identical to the non-occlusion-aware variant.

Occlusion-aware search The above direct search procedure fails, if the target pixel was occluded in the source sample. In this situation, the inverse flow from the previous sample is an inaccurate initial guess of the current inverse flow. But since also occlusion boundaries can only have moved within $[-p,\dots,p]^2$, the problem can be solved by an additional small search in $f^{-1}(0,\mathbf{s}_{i-1})(\mathbf{x})$ for the best initial estimate, as seen in Fig. 10, c and defined in Alg. 2. After this search, the algorithm proceeds as in the non-occlusion-aware variant. All results in this paper were produced using occlusion-aware search.

Sub-pixels accuracy The above procedure searches a low number of *discrete* choices. The continuous forward flow, however, maps every discrete pixel in the initial sample to a continuous sub-pixel location. Sub-pixel accurate inverse flow is found by fetching the four pixels around the discrete optimum and computing the continuous location $\mathbf{y}$ that minimizes the cost of pyramid matching [Bai03]. In practice, we never remember the continuous sub-pixel coordinate in floating point precision, but a discrete NNF with integer coordinates to avoid the accumulation of floating point errors.

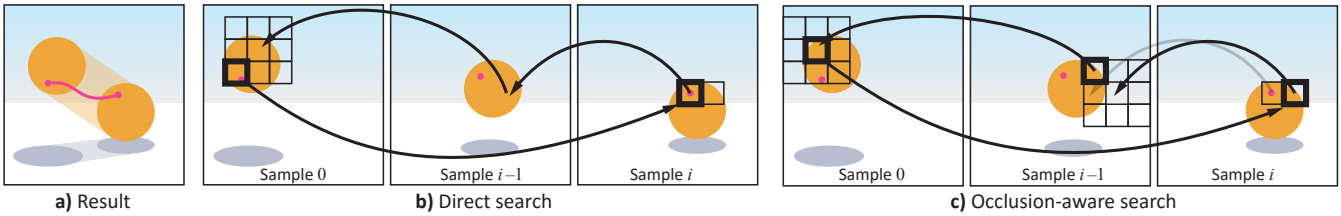


Figure 10: *Details of our searching procedure. Consider a yellow ball with a pink dot casting a shadow moving from left to right and rotating to produce motion blur* (a). *Our method inverts the flow from sample $i-1$ to sample i. For the direct search (Alg. 1), in sample i, for one pixel* (thick box)*, the inverse flow in sample $i-1$ is looked up. Using this result, a neighborhood $[-p,\dots,p]^2$ is searched in sample* 0*, i. e., the original image, to find the pixel that best maps to the current sample* (again, thick box)*. This procedure works well, except at occlusions, which require an occlusion-aware search (Alg. 2) instead* (c)*. Here the pixel of interest in sample i* (thick box) *is an occlusion, i. e., the same pixel in sample $i-1$ belongs to the background. The solution is to perform two searches: The first one finds the best position in sample $i-1$, the second one proceeds as in direct search. Note, that in the case of simple 2D motion like in this example, the second search is not necessary, because the first search finds the best position immediately. In case of perspective 3D motion, the second search prevents error accumulation.*

5.3.3. Sorting

When using LDIs, the sparsity naturally occurring in the back layers can be used to prevent wiping even more. On a per-pixel basis, values are simply pushed back as far as possible in the layer ordering, as illustrated in Fig. 11. This is done once per input LDI. All LDI results in this paper were produced using sorting.

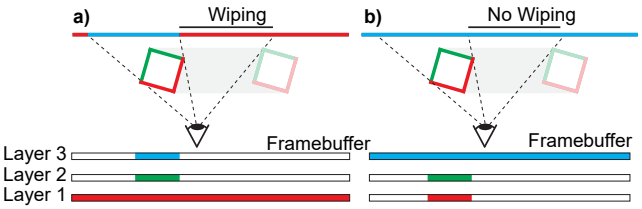


Figure 11: *Per-pixel sorting of layers prevents wiping, here illustrated for a box moving in front of a planar background.*

5.3.4. Implementation

Requesting and searching are implemented in parallel for all pixels in all tiles. Our prototype is implemented in the OpenGL shading language. The plan is stored in a vertex buffer object and executed by layered rendering, where the samples correspond to layers. A geometry shader emits quads for all tiles that require updating. Synchronization has to be performed, such that all tiles in one layer are finished before a new layer can start.

5.4. Aggregation

Once the inverse flow for a view sample has been found, it can be used to perform a simple texture lookup of the input image to create the associated novel view. After a sample has been produced, it can contribute to a single, or multiple output images. Let n_o denote the number of output images. In the simplest case, the output is a single image, and all samples are averaged. The other extreme is temporal up-sampling: here multiple samples are produced and each one is a result image. In general, multiple samples can contribute to multiple output images. In general, the aggregation of n_s samples into n_o images can be described as an *aggregation matrix* denoted as $A \in \mathbb{R}^{n_s \times n_o}$. Fig. 12 shows a visualization of A. The aggregation is performed point-wise for all output modalities. When not using LDIs as input to the system, disocclusions occur naturally. During aggregation, these undefined regions are simply ignored.

6. Results and Discussion

Here we present qualitative (Sec. 6.1) as well as quantitative (Sec. 6.2) results of our approach, and discuss design choices (Sec. 6.3) and limitations (Sec. 6.4).

6.1. Qualitative Results

Here we show several applications of planning minimal warping.

Temporal up-sampling Besides distribution effects, a typical application of warping is temporal up-sampling (frame rate conversion) [DER*10, BMS*12]. This is challenging in the presence of complex geometry and motion. In Fig. 12 we show a sequence of several moving characters, undergoing complex accelerating local motion between two key-frames (left and rightmost image).

Stereo-to-light field A typical application requiring many warps is conversion of stereo content to light fields, in particular, when correct inter-view filtering is required to avoid aliasing [ZMD*06]. In Fig. 14, depth is created by finding correspondences from the stereo pair, which is then used to create many virtual views that are carefully combined using a custom aggregation matrix to avoid inter-view aliasing.

Depth-enabled photo effects Finally, we demonstrate the quality and time of adding several effects to acquired RGB-D data [SHKF12] in Fig. 15 using our approach.

6.2. Quantitative Results

Our approach is compared to different competitors in terms of computing different effects. As competitors, we consider *i)* point splatting, *ii)* grid warping with a cell size of one pixel [MMB97], *iii)*

Figure 12: *Temporal up-sampling from two key-frames to in-between images with a* 360° *degree tent shutter. Insets show pairs of our interpolation* (top) *and the rendered reference* (bottom)*. The aggregation matrix* $\mathsf{A} \in \mathbb{R}^{256\times 3}$ *is shown right (large weights are darker).*

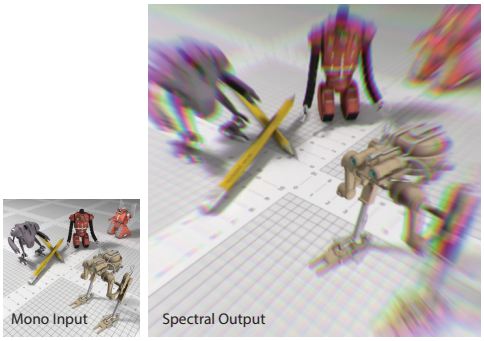


Figure 13: *Chromatic aberrations produced by our method.*

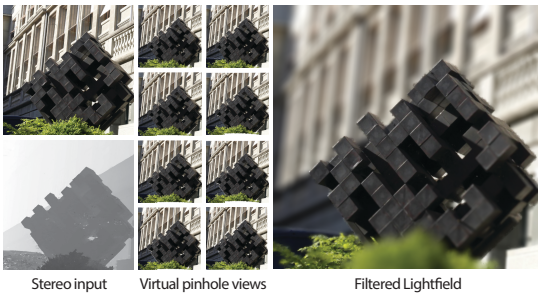


Figure 14: *Conversion of a stereo image* (left) *into a light field* (right) *– here showing a single new view. Our approach creates many virtual views* (center) *that are carefully combined to provide inter-view anti-aliasing [ZMD*06].*

quad tree-warping [DRE*10], and *iv)* a ray-traced reference. We did not consider iterative gathering methods in this comparison, because the complex flow fields with wide baselines lead to non-convex energies to minimize, requiring these methods to be initialized, i. e., with a quad tree [BMS*12]. This shows, that any converging search method can in principle not be faster than our approach which is already 2 to 3 times faster than quad tree-warping. All competitors had the same LDIs as input.

The effects tested are *i)* depth-of-field, *ii)* motion blur, *iii)* soft shadows, *iv)* spectral caustics, and *v)* combinations of the above. Fig. 16 shows images produced from our approach as well as insets made from our approach and the ray-tracing reference. The results were produced on a Nvidia Geforce GTX 980Ti with an Intel Xeon E5-1607 CPU at a resolution of 1024×1024 pixels and are summarized in Table 1. The given timings include all steps described in the paper, except for the "Prism" scene, where the spectral photon tracing (Sec. 4.2.3) was done using an external application. We performed same-quality comparisons, as can be seen from the almost identical SSIM values. We observe that our approach can produce same-quality images in significantly less time than common approaches using warping. Additionally, our novel shadow and spectral flow formulations allow our approach to produce a more versatile range of distribution effects.

In Table 2 we give timings for the individual processing stages of our method. It can be seen that the planning stage usually constitutes only a small fraction of the total amount of work. It is furthermore noticeable that the timing for the warping is almost in the same order of magnitude as the aggregation step i. e., summing the novel views.

Table 3 shows equal-time and equal-quality comparisons of our approach to a path tracing solution created with Renderman 20.10. It can be seen that path tracing, which has to perform shading evaluation for each ray, produces results of significantly less quality in the same time and needs considerably longer to compute equal-quality results. This is true even if the time to shade a pinhole image i. e., one sample per pixel, which is the input to our system (timings are given in the *Shading* column of Table 2), is taken into account.

6.3. Discussion

Our cubature employs a regular sampling of the distribution domain. In theory, the approach therefore needs time exponential in the number of distribution dimensions n_d. In practice, the volume of all distribution flows (i. e., $n_s = \prod \mathbf{b}_i$) effectively constitutes the amount of work to be done. Consequently, it makes no difference if a combined lens-time sample space requires $10\times10\times10$ samples, or if motion blur alone smears a single pixel across 1000 pixels. We share this inherently strong dependency on the magnitude of the distribution effects to be produced with other rendering methods, including Monte Carlo approaches. Producing a novel view via minimal warping requires nothing more than two texture lookups in a 3×3 neighborhood per pixel. This is in contrast to producing a sample via ray-tracing, which exhibits larger constants and scales with geometric scene complexity, making cubature integration infeasible.

6.4. Limitations

Our main limitation, shared with typical assumptions in production, is to assume shading can be decoupled from resolving visibility. In several occasions such as small but strong highlights or shadows in

Figure 15: *Results of our method applied to acquired RGB-D data from the NYU dataset [SHKF12].* a) *and* b)*: Depth of field, 256 samples.* c)*: Motion blur from camera motion, 64 samples.* d)*: Motion blur from camera motion, 128 samples.* e)*: Original images.*

Table 1: *Efficiency and visual quality of different methods (columns) on different scenes (rows). Factor is time relative to our solution (smaller is better). Similarity is measured in SSIM [WBSS04] relative to a ray-tracing reference (larger is better). Distribution effects marked with an asterix cannot or can only partially be reproduced by the competitors.*

Scene	Effect	Point [ZPVBG01]			Grid [MMB97]			Quad-tree [DRE*10]			Ours	
		Time	Fac.	Sim.	Time	Fac.	Sim.	Time	Fac.	Sim.	Time	Sim.
Dancing Dummies	MB	4.1 s	×4.2	.93	2.5 s	×2.6	.93	2.1 s	×2.1	.92	**1.0 s**	.92
Garden	DOF	84.8 s	×2.1	.91	99.5 s	×2.4	.90	104.9 s	×2.6	.88	**41.2 s**	.90
Bouncing Cubes	MB + Shadow MB*	35.9 s	×4.8	.88	24.3 s	×3.2	.86	15.2 s	×2.0	.85	**7.5 s**	.86
Vehicle	Soft Shadows*	-	-	-	-	-	-	-	-	-	**40.0 s**	.96
Space Station	MB + DOF	1350.1 s	×3.5	.93	1132.5 s	×2.9	.92	1035.0 s	×2.7	.91	**387.7 s**	.93
Prism	Spectral Caustics*	-	-	-	-	-	-	-	-	-	**4.2 s**	-

the presence of motion blur, this is violated and can become noticeable. The consequences of this decoupling have been analyzed and discussed elsewhere [CCC87]. Another limitation is anti-aliasing: in our approach, every image pixel is associated with a single visibility sample and a unique depth value (modulo the layers of the LDI). Anti-aliasing can be applied by later super-sampling, but this remains costly as it increases both memory requirements and compute time.

While the limitations given above apply to all warping-based approaches, wiping is an error source unique to our approach. For flow configurations with complex visibility relations this results in image regions disappearing after occlusions for single branches of the sample tree. Wiping is rare in practice, not only because of the sample tree structure, but also because different geometry is placed on different LDI layers, which in turn are purposefully sorted to reduce the problem.

Our approach has the highest throughput if many very similar views are required. If the distribution space is not sampled densely (i. e., $p \gg 1$), searching becomes inefficient. Therefore, warping an image into another single image is better done using other methods.

7. Conclusion

We have presented the first approach to frame the problem of producing novel views by looking at the entire family of flows instead of all warps individually. Inspired by recent success in solving inverse problems that find the flow given the image [ZTS*16], we have shown that given the forward flow organized in a tree of flows, the inverse warping can become as simple as looking up a small pixel neighborhood for each pixel.

Our approach relies on LDIs to resolve disocclusions. An avenue for future work is to consider alternative input formats like per-pixel lists or a point cloud. Furthermore, our approach could be extended for rendering global illumination effects beyond caustics.

We have shown, how planning minimal warping provides a fast and flexible alternative to ray-tracing or multi-sampling using accumulation buffering, yet with enough flexibility to support all combinations of distribution effects such as motion blur, depth-of-field, soft shadows, and spectral shading. In particular, upcoming output devices such as high-refresh rate displays found in head-mounted displays or light field displays, require a massive amount of pixels in space and time. Yet, those images are redundant and almost identical. Minimal warping is the first approach to exploit this redundancy, resulting in flexible, efficient and high-quality imagery.

Acknowledgements

This work was partly supported by the Fraunhofer and the Max Planck cooperation program within the framework of the German pact for research and innovation (PFI).

References

[AMMH07] AKENINE-MÖLLER T., MUNKBERG J., HASSELGREN J.: Stochastic rasterization using time-continuous triangles. In *Graphics Hardware* (2007), pp. 7–16. 2

[Bai03] BAILEY D. G.: Sub-pixel estimation of local extrema. In *Proc. Image and Vision Computing* (2003), pp. 414–19. 9

[BMS*12] BOWLES H., MITCHELL K., SUMNER R. W., MOORE J., GROSS M.: Iterative image warping. *Computer Graphics Forum 31*, 2pt1 (2012), 237–246. 3, 4, 10, 11

Table 2: *Timings for the individual stages. Two numbers in the* Layers *column denote the number of non-shadow and shadow layers. The timing of flow estimation for* Prism *does not include the spectral photon tracing, for which an external application was used. Timings given in italics are shown for completeness and are independent of our approach.*

Scene	Layers	n_d	n_s	Shading	Flow est.	Planning	Tiling	Warping	Aggreg.
Dancing Dummies	3	1	191	*115 s*	0.01 s	0.06 s	0.08 s	0.58 s	0.27 s
Garden	4	2	5385	*168 s*	0.05 s	0.91 s	0.09 s	30.96 s	9.15 s
Bouncing Cubes	5+5	1	848	*150 s*	0.05 s	1.15 s	0.09 s	4.89 s	1.36 s
Vehicle	2	2	11230	*95 s*	0.01 s	0.38 s	0.09 s	27.40 s	12.13 s
Space Station	5	3	53864	*426 s*	0.04 s	6.67 s	0.13 s	275.78 s	105.03 s
Prism	1	1	1059	*103 s*	0.01 s	0.12 s	0.08 s	3.26 s	0.74 s

Table 3: *Equal-time and equal-quality comparison to a path tracer.*

Scene	Equal Time (SSIM)	Equal Quality (Time)
Dancing Dummies	.70	3300 s
Garden	.38	4800 s
Bouncing Cubes	.80	480 s
Space Station	.89	1400 s

[BSS*13] BELCOUR L., SOLER C., SUBR K., HOLZSCHUCH N., DURAND F.: 5D covariance tracing for efficient defocus and motion blur. *ACM Trans. Graph (Proc. SIGGRAPH) 32*, 3 (2013), 31. 3

[CCC87] COOK R. L., CARPENTER L., CATMULL E.: The Reyes image rendering architecture. *SIGGRAPH Comp. Graph. 21*, 4 (1987), 95–102. 2, 5, 12

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. *SIGGRAPH Comput. Graph. 18*, 3 (Jan. 1984), 137–45. 2

[CW93] CHEN S. E., WILLIAMS L.: View interpolation for image synthesis. In *SIGGRAPH* (1993), pp. 279–88. 3

[DER*10] DIDYK P., EISEMANN E., RITSCHEL T., MYSZKOWSKI K., SEIDEL H.-P.: Perceptually-motivated real-time temporal upsampling of 3D content for high-refresh-rate displays. *Comp. Graph. Forum (Proc. Eurographics) 29*, 2 (2010), 713–22. 10

[DGR*09] DONG Z., GROSCH T., RITSCHEL T., KAUTZ J., SEIDEL H.-P.: Real-time indirect illumination with clustered visibility. In *VMV* (2009), pp. 187–196. 8

[DHS*05] DURAND F., HOLZSCHUCH N., SOLER C., CHAN E., SILLION F. X.: A frequency analysis of light transport. *ACM Trans. Graph. (Proc. SIGGRAPH) 24*, 3 (2005), 1115–26. 3

[DRE*10] DIDYK P., RITSCHEL T., EISEMANN E., MYSZKOWSKI K., SEIDEL H.-P.: Adaptive image-space stereo view synthesis. In *Proc. VMV* (2010), pp. 299–306. 3, 4, 11, 12

[DSAF*13] DIDYK P., SITTHI-AMORN P., FREEMAN W. T., DURAND F., MATUSIK W.: Joint view expansion and filtering for automultiscopic 3d displays. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 32*, 6 (2013). 3

[EBR*14] ELEK O., BAUSZAT P., RITSCHEL T., MAGNOR M., SEIDEL H.-P.: Spectral ray differentials. *Proc. EGSR 33*, 4 (2014). 3

[HA90] HAEBERLI P., AKELEY K.: The accumulation buffer: hardware support for high-quality rendering. *SIGGRAPH Comp. Graph. 24*, 4 (1990), 309–18. 2, 4, 6

[HAM06] HASSELGREN J., AKENINE-MÖLLER T.: An efficient multi-view rasterization architecture. In *Proc. EGSR* (2006), pp. 61–72. 2

[HWRH13] HEIDE F., WETZSTEIN G., RASKAR R., HEIDRICH W.: Adaptive Image Synthesis for Compressive Displays. *ACM Trans. Graph. (Proc. SIGGRAPH) 32*, 4 (2013), 1–11. 2

[Ige99] IGEHY H.: Tracing ray differentials. In *Proc. SIGGRAPH* (1999), ACM, pp. 179–186. 3

[ITM*14] ITO A., TAMBE S., MITRA K., SANKARANARAYANAN A. C., VEERARAGHAVAN A.: Compressive epsilon photography for post-capture control in digital imaging. *ACM Trans. Graph. 33*, 4 (2014), 88. 3

[Ken13] KENSLER A.: *Correlated multi-jittered sampling*. Tech. rep., Pixar Technical Memo, 2013. 7

[KMA*15] KETTUNEN M., MANZI M., AITTALA M., LEHTINEN J., DURAND F., ZWICKER M.: Gradient-domain path tracing. *ACM Trans. Graph. 34*, 4 (2015), 123. 3

[LAC*11] LEHTINEN J., AILA T., CHEN J., LAINE S., DURAND F.: Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. 30*, 4 (2011). 2

[LES09] LEE S., EISEMANN E., SEIDEL H.-P.: Depth-of-field rendering with multiview synthesis. In *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* (2009), vol. 28, ACM, p. 134. 3

[Lip80] LIPPMAN A.: Movie-maps: An application of the optical videodisc to computer graphics. In *ACM SIGGRAPH* (1980), vol. 14, pp. 32–42. 3

[McM97] MCMILLAN L.: *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, University of North Carolina at Chapel Hill, 1997. 3

[MMB97] MARK W. R., MCMILLAN L., BISHOP G.: Post-rendering 3D warping. In *Proc. i3D* (1997). 3, 4, 10, 12

[MVH*14] MUNKBERG J., VAIDYANATHAN K., HASSELGREN J., CLARBERG P., AKENINE-MÖLLER T.: Layered reconstruction for defocus and motion blur. *Comp. Graph. Forum 33*, 4 (2014), 81–92. 2

[RA99] ROBINSON D., ATCITTY C.: Comparison of quasi-and pseudo-Monte Carlo sampling for reliability and uncertainty analysis. In *Proc. AIAA Probabilistic Methods* (1999). 6

[Ras09] RASKAR R.: Computational photography: Epsilon to coded photography. *Emerging Trends in Visual Computing* (2009), 238–253. 3

[SGHS98] SHADE J., GORTLER S., HE L.-W., SZELISKI R.: Layered depth images. In *Proc. SIGGRAPH* (1998), pp. 231–42. 3

[SHKF12] SILBERMAN N., HOIEM D., KOHLI P., FERGUS R.: Indoor segmentation and support inference from RGBD images. In *ECCV* (2012). 10, 12

[SSLK13] SIBBING D., SATTLER T., LEIBE B., KOBBELT L.: SIFT-realistic rendering. In *Proc. 3DV* (2013). 4

[SW01] SUYKENS F., WILLEMS Y. D.: Path differentials and applications. In *Proc. EGSR* (2001), Springer, pp. 257–268. 3

[TK96] TORBORG J., KAJIYA J. T.: Talisman: Commodity realtime 3D graphics for the PC. In *Proc. SIGGRAPH* (1996), pp. 353–63. 3

[VBR*99] VEDULA S., BAKER S., RANDER P., COLLINS R., KANADE T.: Three-dimensional scene flow. In *ICCV* (1999), vol. 2, pp. 722–729. 5

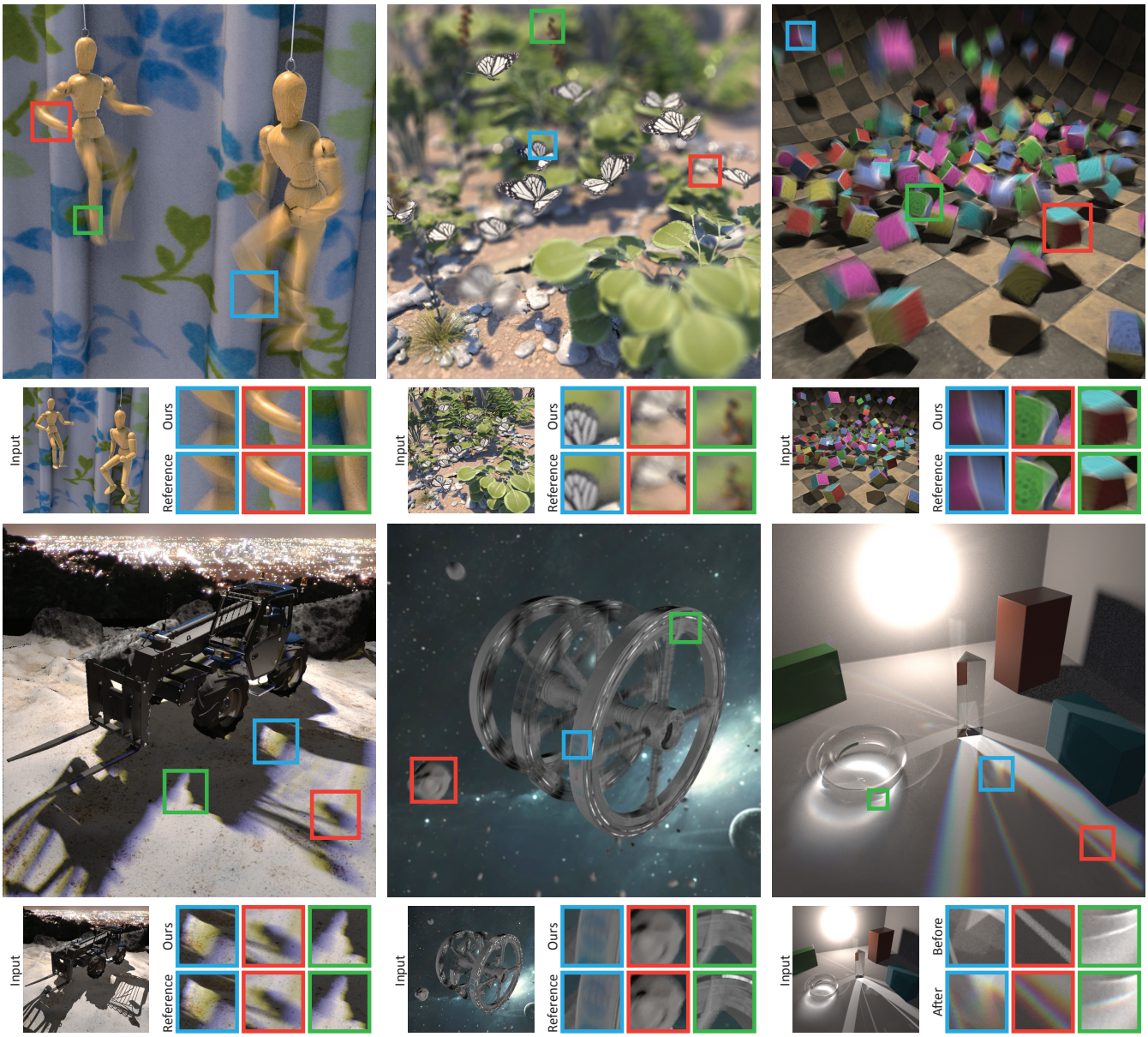


Figure 16: *Results of our method. The large figures show the end-result. Insets show comparisons between a multi-pass reference and ours.*

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: from error visibility to structural similarity. *IEE Trans. Image Proc. 13*, 4 (2004), 600–612. 12

[WPS*15] WIDMER S., PAJAK D., SCHULZ A., PULLI K., KAUTZ J., GOESELE M., LUEBKE D.: An adaptive acceleration structure for screen-space ray tracing. In *Proc. HPG* (2015), pp. 67–76. 3

[YMRD15] YAN L.-Q., MEHTA S. U., RAMAMOORTHI R., DURAND F.: Fast 4D sheared filtering for interactive rendering of distribution effects. *ACM Trans. Graph. 35*, 1 (2015), 7. 3

[YTS*11] YANG L., TSE Y.-C., SANDER P. V., LAWRENCE J., NEHAB D., HOPPE H., WILKINS C. L.: Image-based bidirectional scene reprojection. *ACM Trans. Graph. 30*, 6 (2011), 150:1–150:10. 3, 4

[YWY10] YU X., WANG R., YU J.: Real-time depth of field rendering via dynamic light field generation and filtering. In *Comp. Graph. Forum* (2010), vol. 29, pp. 2099–107. 2

[ZHR*09] ZHOU K., HOU Q., REN Z., GONG M., SUN X., GUO B.: Renderants: interactive REYES rendering on GPUs. In *ACM Trans. Graph. (Proc. SIGGRAPH)* (2009), vol. 28, p. 155. 2

[ZMD*06] ZWICKER M., MATUSIK W., DURAND F., PFISTER H., FORLINES C.: Antialiasing for automultiscopic 3d displays. In *SIGGRAPH Sketches* (2006). 3, 10, 11

[ZPVBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proc. SIGGRAPH* (2001), pp. 371–8. 4, 12

[ZTS*16] ZHOU T., TULSIANI S., SUN W., MALIK J., EFROS A. A.: View synthesis by appearance flow. In *Proc. ECCV* (2016). 12