# Stereo Compositing Accelerated by Quadtree Structures in Piecewise Linear and Curvilinear Spaces

Dmitriy Pinskiy Joseph Longson Peter Kristof Evan Goldberg Robert Neuman

Walt Disney Animation Studios

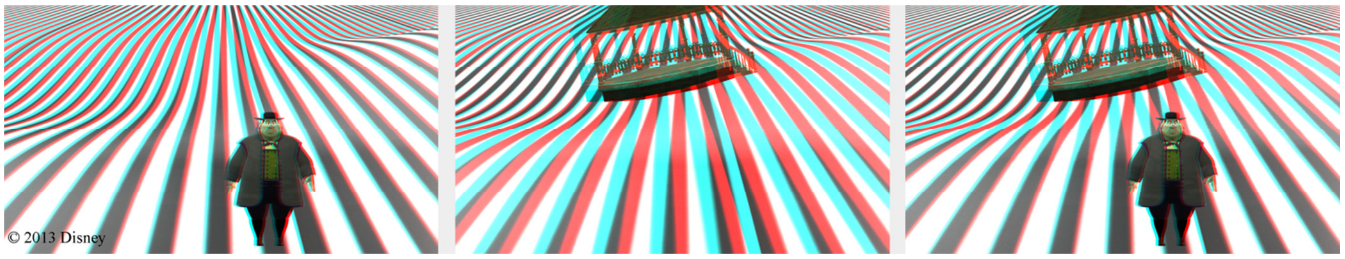


**Figure 1:** *Our compositing scheme, even when applied to stereo CG renderings with drastically different stereo settings (e.g. the left and middle renderings), still produces smooth transition from foreground to background regions as shown on the right.*

## Abstract

We present a new stereoscopic compositing technique that combines volumetric output from several stereo camera rigs. Unlike previous multi-rigging techniques, our approach does not require objects rendered with different stereo parameters to be clearly separable to prevent visual discontinuities. We accomplished that by casting not straight rays (aligned with a single viewing direction) but curved rays, and that results in a smooth blend between viewing parameters of the stereo rigs in the user-defined transition area. Our technique offers two alternative methods for defining shapes of the cast rays. The first method avoids depth distortion in the transition area by guaranteeing monotonic behavior of the stereoscopic disparity function while the second one provides a user with artistic control over the influence of each rig in the transition area. To ensure practical usability, we efficiently solve key performance issues in the ray-casting (e.g. locating cell-ray intersection and traversing rays within a cell) with a highly parallelizable quadtree-based spatial data structure, constructed in the parameterized curvilinear space, to match the shape definition of the cast rays.

**CR Categories:** I.3.3 [Picture/Image Generation]: Line and curve generation, Viewing algorithms, I.3.4 [Graphics Utilities]: Graphics editors, I.4.8 [Scene Analysis]: Stereo, I.4.10 [Image Representation]: Volumetric.

**Keywords:** stereo, line and curve generation, viewing algorithms.

## 1 Introduction

Recent popularity of stereoscopic rendering in CG animation has led to increasing complexity of stereoscopic 3D films. Complex animation shots should not be restricted to a single stereo setup. We desire the ability to assign different stereo depths to different groups of objects in order to allow greater artistic flexibility and control of the stereoscopic 3D effect. This is generally accomplished with multi-rigging - rendering from different pairs of stereo cameras separately and then compositing the output together. The muti-rigging technique has been heavily employed in many feature films (e.g. "Meet the Robinsons" and "Beowulf") [Mendiburu 09]. However, multi-rigging, as Mendiburu describes, has an important artistic limitation - it requires objects rendered with different stereo parameters to be clearly separatable with only empty space between them. If this requirement is not met, the output results in visual discontinuities. This tremendously limits the usefulness of multi-rigging since in many cases there are large common objects (e.g. secondary props) filling the space between the main sets of objects. The one of the most common objects that might be visible in several stereo rigs and consequently prevent from applying multi-rigging composition is a ground plane as shown in Figure 2.

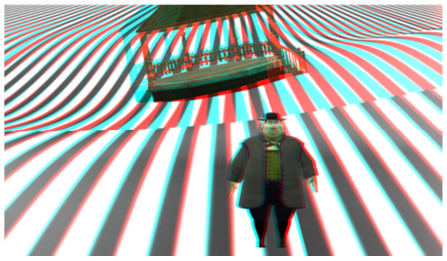


**Figure 2:** *Traditional multi-rigging in results in visual discontinuities unless composited objects are separable.*

The goal of our method is to provide artistic freedom and eliminate the above restriction and thus increase the scope of multi-rigging in virtually any CG scene. It is accomplished by utilizing a type of volumetric data known as deep images and then casting rays, curved to guarantee the visual continuity between the stereo pairs. To ensure practical value, we also present a highly parallelizable, quadtree-based spatial data structure. The performance efficiency of this spatial data structure during ray-casting comes from its construction, which is done in the parameterized curvilinear space defined by the casting-rays' shape definition. By doing this, we effectively solve key performance issues of ray-casting, e.g. locating cell-ray intersection and traversing rays within a cell.

The principal contributions of our paper are: (1) removal of the empty space restriction to ensure greater artistic freedom in multi-rigging, (2) a compositing scheme that uses ray-casting to smoothly connect multiple stereo cameras in a multi-rig setup and (3) an optimized quadtree that delivers up to two times speed-up for CG shots.

## 2 Background and Related Work

One of the key challenges that we face in compositing the output of multiple stereo rigs is producing a depth-coherent look. As we progress from one region to another, we need the ability to prescribe appropriate disparities in order to seamlessly bridge stereoscopic settings of the neighboring composited regions. Although this particular problem has not been addressed in the context of multi-rigging, disparity manipulation has been researched in a number of papers in various other contexts. Given a desired disparity range for the scene, Jones et al. compute the left-right camera separation for the optimum scene depth [2001], and Holliman calculates the stereoscopic camera parameters for the scene to obtain needed disparity in a given region of interest [2004]. Both works can be considered as scene planning approaches. In contrast, we post-process the output of the stereo-rigs and respect users' stereo depth choices in the composited regions, allowing depth manipulation only in the overlapping regions to ensure seamless transition. Depth manipulation in post-processing has been offered in a number of other approaches. For example, Koppal et al. describe a set of post-processing tools to achieve the desired disparity for the 2D images [2011] and Lang et al. provide a more advanced algorithm for smoothly warping 2D images [2010]. Unfortunately, 2D image manipulation introduces distortion, and, as a result, some objects can appear unnaturally stretched and compressed. In addition, approaches that are based on 2D image warping suffer from problems due to hole-filling issues and the lack of proper handling of semi-transparency since the data behind opaque objects is not available. To avoid these problems, Kim et al. compute and modify disparity information based on volumetric input, using light fields [2011]. However, in a production environment, light fields can be more costly to produce than deep images, which are easily generated with production quality rendering packages such as Pixar's Renderman.

Deep images have been primarily used in the rendering of high quality shadowing [Lokovic and Veach 2000]. However, now they are also commonly employed in final compositing, using software packages such as Nuke and Houdini. Sample-based deep images represent a rendering produced by a camera with the viewing direction aligned with z-axis and the image plane aligned with XY plane. A deep image is given as an $N$ x $M$ array of deep pixels $P_{n,m}$ (see Figure 3). Each deep pixel is a collection of samples, sorted by their depth:

$$P_{n,m} = \{(c_0, a_0, z_0), (c_1, a_1, z_1), \dots\}; \;\; z_0 < z_1 < \dots < z_n$$

where $c_i$, $a_i$, $z_i$ ($i \in N$) represent respectively a sample's color, opacity, and z-depth components along the ray emitted from the pixel with image coordinates ($n$, $m$). A complete set of all pixels from the same height $m$ forms a slice $S_m = \{P_{0,m}, P_{1,m}, \dots, P_{n,m}\}$. We define a deep image function $\boldsymbol{I}$ as a map from a deep pixel and bounding z depth range, [$z_{start}$, $z_{end}$], to the subset of samples that contribute to this deep pixel:

$$\boldsymbol{I}(n, m, z_{start}, z_{end}) = \{(c, a, z) \in P_{n,m} \mid z_{start} \le z \le z_{end}\} \quad (1)$$

To ensure that $\boldsymbol{I}$ has logarithmic time complexity, the samples are sorted by their z-values in each deep pixel.

The central technique employed by our algorithm is the casting of curved rays through a volume. This area has been researched previously in the context of the creation of artistic effects and scene exploration. Groller and Weiskopf et al. traverse the curved cast rays directly in the physical space (P-space) [1995, 2004], which involves computationally heavy intersection tests of a curved cast ray with the geometrical primitives to find its entry and exit points. Coleman et al. offer, in addition to P-space based computations, an approach that is based on the computational space (C-space) [2005]. The definition of C-space ensures that the curved rays become straight and aligned with C-space's coordinate axis. Consequently, computations in C-space dramatically simplify ray-casting. However, translating the geometry into C-space, (e.g. resampling the geometry into the new-axis aligned grid structure) can lead to significant error. Another related area is casting straight rays through curved data. This area also has been addressed with P-space approaches [e.g. Uselton 1991, Ramamoorthy and Wilhelms 1992, Hong and Kaufman 1999] and C-space approaches that are based on “unbending” geometry, rather than rays [e.g. Fruhauf 1994]. In our ray-casting, we use both a P-space and a C-space based traversal. In a typical stereo composition setup, the overlapping volume between the two stereo-rigs' regions is generally sparse (e.g. usually only includes a ground plane and some props); otherwise objects of the farthest region would be occluded. We exploit the sparse nature of our data and build a performance accelerating quadtree-like structure per each horizontal slice of the deep images in the overlapped area. The quadtree's cells are aligned with the coordinate axes of the curvilinear C-space, defined by the curved cast rays. Consequently, we can effectively skip all empty space as we follow our cast rays. In general, since resampling leads to sampling errors, we do not attempt to resample the input geometry into our curvilinear structure. Instead, we cast curved rays through non-empty leaves in P-space until the opacity reaches one. By doing this, we combine the advantages of both approaches. The C-space based ray traversal lets us quickly jump through regions without geometry while the P-space based traversal ensures that we do not compromise quality in areas where we have data.

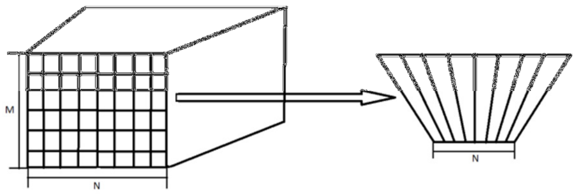


**Figure 3:** *An example of a NxM deep image generated for perspective view (on the left) and a horizontal slice of the pixel stack (on the right)*

## 3 Deep Compositing Algorithm

We start our discussion of the algorithm with the stereo setup and the problem definition. To ensure optical axis convergence, we shift the cameras' image planes by a user provided Horizontal

Image Translation (HIT) value, keeping the image planes perpendicular to the same viewing direction. We define the camera space with the origin at the mid-point of the left and right cameras' focal points. Our x-axis is given by the offset of the focal point of the right camera from the focal point of the left camera. The camera's viewing direction represents the direction of the z-axis. The y-axis is given by the cross-product of the other two axes. Given two intersecting cast rays, originated at pixels with coordinates ($i_L$, $j$) and ($i_R$, $j$) on the left and right cameras' image planes on a slice $j$, disparity is computed at their intersection by subtracting the horizontal component of the right pixel from the horizontal component of the left one, i.e.

$$disparity = i_L - i_R \qquad (2)$$

Thus along the viewing direction (i.e. the z-axis) the disparity progresses from negative to positive. Since the image planes of the camera rig are offset in the x direction, we have a constant disparity corresponding to any particular z-depth.

Although our algorithm trivially extends to setups with an arbitrary number of stereo camera rigs, for the sake of simplicity we describe the algorithm for multi-rigs with two stereo pairs of cameras – $\boldsymbol{Cam^L_1}$ / $\boldsymbol{Cam^R_1}$, and $\boldsymbol{Cam^L_2}$ / $\boldsymbol{Cam^R_2}$ with focal points at $F^L_1$, $F^R_1$, $F^L_2$, and $F^R_2$ (see Figure 4). Their corresponding deep images are given by the deep image functions $\boldsymbol{I^L_1}$, $\boldsymbol{I^R_1}$, $\boldsymbol{I^L_2}$, and $\boldsymbol{I^R_2}$, as defined by (1). The user-specified inter-axial distances and HIT values of the cameras are set to ensure that $\boldsymbol{Cam^L_1}$ / $\boldsymbol{Cam^R_1}$ would produce a specific stereoscopic depth effect for foreground objects and $\boldsymbol{Cam^L_2}$ / $\boldsymbol{Cam^R_2}$ would output a different desired stereoscopic depth effect for background objects. The first stereo camera rig's rendering volume, bounded by the $near_1$ and $far_1$ clipping z-planes, overlaps with the second stereo rig's volume, bounded by the $near_2$, $far_2$ clipping z-planes (*i.e.* $near_1 < near_2 < far_1 < far_2$), thus forming the overlap region between $near_2$ and $far_1$ z-planes. We assume that our multi-rig system is disparity-consistent, meaning that the stereo disparity of the first rig at the $near_2$ z-depth is less than the stereo disparity of the second rig at the $far_1$ z-depth.

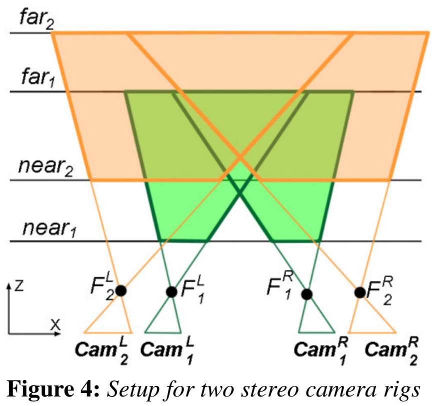


**Figure 4:** *Setup for two stereo camera rigs*

Our goal is to produce two 2D rgb images that are the composited result of the left and right views. Conceptually, we need to merge three volumetric regions (for each view) – (1) the $near_1$ - $near_2$ region, solely controlled by the first pair of the deep images, (2) the $near_2$ - $far_1$ overlapping (blending) region, and (3) the $far_1$ - $far_2$ region, derived from the second pair of deep images. It is crucial that the overlapping blending regions be defined as a seamless transition between the first pair of the deep images and the second. In addition, the user should have intuitive controls to art-direct the behavior in the blending region, by prescribing the influence of each camera pair. Without losing generality, we derive construction of a composite for the left view; the right view can be analogously produced.

In our method, we utilize conventional ray-casting, where the rays are cast along the viewing direction. For each pixel ($i$, $j$) of the output 2D image, color $c_{i,j}$ is accumulated along ray $l_{i,j}$, in the viewing, direction as

$$c_{i,j} = a_0 c_0 + (1 - a_0)\, a_1 c_1 + (1 - (1 - a_0)\, a_1)\, a_2\, c_2 + \dots \qquad (3)$$

where: ($c_0$, $a_0$), ($c_1$, $a_1$), ($c_2$, $a_2$), ... are respectively opacity and color components of samples located along $l_{i,j}$ and sorted by their z-depth components: $z_0$, $z_1$, $z_2$, ... In our case the viewing direction is not well defined since the viewing direction of the same deep pixel ($i$, $j$) is different for the first and second deep images. Thus the key challenge of solving (3) is to determine $\boldsymbol{T}_{near1,\, far2}$, a set of sample triplets ($c$, $a$, $z$) along a cast ray $l_{i,j}$ between $near_1$ and $far_2$

$$\boldsymbol{T}_{near1,far2} = \{(c_0, a_0, z_0), (c_1\, a_1, z_1), \dots\} \qquad (4)$$

where $near_1 < z_0,\ z_1\ , \dots < far_2$. $\boldsymbol{T}_{near1,\, far2}$ can be subdivided into three subsets $\boldsymbol{T}_{near1,\, near2}$, $\boldsymbol{T}_{near2,\, far1}$, $\boldsymbol{T}_{far1,far2}$, based on the $near_1$, $near_2$, $far_1$, $far_2$ z-depths. Since the region bounded by $near_1$ and $near_2$, is controlled solely by the first deep image $l_{i,j}$ , it can be represented as a straight line segment aligned with the viewing direction of deep pixel ($i$, $j$) of the first deep image. Consequently the samples can be obtained by locating the deep pixel's samples with z-components between $near_1$ and $near_2$, i.e. $\boldsymbol{T}_{near1,near2} = \boldsymbol{I^L_1}(i, j, near_1, near_2)$. Analogously in the $far_1$-$far_2$ region, $\boldsymbol{T}_{far1,far2} = \boldsymbol{I^L_2}(i, j, far_1, far_2)$. Therefore (4) can be rewritten as

$$\boldsymbol{T}_{near1,far2} = \boldsymbol{I^L_1}(i, j, near_1, near_2) \cup \boldsymbol{T}_{near2,far1} \cup \boldsymbol{I^L_2}(i, j, far_1, far_2) \qquad (5)$$

Thus determining $\boldsymbol{T}_{near1,\, far2}$ reduces to finding $\boldsymbol{T}_{near2,\, far1}$.

## 3.1 Shape Definition of Cast Ray

One possible alternative for defining the shape of $l_{i,j}$ in the blending region is to represent it as a straight line segment that connects the end of $l_{i,j}$ portion in the $near_1$-$near_2$ region and the beginning of $l_{i,j}$ portion in the $far_1$-$far_2$ region. Thus a sample $P$ on $l_{i,j}$ in the blending region could be computed using a simple linear interpolation:

$$P = (1 - t) * C_0 + t * C_1 \qquad (6)$$

where $C_0$ is the intersection of $l_{i,j}$ with $near_2$ z-plane, $C_1$ is the intersection of $l_{i,j}$ with $far_1$ z-plane, t is float parameter between 0 and 1 (Figure 5a).

In order to produce a believable stereoscopic visualization, we need to ensure that the disparity function is monotonically increasing in the viewing direction, (i.e. along the casting rays). Disparity naturally increases in the first and the last regions since each of them are based on a single pair of stereo cameras and thus the standard stereoscopic rendering rules are applied. Since the disparity function (2) is monotonic and the monotonic behavior is invariant under affine transformation (6), the resulting disparity in

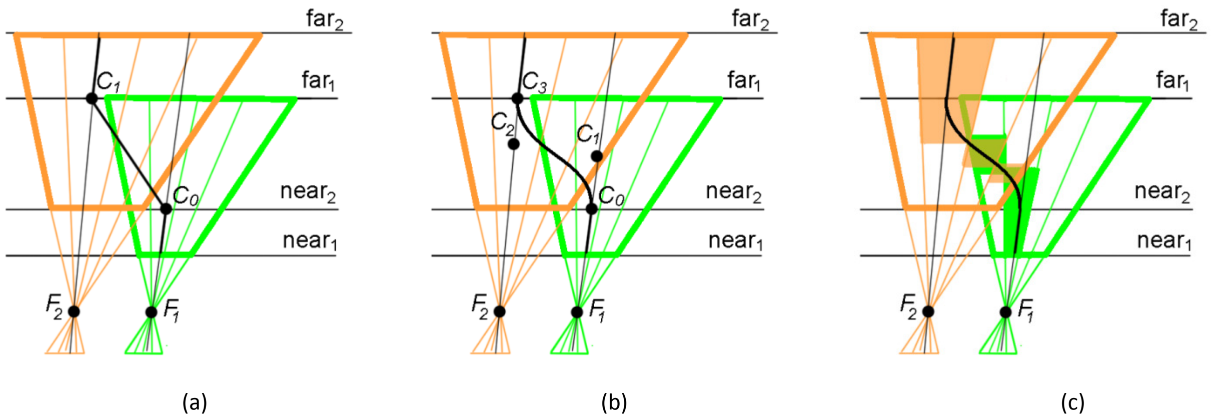


**Figure 5:** *Cast rays' shapes can be defined based on linear interpolation (a) which guarantees monotonic disparity or a higher degree polynomial (b) which provides smooth transition. Samples along cast rays come from both volumes, except areas outside the frustum (c).*

the blending region is monotonic. Its increasing nature comes from the stereo rig being disparity-consistent (i.e. the disparity of the first rig at $near_2$ is less than the second rig's disparity at $far_1$).

Unfortunately, the drawback of the linear definition of $l_{i,j}$ is that it introduces visual seams on the boundaries of the blending region as seen in Figure 6. To avoid these seams and ensure smooth transition between the regions, we use a higher degree polynomial. Inside the blending region we define $l_{i,j}$ using a cubic Bezier curve, which lies on the plane of the *j*-th slice of the deep images. To ensure $C^0$ continuity along $l_{i,j}$, we place the curve's first control vertex $C_0$ at the intersection of $l_{i,j}$ with $near_2$ z-plane. To preserve $C^1$ continuity at $near_2$, where $l_{i,j}$ transitions into the blending region, we offset the second control vertex $C_1$ from $C_0$ in the viewing direction of the deep pixel of the first camera. The other two control vertices, $C_2$ and $C_3$, are similarly defined. (Figure 5b). To enforce monotonic behavior of $l_{i,j}$ in the z-direction, we require $C_1.z < C_2.z$, (and thus the control vertices remain in ascending order along the z-depth).

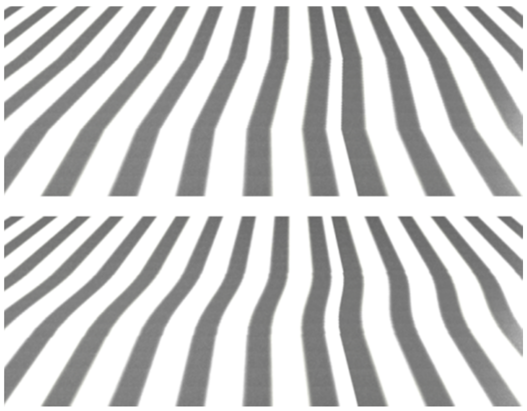


**Figure 6:** *The linear and curvilinear schemes are applied to composite the right cameras' view of a planar surface textured with line segments. Transition area boundaries, clearly revealed in the linear composition (top), are not present when the curvilinear scheme is applied (bottom)*

Our algorithm calculates the positions of the control vertices for each cast ray. However, to guarantee a consistent appearance across all rays, the same ratio $|C_0C_1| / |C_2C_3|$ is used in all control point calculations. The significance of this ratio is that it describes the cameras' influence relative to each other since $C_0C_1$ and $C_2C_3$ represent the viewing directions of the pixel in the two deep images. By letting users prescribe the desired ratio, we provide them with artistic control over the blending of the composite. The shape of the cubic Bezier curve approaches the linear interpolation equation as the lengths of $C_0C_1$ and $C_2C_3$ segments shrink. By scaling up and down the segments' length, the user can balance between the monotonically increasing disparity and transition smoothness.

## 3.2 Sampling along Cast Ray

Actual sampling along the ray's path in the blending region is based on discretization of $l_{i,j}$ in terms of deep pixel sample intervals. Let $l_{i,j}$ intersect $k_1$ deep pixels of the first deep image and $k_2$ deep pixels of the second deep image in the blending region; then $l_{i,j}$ encounters the following sets of samples in the first and second deep images respectively:

$$\boldsymbol{T}^{(1)}{}_{near2,\,far1} = \boldsymbol{I}^{\boldsymbol{L}}{}_{\boldsymbol{1}}(i, j, near_2, z_0) \cup \boldsymbol{I}^{\boldsymbol{L}}{}_{\boldsymbol{1}}(i\text{-}1, j, z_0, z_1) \cup \ldots \cup \boldsymbol{I}^{\boldsymbol{L}}{}_{\boldsymbol{1}}(i\text{-}k_1, j, z_{k1-1}, far_1)$$

$$\boldsymbol{T}^{(2)}{}_{near2,far1} = \boldsymbol{I}^{\boldsymbol{L}}{}_{\boldsymbol{2}}(i + k_2, j, near_2, z_0) \cup \boldsymbol{I}^{\boldsymbol{L}}{}_{\boldsymbol{2}}(i+k_2 - 1, j, z_0, z_1) \cup \ldots \cup \boldsymbol{I}^{\boldsymbol{L}}{}_{\boldsymbol{2}}(i, j, z_{k2-1}, far_1)$$

where $z_i$ ($i \in \{0,\ldots, k_1\text{-}1\}$) and $z_j$ ($j \in \{0,\ldots, k_2\text{-}1\}$) are the z components of intersection points of $l_{i,j}$ with the boundaries of the deep pixels' cones in the first and second images respectively (Figure 5c). Although most of the length of $l_{i,j}$ is in the bounding volumes of both images, small portions of $l_{i,j}$ adjacent to $near_2$ and $far_1$ might lie only in one of the two deep images. That typically happens when $(i+k_2)$ is greater than the horizontal resolution $N$ of the deep image, or $(i\text{-}k_1)$ is less than zero. In these cases, samples will be used only from the single corresponding deep image. While traversing the ray's regions that lie in the overlapping portions of the bounding volumes of the deep images, we collect samples from both images and blend them based on their location relative to $near_2$ and $far_1$. We ensure that advancing along $l_{i,j}$ from $near_2$ to $far_1$, the first deep image loses its influence and the second one gains influence. This is handled by the interpolation functions $f_1(a, z)$ and $f_2(a, z)$ that modify opacities of the samples of the first and second deep images respectively:

$$f_1\ (a, z_i) = \begin{cases} a \bullet \left(1 - \dfrac{far_1 - z_i}{far_1 - near_2}\right) & if\ i - k_1 \geq 0 \\ a & if\ i - k_1 < 0 \end{cases}$$

$$f_2(a, z_i) = \begin{cases} a \bullet \dfrac{far_1 - z_i}{far_1 - near_2} & if\ i + k_2 < N \\ a & if\ i + k_2 \geq N \end{cases}$$

Thus $\boldsymbol{T}_{near2,far1}$ needed for (5) can be defined as a set of $\boldsymbol{T}^{(1)}_{near2,\,far1}$ and $\boldsymbol{T}^{(2)}_{near2,far1}$ samples with modified opacities:

$$\boldsymbol{T}_{near2,far1} = \{\ c, f_1(a), z) \mid (c, a, z) \in \boldsymbol{T}^{(1)}_{near2,far1}\ \} \cup \{\ c, f_2(a), z) \mid (c, a, z) \in \boldsymbol{T}^{(2)}_{near2,far1}\ \}$$

## 4 Implementation using Curvilinear Quadtrees

In our implementation, since samples are already pre-sorted inside their deep pixels, looking them up for a particular z-range is a fast operation with logarithmic complexity and does not involve any expensive arithmetic. However, if the stereoscopic parameters of the composited regions are vastly different, and the input deep images have high resolution, each ray $l_{i,j}$ has to be cast through a large number of deep pixels, which results in a significant number of intersection tests between $l_{i,j}$ and the deep pixels' boundaries. Consequently, in these cases, the large number of intersection tests becomes a performance issue. We observe that in practice the first stereo camera is aimed at the near objects, while the second camera is aimed at the background objects, and there is a relatively large volume of empty space between them. Thus the vast majority of the deep pixels' regions that the rays cross are empty. Therefore a significant portion of the intersection tests do not contribute directly to the resulting color of a rasterized pixel.

To take advantage of this observation, we present a performance acceleration data structure specifically aimed to detect and skip empty areas along ray paths. Our data structure, built for each slice of the deep image, is a variation of a quadtree, which has been optimized for traversing rays in the viewing direction. To reduce the curved ray/deep-pixel intersection tests to simple float-comparison operations, we construct our data structure in curvilinear space $\boldsymbol{C}$ where the curved rays become straight lines.

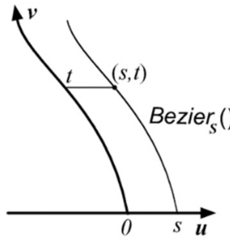


**Figure 7:** *In our curvilinear system of coordinates, the shape of curved v axis depends on s (coordinate on u axis)*

The curvilinear space $\boldsymbol{C}$ is a 2D parameterized space with one dependent axis, defined over a slice of the deep image. The first coordinate axis of $\boldsymbol{C}$, called *u,* coincides with the horizontal direction of the image plane. Consequently, *s,* a *u* coordinate, is the horizontal component of some pixel from the given slice. The second coordinate axis of $\boldsymbol{C}$, called *v*, depends on *s* and is parallel to $Bezier_s$ - a Bezier curve that is a part of the cast ray, originated at the pixel with the horizontal component, *s* (Figure 7). Thus a point *(s,t)* in $\boldsymbol{C}$ can be mapped to the camera's 3D point as:

$$(s, t) = s\,u + t\,v = Bezier_s(t) \quad (7)$$

Since we are only interested in points on the cast rays; $\boldsymbol{C}$ is defined only over the domain of such *(s,t)* tuples such that $s \in \mathbf{N}$, $t \in \mathbf{R}$, and *t* is within the parameterization domain of $Bezier_s$.

To construct the quadtree in the space $\boldsymbol{C}$, we recursively subdivide the quadtree nodes until either the current node does not contain any samples or some maximum depth is reached, as shown in the pseudo-code below:

```
def buildBranch( deepImage, s0, s1, t0, t1, depth ):
  if ( depth < maxDepth ):
      if ( isEmptyArea( deepImage, s0, s1, t0, t1 )):
          createLeaf (s0, s1)
     else:
          buildBranch( deepImage, s0, ( s0+s1 ) / 2, t0,
                    ( t0+t1 ) / 2, depth+1 )
          buildBranch ( deepImage, s0, (s0+s1 ) / 2,
                    ( t0+t1 )/2, t1, depth+1 )
          buildBranch( deepImage, ( s0+s1 ) / 2, t1, t0,
                    ( s0+s1 )/2, depth+1 )
          buildBranch( deepImage, ( s0+s1 ) / 2, s1,
                    (t0+t1 )/2, t1, depth+1 )
  else:
       createLeaf( s0, s1 )
```

In *isEmptyArea()*, we first compute four corners, (*s0, t0*), (*s0, t1*), (*s1, t0*), (*s1, t1*) in the camera space using (7) and obtain their z-depth range $z_{min}$, $z_{max}$. We then multiply the corners by the image view matrix, obtaining four pixel coordinates. Since these four pixels all belong to the same slice, they should have the same vertical component. Thus, on the image plane, we can construct a bounding line that includes the four corners. Finally using the deep pixel function, we check, in logarithmic time, to see if there is at least one sample within the depth range $z_{min}$, $z_{max}$ of any of deep pixels lying on the bounding line. Thus, we can construct our data structure without performing explicit line-line intersection tests.

To streamline the traversing of the leaves, we alter the conventional quadtree data structure, by requiring each leaf to point only to its immediate neighbor(s) in the viewing direction. After we initialize these pointers, we can reduce our memory footprint of the data structure by deleting all branch nodes and keeping pointers only to the first row of the leaves (Figure 8). Also for each leaf, we store bounding coordinates in the horizontal direction, $s_{min}$ and $s_{max}$.

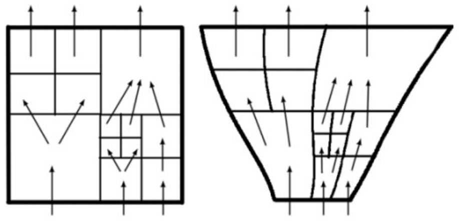


**Figure 8:** *Example of quadtree data structure in curvilinear space (left) and camera space (right). The arrows indicate pointers to next neighbors in the viewing direction.*

Taking advantage of Axis Aligned Bounding Box (AABB) orientation of the leaves, we trace the curved rays through our data structure. For each ray $l_{ij}$ in the *i*-th slice we locate the seed leaf, which is a leaf in the first row with bounding coordinates in the horizontal direction, $s_{min}$ and $s_{max}$, such that $s_{min} < j < s_{max}$.

(Since the leaves are sorted, locating the seed has logarithmic complexity.) If the seed leaf contains samples, we accumulate color as described in the previous section before proceeding with the proper immediate neighbor with *u* range that contains *j*. Otherwise, we skip the area covered by the current leaf and jump to the next leaf in the viewing direction. If there is more than one immediate neighbor, we compare the *u* ranges of each leaf and select the proper one to proceed with. We keep repeating this step advancing until either the accumulated opacity becomes one, or we reach the boundary of our tree.

## 5 Results

We applied our method to the 2048 x 858 deep images that are shown in Figure 9 and reflect a setup for a typical scene – foreground and background object sets along with the ground shared by both sets. Our data structure yielded a two times speedup compared to the non-quadtree based implementation. Due to accumulated opacity prior to reaching $near_2$, 30% of the cast rays did not need to cast through the blend region. About a half of the remaining rays did not accumulate any color over the blend region, and their color was controlled exclusively by the second camera rig. The quadtrees for their slices had the minimum depth, i.e. 1, which allowed us to skip the entire transition region without explicit intersection tests. The remaining cast rays had to travel across up to 20 deep pixels, but our quadtree structure (with maximum depth 7) reduces the number of explicit intersection tests on average to 3 per each ray. We build and process quadtrees in parallel (one thread per slice). A full composition of this example runs on a 16 core machine at 2.7 GHz per core with 64 GB RAM in 20 seconds.

Our work is inspired by real-world production needs and naturally fits into the stereo layout artists' workflow. It typically consists of three main steps - (1) identifying objects of the interest at various z-depths, (2) assigning stereoscopic parameters to achieve the desired stereo volume at their depths, and (3) adjusting viewing angles and camera clipping planes to hide discontinuities. Our proposed solution simplifies the workflow by eliminating the third step since now the user can blend between the desired stereo settings. Figure 10 shows that a desired stereo volume for the front tree makes the background castle look flat; while a stereo volume needed to "inflate" the castle makes the tree appear too close to the viewer, which causes extreme discomfort. If we apply our algorithm to the overlapping area starting just past the first foreground tree and ending before the castle's wall, we achieve a smooth transition between the stereo volumes (Figure 10 c).

## 6 Conclusion and Future Work

We have introduced a new method to perform seamless stereo compositing without the requirement of clearly separable objects. Composition is performed by bending cast rays from the viewing direction of the first set of cameras to the viewing direction of second rig. To ensure the practical value, we provided the user with artistic control over the blending and accelerated the algorithm with specialized quadtee data structure.

We have encountered several areas that we will address in future. These areas include view-dependent shading, which is a general problem for stereo and causes artifacts when specular highlight register in only one eye. Another area has to do with quadtree optimization for handling dense transition regions such as clouds or smoke.

## References

BROSZ, J., SAMAVATI, F., SHEELAGH, M., and SOUSA, M. 2007. Single camera flexible projection. In *the 5th international Symposium on Non-Photorealistic Animation and Rendering*, pp. 33-42.

COLEMAN P., SINGH, K., BARETT, L., SUDARSANAM, N., and GRIMM, C. 2005. 3D screen-space widgets for non-linear projection. In *the 3rd international Conference on Computer Graphics and interactive Techniques in Australasia and South East Asia,* pp. 221-228.

FRUHAUF, T. 1994. Raycasting of Nonregularly Structured Volume Data, *Computer Graphics Forum (Eurographics'94)*, pp. 294-303.

GROLLER, E. 1995. Nonlinear ray tracing: Visualizing strange worlds. *The Visual Computer*, vol. 11, No. 5, pp. 263-274.

HOLLIMAN, N. 2004. Mapping perceived depth to regions of interest in stereoscopic images. In *SPIE*, vol. 5291, pp. 117–128.

HONG, L., and KAUFMAN, A. 1999. Fast Projection-Based Ray-Casting Algorithm for Rendering Curvilinear Volumes. *IEEE Transactions on Visualization and Computer Graphics*, vol 5, pp. 322-332.

JONES, G., LEE, D., HOLLIMAN, N., and EZRA, D. 2001. Controlling perceived depth in stereo-scopic images. In *SPIE*, vol. 4297, pp. 42–53.

KOPPAL, S., ZITNICK, C., COHEN, M., KANG, S. B., RESSLER, B., and COLBURN, A. 2011. A viewer-centric editor for 3D movies. *IEEE CG&A,* vol 31, issue 1, pp. 20–35.

KIM C., HORNUNG, A., HEINZLE, S., MATUSIK W., and GROSS M. 2011. Multi-perspective stereoscopy from light fields. In *SIGGRPAH Asia 2011,* pp. 190:1-190:10.

LANG, M., HORNUNG, A., WANG, O., POULAKOS, S., SMOLIC, A., and GROSS, M. 2010. Nonlinear disparity mapping for stereoscopic 3D. In *SIGGRAPH 2010,* pp 75:1-75:10.

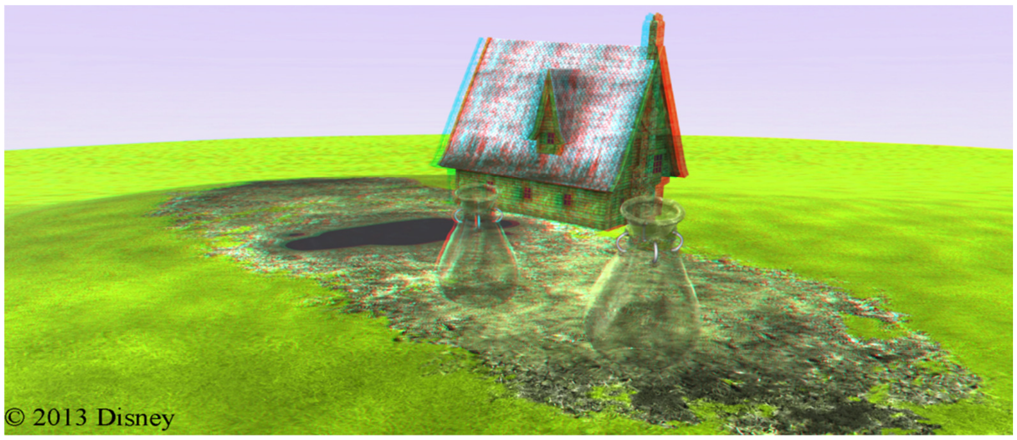
LOKOVIC, T. and VEACH, E. 2000. Deep shadow maps. In *SIGGRAPH 2000,* pp. 385-392.

MENDIBURU, B. 2009. 3D Movie Making: Stereoscopic Digital Cinema from Script to Screen. *Focal Press.*

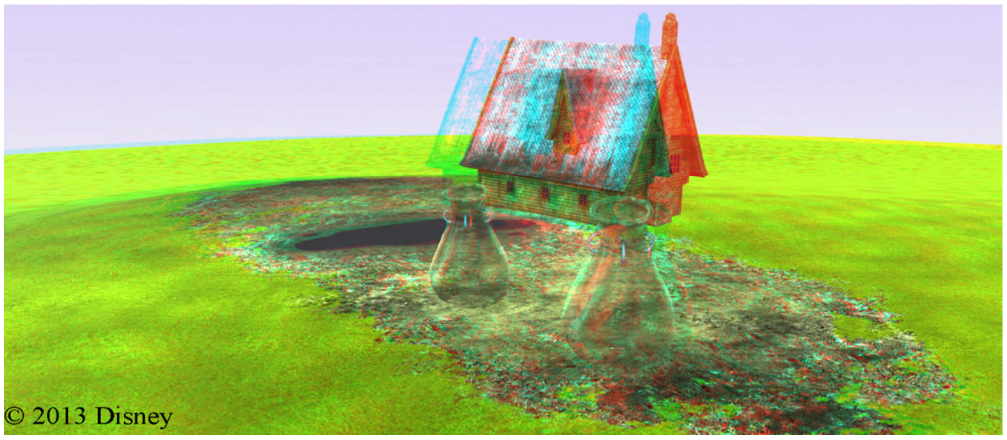
RAMAMOORTHY*, S. and* WILHELMS, J. *1992.* An analysis of approaches to ray-tracing curvilinear grids. *Tech. Report UCSC-CRL-92-07,* University of California, Santa Cruz CA, USA.

USELTON, S. 1991. Volume rendering for computational fluid dynamics: initial results. *Technical Report RNR-91-026*, NAS-NASA Ames Research Center, Moffet Field CA, USA.

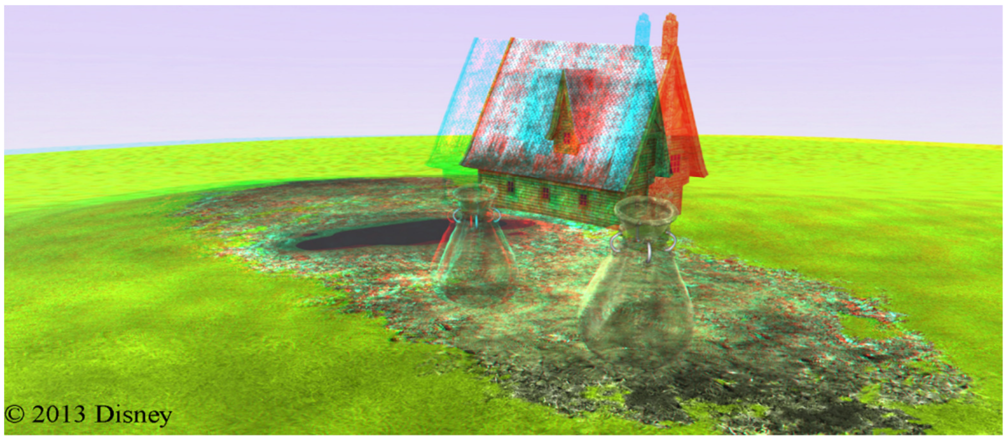
WEISKOPF, D., SCHAFHITZEL, T., ERTL T. 2004. GPU-based nonlinear ray tracing. In *Computer Graphics Forum,* vol 23, issue 3, pp. 625-633.



(a)

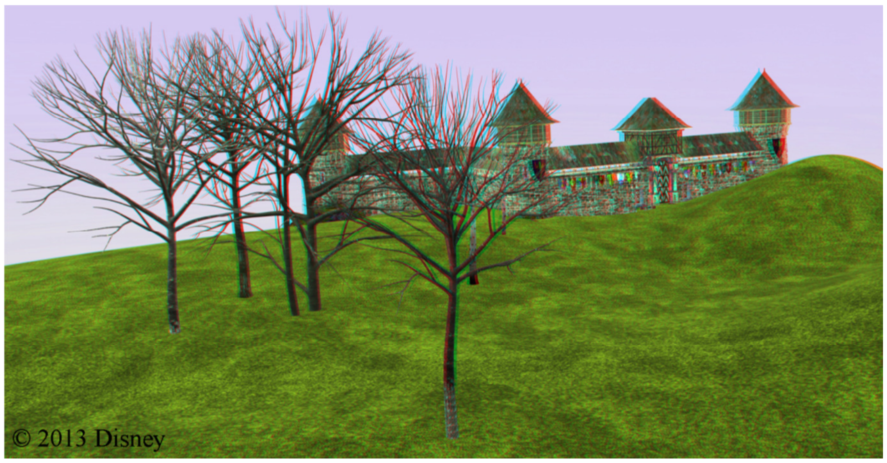


(b)

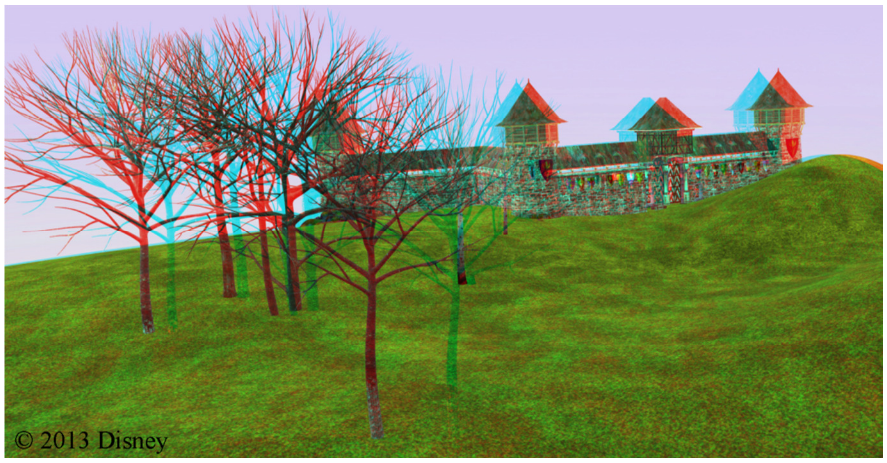


(c)

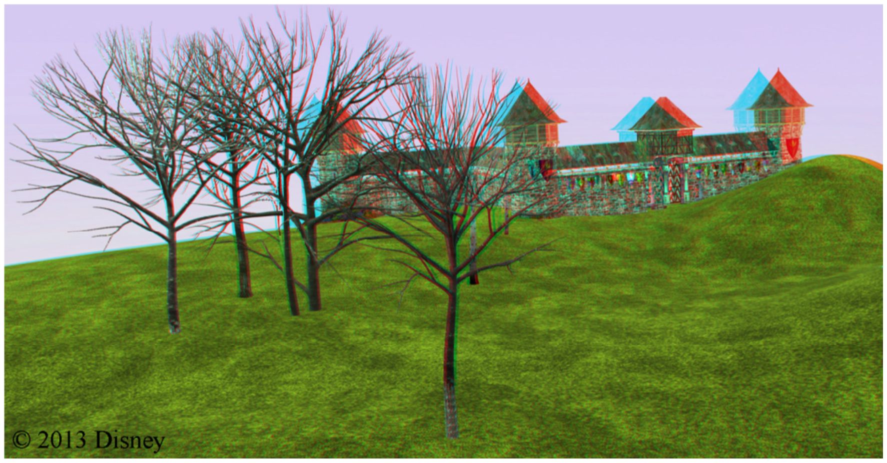
**Figure 9:** Stereo renderings produced by the first and second stereo cameras are shown in (a) and (b). In (c), we see the composited result of the overlapping region starting just after the foregroud semi-transparent sack and ending before the house, which exemplifies handling of semitransparent objects



(a)



(b)



(c)

**Figure 10:** Using stereo settings of the first camera in (a) insure that the front object is at a comfortable viewing volume; though this makes the background appear “flat”. The stereo settings of the second camera in (b) produces the desired stereoscopic volume for the castle, but makes the front objects appear too close to the viewer. Applying our stereo-composition algorithm for the blending region, starting just after the front tree and ending at the castle’s wall, we can produce a smooth transition between the two stereo volumes as shown in (c).