A Rendering Framework for Multiscale Views of 3D Models

Wei-Hsien Hsu* Kwan-Liu Ma† Carlos Correa‡
University of California at Davis Lawrence Livermore National Laboratory

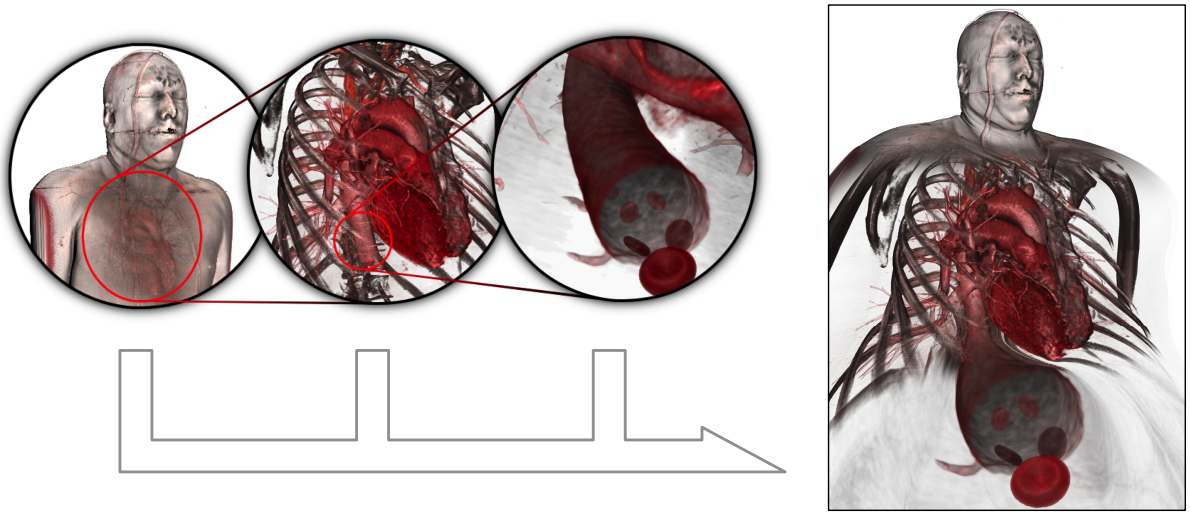


Figure 1: *A continuous multiscale view (right) of a volumetric human body dataset shows three different levels of detail (left three) in a single image. The image on the right is directly rendered with our multiscale framework.*

Abstract

Images that seamlessly combine views at different levels of detail are appealing. However, creating such multiscale images is not a trivial task, and most such illustrations are handcrafted by skilled artists. This paper presents a framework for direct multiscale rendering of geometric and volumetric models. The basis of our approach is a set of non-linearly bent camera rays that smoothly cast through multiple scales. We show that by properly setting up a sequence of conventional pinhole cameras to capture features of interest at different scales, along with image masks specifying the regions of interest for each scale on the projection plane, our rendering framework can generate non-linear sampling rays that smoothly project objects in a scene at multiple levels of detail onto a single image. We address two important issues with non-linear camera projection. First, our streamline-based ray generation algorithm avoids undesired camera ray intersections, which often result in unexpected images. Second, in order to maintain camera ray coherence and preserve aesthetic quality, we create an interpolated 3D field that defines the contribution of each pinhole camera for determining ray orientations. The resulting multiscale camera has three main applications: (1) presenting hierarchical structure in a compact and continuous manner, (2) achieving focus+context visualization, and (3) creating fascinating and artistic images.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms;

Keywords: multiscale views, camera model, levels of detail, visualization

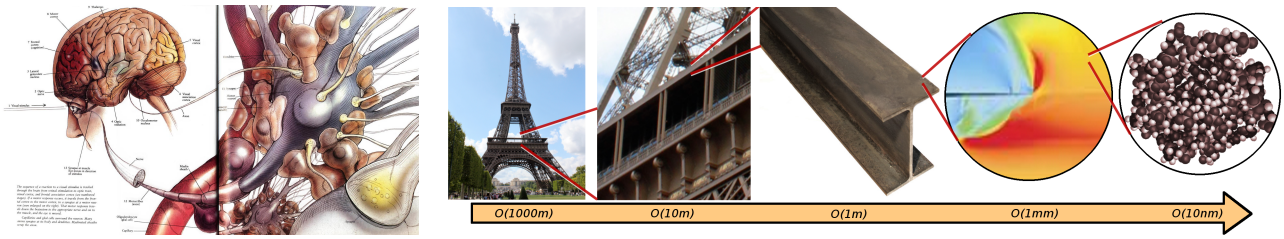
Links: DL PDF

*e-mail: whhsu@ucdavis.edu
†e-mail: ma@cs.ucdavis.edu
‡e-mail: correac@llnl.gov

1 Introduction

This project is motivated by an illustration created by artists at the Exploratorium in San Francisco. As shown in Figure 3, this illustration depicts both macro and micro perspectives of the human circulatory system in a continuous landscape across multiple scales. The seamless continuity between scales vividly illustrates how molecules form blood cells, how blood cells are distributed in a blood vessel, how the blood vessel connects to a human heart, and where the heart is located in a human body. The astonishment and fascination evoked by the illustration, along with its high educative value, won it first place in the illustration category of the 2008 U.S. National Science Foundation and Science Magazine Visualization Challenge.

In scientific studies, it is often desirable to illustrate complex physical phenomena, organic structures, and man-made objects. Many of these physical structures are hierarchical in nature. Static multiscale illustrations are frequently used to convey hierarchical structures, such as the anatomy of organ systems and the design of engineered architectures, as shown in Figure 2. Large terrain data, on the other hand, is usually encapsulated in explorable, navigable interactive systems [McCrae et al. 2009; Google 2010]. Animations are also helpful for presenting extremely large datasets



(a) (b)

Figure 2: *Examples of multiscale illustrations. (a) A hand-drawn illustration by Carol Donner [Bloom et al. 1988], depicting the hierarchical structure of the human nervous system. (b) A multiscale illustration of the Eiffel Tower using a zoom-in metaphor.*

which are difficult for users to navigate directly, such as those consisting of the solar system and the universe [Cosmic Voyage 1996; The Known Universe 2009].

Figure 3: *"Zoom Into the Human Bloodstream" by Linda Nye and the Exploratorium Visualization Laboratory [Nye 2008]. With permission from Exploratorium, San Francisco, CA, USA. All rights reserved.*

Out of the above scenarios, the creation of multiscale illustrations is the most challenging because there is no direct way to project a complex hierarchical 3D scene to a 2D image. In this paper, we focus on generating continuous multiscale images. Unlike traditional multiscale illustrations, in which each scale is displayed separately from others (Figure 2(b)), a continuous multiscale image shows objects at several levels of detail with smooth object-space continuity between different scales. Although multiperspective rendering has received some attention [Yu et al. 2008], previous work has not specifically addressed seamless multiscale image rendering.

We introduce a rendering framework which generates and casts non-linearly bent rays into a 3D scene, and projects multiple scales of interest onto a single image. Our multiscale rendering framework consists of a sequence of pinhole cameras, each of which captures a view of interest at a particular scale. The camera rays for each pixel in the final projected image are generated based on a user-defined image mask which specifies the interesting regions in each view. The rays are bent gradually from one scale to another to maintain object-space continuity as well as image-space smoothness. Our framework can be used on both polygon models and volumetric data. The resulting views are useful in many contexts. The most direct application is the presentation of complex hierarchical structures, as shown in Figure 1. Our multiscale camera can also achieve a focus+context effect, a technique frequently used in many visualization applications. Finally, we show that it can potentially create pictures that mimic artistic or impossible views of objects or scenes like the kind made by the artist M. C. Escher.

2 Related Work

Camera projection is fundamental to computer graphics, since every 3D scene uses a projection to form a 2D image. As a result, various camera models have been developed for different scene modeling and rendering purposes. The General Linear Camera Model (GLC) developed by Yu and McMillan [2004b] uses three rays to define the affine combination and to generate other sampling rays. GLC is capable of modeling most linear projections, including perspective, orthogonal, push-broom, and crossed-slits projections. GLC was further extended in the Framework for Multiperspective Rendering [Yu and McMillan 2004a], which is achieved by combining piecewise GLCs that are constrained by an appropriate set of rules and interpolated rays that are weighted based on the distance to the closest fragment in a GLC region. Another type of camera model uses image surfaces to explicitly specify how sampling rays propagate in a scene [Glassner 2000; Brosz et al. 2007]. In Glassner's work, rays are defined by only two NURBS surfaces, whereas Brosz et al. used parametric surfaces to define a flexible viewing volume, and are thus able to accomplish non-linear projections such as fish-eye or cylindrical projections. However, although these camera models have employed different methods to contruct view frusta to achieve either linear or non-linear camera projections, their approaches focus on the manipulation of a single viewpoint and cannot achieve the multiscale projection that we desire.

Much work has been done on creating non-linear camera projections. Wang et al. [2005] presented a camera lens technique based on geometric optics for volume rendering. Camera rays are refracted according to the selected lens type at the image plane before being shot into the 3D volume. But since the lens is put in front of the image plane, it can only achieve a limited magnification within a single viewing direction. Sudarsanam et al. [2008] created camera widgets that encapsulate specific aspects of non-linear perspective changes and allow users to manipulate both the widgets and their image-space views. Instead of explicitly modifying camera rays, Mashio et al. [2010] introduced a technique that simulates non-perspective projection by deforming a scene based on camera parameters. But these two methods focus on revealing or magnifying multiple different interesting regions in a scene.

The Graph Camera developed by Popescu et al. [2009] introduces three basic construction operations on the frusta of planar pinhole cameras (PPC) to connect several viewing regions in a 3D scene. Similar to the Graph Camera, Cui et al. [2010] introduced the curved ray camera, which generates a bent view frustum based on a sequence of PPC's and provides C^1 continuity at the connections between each PPC segment to circumvent occluders. Both the Graph Camera and the curved ray camera connect successive PPC's by first overlapping the frusta and then binding the piecewise trimmed frusta. However, since the PPC's are bound in a se-

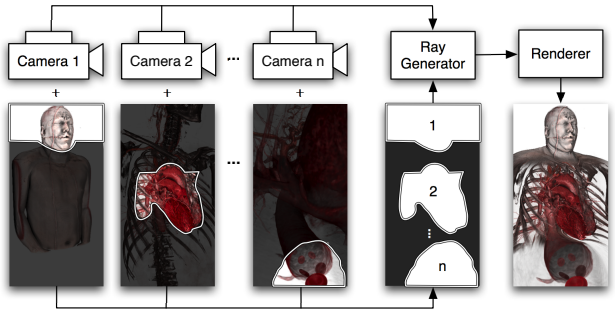


Figure 4: *Dataflow of multiscale rendering. The process starts by setting up separate pinhole cameras for different scales of view and image masks which indicate interesting regions in each view. Image masks are merged into a single image and passed to the camera ray generator, along with camera information. Non-linear bent rays are generated accordingly and used to sample the scene.*

quential way that one camera is placed after another, the piece-wise viewing frustum is intrinsically diverging due to the nature of perspective projection. Although the Graph Camera supports frustum splitting and can possibly achieve a convergent viewing frustum by merging two PPC's with converging viewing directions, it is hard to set up PPC's in this way so as to obtain sufficient multiscale magnification; the curved ray camera only supports sequential frustum bending and can never achieve a convergent frustum which is essential in creating multiscale effects. In short, their approaches focus on revealing hidden objects in a large 3D scene, whereas ours is designed for visualizing multiple levels of detail of the same object.

Other types of multiscale or focus+context rendering include image-space or object-space deformation. Böttger et al. [2006] presented a technique for generating complex logarithmic views for visualizing flat vector data. A similar technique was later employed to visualize complex satellite and aerial imagery, showing details and context in a single seamless image [Bttger et al. 2008]. But their approaches are mainly designed for generating flat cartographic map projections. Focus+context effects for 3D data can also be achieved by distorting the object so as to magnify certain focal regions [Carpendale et al. 1996; Wang et al. 2008; Wang et al. 2011]. However, deforming objects can possibly lead to severe distortion if high magnification is demanded.

3 Multiscale Image Composition

A continuous multiscale image is composed of two types of regions. The first type consists of the ordinary perspective views at each scale of interest, and the other type consists of the transitions that smoothly connect views at different scales. An intuitive way to create views at multiple scales is to use several pinhole cameras, with each camera capturing a perspective view at a different scale. In order to render transitions between scales, a naive approach is to separately render each view and then blend them together using either an illustration metaphor (as shown in Figure 2(b)), or a seamless image-stitching algorithm, such as the graph cut [Kwatra et al. 2003] or Poisson image editing [Pérez et al. 2003]. Although the images created by these methods appear to be seamless, the underlying content is not continuous in object-space, and thus can make it difficult for viewers to comprehend the true spatial relations between scales.

We introduce a multiscale rendering framework that generates camera rays non-linearly cast through all scales of interest. Since the camera rays in our model are bent coherently and march consistently, the objects projected on the image are continuous in both image-space and object-space. Our approach starts by setting up several pinhole cameras for viewing different scales of interest, utilizing most users' ease and familiarity with manipulating ordinary pinhole cameras. Each camera produces an image of its view. In order to combine all such views to form a single multiscale image, we use a user-specified image mask to indicate regions of each view that the user would like to show in the final image. In other words, every camera projects only part of its view onto the final image space, based on a corresponding image mask as shown in Figure 4.

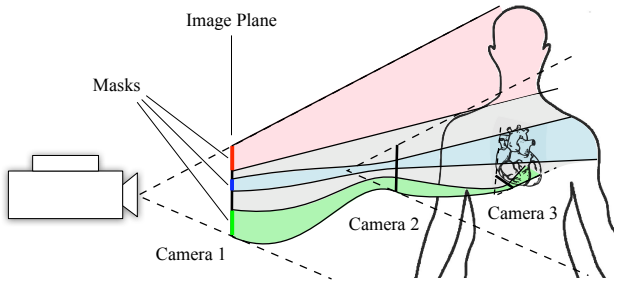


Figure 5: *Through careful image mask and camera placement, camera rays can cast through multiple scales to capture features of interest.*

In order to ensure consistency while projecting different camera viewpoints onto different parts of the image, we force all camera rays to originate from the first camera, which is the one that has the largest scale of view. The rest of the cameras define intermediate points that camera rays must pass through in order, from the largest to smallest scales. We use Bézier curves to connect the near-clipping planes of two successive cameras, as described in the following section. Figure 5 depicts the resulting non-linear viewing frustrum based on the input pinhole cameras and image masks.

The gray regions between colored regions in Figure 5 are transition areas where camera rays need to gradually change between nearby colored regions to maintain ray coherence. Two things are worth noting when dealing with camera ray bending in transition regions:

Ray intersection. Camera rays that are bent at unequal degrees have a chance to intersect with one another. Simple ray direction interpolation such as linear interpolation between two nearest preserved rays can generate intersecting rays that might result in the same object being projected onto the image more than once.

Ray coherence. Adjacent camera rays must be coherent so as to avoid too much distortion in the resulting image. For example, camera rays which are emitted from the pixels in a horizontal scanline should maintain their spatial relationship after bent.

The first issue is directly related to whether sampling rays can project the scene correctly. The second issue if properly addressed can minimize distortion and lead to aesthetically appealing view. We discuss these two issues and our solutions in the next section.

4 Camera Ray Generation

In order to generate camera rays that seamlessly blend the views specified by the user, we introduce a streamline-based approach. A key step of this approach is the construction of a vector field based on the selected views. Rays are derived by tracing streamlines in this vector field. The process of generating the camera rays is as follows:

1. Initialize a set of *guide* curves, which connect a sequence of camera planes via Bézier curves.
2. Derive the complete vector field that best matches these curves via minimization.
3. Trace streamlines from each pixel in the image by integrating the vector field.
4. Each streamline defines the path that light traverses to generate a multiscale image.

4.1 Camera Setup

The first step in our design is to set up the cameras that generate the multiscale image. As an input to our process, we define a series of cameras C_i, defined by a position E_i, a look-at vector V_i, a field of view f_i, an aspect ratio a_i and a near clipping distance d_i, for $i \in \{1, \ldots, N\}$. Thus, the near clipping plane of each camera C_i is defined by the point $E_i + V_i d_i$ and the normal vector V_i, and the range is determined by the f_i and a_i. Each camera also has an user-defined binary mask $I_{i,x,y}$ which indicates the preserved viewing regions in its pixel space. As a result, every single pixel has a position $p_{i,x,y}$ and a ray vector defined by $v_{i,x,y} = (p_{i,x,y} - E_i)/|p_{i,x,y} - E_i|$

4.2 Multiscale Frustum Construction

As a preliminary step, we bind successive cameras so that the regions of interest are propagated smoothly along the multiple scales. In order to do that, we trace Bézier curves between two successive cameras. More specifically, given an ordering of the cameras $C_1, \ldots, C_N$, we bind two cameras C_j and C_{j+1} via Bézier curves $B_{j,x,y}(t)$ for each pixel $p_{j,x,y}$, using the pixel positions and ray directions as their control points:

$$B_{j,x,y}(t) = \sum_{k=0}^{3} b_{k,n}(t) Qk \tag{1}$$

with

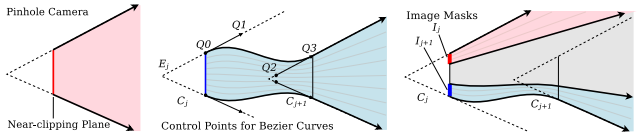
$$\begin{aligned} Q0 &= p_{j,x,y} \\ Q1 &= p_{j,x,y} + v_{j,x,y}\beta|p_{j,x,y} - p_{j+1,x,y}| \\ Q2 &= p_{j+1,x,y} - v_{j+1,x,y}\beta|p_{j,x,y} - p_{j+1,x,y}| \\ Q3 &= p_{j+1,x,y} \end{aligned}$$

where β controls the distances between the first and latter two control points (according to our experiments, $\beta = 0.2$ gives sufficient smoothness) and $b_{k,n}$ are the Bernstein polynomials of degree 3.

Figure 6(a) illustrates a pinhole camera with its viewing frustum highlighted in red. Figure 6(b) shows the ray binding between C_j and C_{j+1} where C_{j+1}'s rays are extended at the near-clipped end to couple with C_j's rays ensuring that every ray originates from C_1. Figure 6(c) depicts ray coupling with the application of image masks. Note that a ray is curved to the next camera only when it originates in the region that is assigned to a descendant camera. In this figure, the red region indicates the preserved view for C_j, and the blue region indicates the preserved view for C_{j+1}. As a result, camera rays in the red region cast linearly, while rays in the blue region proceed from the near-clipping plane of C_j, march forward along Bézier curves, which are constructed based on the original ray directions, and arrive at C_{j+1}'s clipping plane. Rays then continue to proceed linearly to capture the view of C_{j+1}.

4.3 Streamlines as Camera Rays

The previous step defines a series of *guiding* curves that indicate how rays should be traced to realize the multiscale camera. However, we need to ensure that these rays do not intersect and that



(a) (b) (c)

Figure 6: *(a) A pinhole camera and its view frustum. (b) The rays of C_{j+1} are extended backward to couple with C_j using Bézier curves. Small dotted arrows illustrate control points, which are the original ray orientations of the two views. (c) Different portions of the rays are assigned to different views based on the image masks. The red region is marked as a preserved viewing region for C_j, blue is for C_{j+1}, and the gray region is the transition between the two. As a result, interesting features of the two views can be seamlessly shown in the same image.*

they are smoothly interpolated in the transition regions. To achieve this, we think of the problem of camera ray generation as tracing streamlines from an underlying vector field.

Streamlines are a set of curves which depict the instantaneous tangents of an underlying vector field. A streamline shows the direction that the corresponding fluid element travels in the field at any point in space. One characteristic of streamlines is that any two given streamlines would never intersect each other as long as no critical points are present [Fay 1994]. If we treat camera rays as streamlines, we can make use of this characteristic to ensure that no camera ray intersections can occur.

4.4 Estimating The Vector Field

To derive streamlines for use as multiscale camera rays, we must first construct the underlying vector field. Since streamlines represent tracks along values in their vector field, the construction of the vector field determines the paths of the streamlines. The problem then becomes: given a set of pinhole cameras and an image mask, how can we construct a vector field whose streamlines are distributed identically to the original camera rays in preserved regions, and gradually transition between the interpolated regions?

To achieve this goal, we consider the Bézier curves as an initial set of streamlines corresponding to an initial guess of the underlying vector field. A complete curved ray $R_{i,x,y}$ for camera C_i at pixel x, y is defined by a set of Bézier curves $B_{j,x,y}$ where $j = 1, \ldots, i-1$, and a ray at $p_{i,x,y}$ pointing toward $v_{i,x,y}$. These viewing rays are the guideline streamlines used to construct an intermediate ray field. This vector field $\mathbf{X}(\mathbf{x} \in \mathbb{R}^3) = (u(\mathbf{x}), v(\mathbf{x}), w(\mathbf{x}))$ is built by filling in the field with the tangent directions of $R_{i,x,y}$ along the curves if the rays originate from the region assigned to camera C_i in its image mask ($I_{i,x,y} = 1$) for every set of curved rays R_1 to R_N, and zero elsewhere, i.e.,

$$\mathbf{X}(\mathbf{x}) = \begin{cases} \frac{dR_{i,x,y}(t)}{dt} & \text{if } I_{i,x,y} = 1, \text{where } R_{i,x,y}(t) = \mathbf{x}, \\ & \text{for } i = 1, \ldots, N \\ (0,0,0) & \text{otherwise} \end{cases} \tag{2}$$

The final vector field $\mathbf{Y}(\mathbf{x})$ can be derived by solving an optimization problem which ensures that the initial guesses are preserved, but also that the difference between neighboring points is minimal, resulting in smooth transitions across all the camera frusta.

There are two ways to formulate this optimization problem: as a direct vector field optimization problem or as a two-step optimization, where we solve for a scalar field first and then fit a vector field using interpolation.

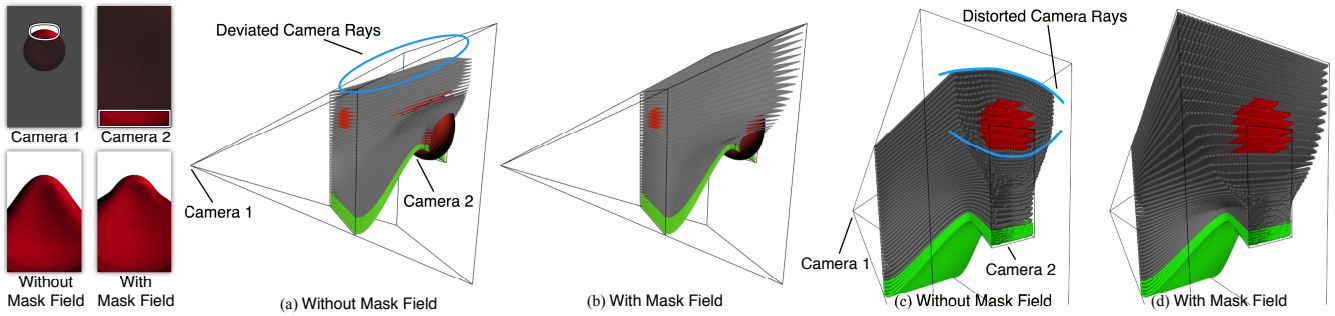


Figure 7: *Comparison of the results from direct vector smoothing and with the use of the scalar field mask. Two camera views with the corresponding masks are shown in the upper left images. The mask for Camera 1 contains only a small portion of the image area, which means only the camera rays in this preserved region are used to fill the view vector fields. The side view in (a) and the rear view in (c) point out that the curved rays generated by using direct vector field smoothing deviate from the expected perspective projection around Camera 1 (highlighted in the blue circle) and cause undesirable distortion (highlighted in the blue arc). (b) and (d) illustrate the coherent rays generated by interpolating the vector fields based on the smoothed scalar field mask. The resulting images are shown in bottom left.*

Vector field optimization. In this case, we formulate the problem solving an equation that minimizes the following energy function [Xu et al. 2010]:

$$\varepsilon(\mathbf{Y}) = \int \varepsilon_1(\mathbf{Y}(\mathbf{x}), \mathbf{X}(\mathbf{x})) + \mu\varepsilon_2(\mathbf{Y}(\mathbf{x}))d\mathbf{x} \tag{3}$$

where

$$\varepsilon_1(\mathbf{Y}(\mathbf{x}), \mathbf{X}(\mathbf{x})) = |\mathbf{X}(\mathbf{x})|^2|\mathbf{Y}(\mathbf{x}) - \mathbf{X}(\mathbf{x})|^2$$
$$\varepsilon_2(\mathbf{Y} = (u(\mathbf{x}), v(\mathbf{x}), w(\mathbf{x}))) = |\nabla u(\mathbf{x})|^2 + |\nabla v(\mathbf{x})|^2 + |\nabla w(\mathbf{x})|^2$$

The term $\varepsilon_1(\mathbf{Y}(\mathbf{x}), \mathbf{X}(\mathbf{x}))$ guarantees that the resulting vector field $\mathbf{Y}$ has exactly the same value as the preliminary field $\mathbf{X}$ at points where $\mathbf{X}(\mathbf{x})$ is not zero, and the second term $\mu\varepsilon_2(\mathbf{Y}(\mathbf{x}))$ is minimized when the neighboring vectors are identical, thus resulting in smooth transitions. The energy equation can be solved by the *generalized diffusion equations* described in the fluid flow literature [Hall and Porsching 1990], and is further discussed in [Ye et al. 2005; Xu et al. 2010].

Unfortunately, although the resulting vector field satisfies our requirements that streamlines pass through all views in preserved regions, and smoothly transition between preserved regions, streamlines in the transitional regions produced by this method may not preserve the characteristics of camera rays in a natural way. To take a simple example, suppose we use only one camera, but that we only assign a small portion of the image mask to the camera. Since only streamlines marked as originating from the image mask are used to fill the vector field, the remaining parts of the resulting smoothed vector field will have the same vectors as the boundary of the marked region and thus fail to project the expected view to the original camera. Figure 7 illustrates a case of two cameras. One can see that the preserved rays for Camera 1 in red only provide a small amount of view vector information. Thus, the neighboring vectors on top of the preserved views have false values that lead to a deviated perspective projection as shown in Figure 7(a). Figure 7(c) shows that the false vector values also cause a distorted distribution of the camera rays that often result in a twisted image.

Scalar Field Optimization. To avoid the distortions in the camera rays, we can alternatively pose the problem as fitting a membership function (scalar) and use this function to interpolate the initial vector fields. Similar to the construction of $\mathbf{X}$, we construct an initial scalar field $\mathbf{M}(\mathbf{x} \in \mathbb{R}^3))$, as shown in Figure 8, where we fill in the mask values along each streamline.

$$\mathbf{M}(\mathbf{x}) = \begin{cases} i & \text{if } I_{i,x,y} = 1, \text{where } R_{i,x,y}(t) = \mathbf{x} \\ & \text{for } i = 1, \ldots, N \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The final scalar field $\mathbf{N}$ can be derived by optimizing a 1-D version of Equation 3:

$$\varepsilon(\mathbf{N}) = \int \varepsilon_1(\mathbf{N}(\mathbf{x}), \mathbf{M}(\mathbf{x})) + \mu\varepsilon_3(\mathbf{N}(\mathbf{x}))d\mathbf{x} \tag{5}$$

where

$$\varepsilon_3(\mathbf{N}(\mathbf{x})) = |\nabla\mathbf{N}(\mathbf{x})|^2$$

and $\varepsilon_1(\mathbf{N}, \mathbf{M})$ is defined as before, but for a scalar field.

To derive a final view vector field, we need to construct a set of intermediate vector fields $\mathbf{X}_1$ to $\mathbf{X}_N$ representing the entire viewing frusta of C_1 to C_N, respectively, where each of the vector fields is defined by $\mathbf{X}_i(\mathbf{x}) = dR_{i,x,y}(t)/dt$ for $R_{i,x,y}(t) = \mathbf{x}$. Based on the smoothed scalar field mask $\mathbf{N}$ and the intermediate view vector fields $\mathbf{X}_1$ to $\mathbf{X}_N$, we can construct a final view vector field using the following function:

$$\hat{\mathbf{Y}}(\mathbf{x}) = \frac{\sum_{i=1}^{n} \mathbf{X}_i(\mathbf{x})\omega(\mathbf{N}(\mathbf{x}), i)}{\sum_{i=1}^{n} \omega(\mathbf{N}(\mathbf{x}), i)} \tag{6}$$

The term $\omega(\mathbf{N}(\mathbf{x}), i)$ is a weighting function that determines the weight for each view vector field according to the mask scalar field. For example, the following weighting function performs linear interpolation between two neighboring view vectors:

$$\omega(\mathbf{N}(\mathbf{x}), i) = \begin{cases} 1 - |\mathbf{N}(\mathbf{x}) - i| & \text{if } |\mathbf{N}(\mathbf{x}) - i| < 1 \\ 0 & \text{otherwise} \end{cases}$$

Other types of interpolation, such as monotonic cubic interpolation, may be applied to increase the smoothness at the boundaries of transitional regions, but increasing the smoothness of boundaries implies a rapid change in the interior of regions. Figures 7(b) and (d) shows the results of using a scalar field to compute the underlying vector field. Because it eliminates distortion and is more computationally efficient, we employ this method in our results.

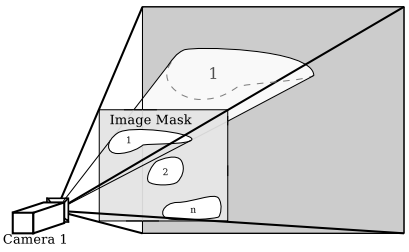


Figure 8: *When filling the mask scalar field, we trace only the camera rays in the corresponding restricted region. The above example illustrates the case where the mask value 1 is filled into the scalar field where the rays of Camera 1 pass through the corresponding region of the image mask.*

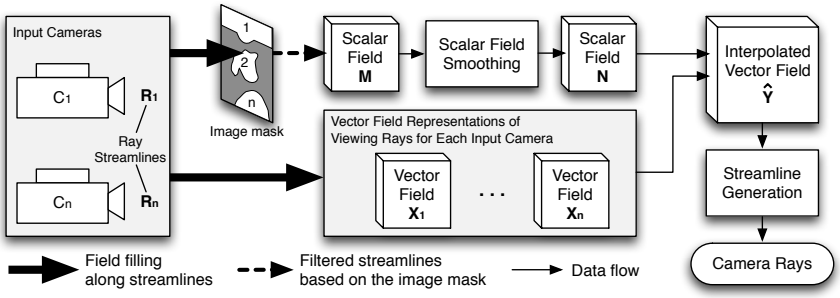


Figure 9: *The process of camera ray generation using scalar field optimization. Input camera rays are taken as streamlines and used to construct view vector fields $\boldsymbol{X}_1$ to $\boldsymbol{X}_N$. At the same time, a scalar field $\boldsymbol{M}$ is filled with the mask value along the filtered streamlines based on the image mask. Ray streamlines are generated upon the view vector field $\hat{\boldsymbol{Y}}$, which is derived by interpolating $\boldsymbol{X}_i$ according to the smoothed scalar field $\boldsymbol{N}$.*

4.5 Streamline Generation

The final step is to generate streamlines from the vector field $\hat{\mathbf{Y}}$. Since streamlines are used to simulate camera rays, each streamline should pass through a pixel on the projection plane (the near-clipping plane of Camera 1). Therefore, we take the position of the pixel on the projection plane as seed points to trace streamlines using the Runge-Kutta method. Once we have all of the streamlines, we can render the final image using these streamlines as sampling rays. The entire ray generation process is summarized in Figure 9.

5 Implementation Details

To implement vector field smoothing and streamline generation, we need to construct a 3D volume of the vector field that encloses the entire space, including all objects and camera frusta. This means we only have a finite number of vector field samples. As a result, the resolution of the volume directly affects the accuracy of streamline integration. This resolution issue becomes especially essential when streamlines are used to cast through multiscale objects, and hence it is imperative to use multi-resolution volumes to achieve adaptive vector sampling.

Rendering 3D scenes with the generated non-linear rays can be done at interactive frame rates thanks to GPU acceleration. However, the iterative process for solving Equation 5 to produce a smooth vector field is computationally expensive. Depending on the desired resolution for calculating the vector field, the computation time varies from a few minutes to several hours. For example, it takes about two hours to compute the vector field in a 320^3 resolution volume for creating Figure 1, using an Intel Core 2 Duo E6600, 2.4 GHz CPU. Parallelization or GPU acceleration can significantly reduce the computational time.

5.1 Ray Tracing

Because our rendering framework requires modification of the camera ray directions, we choose ray tracing as the rendering method. However, most popular raytracers only support linear sampling rays in ray-object intersection tests. In order to achieve non-linear ray tracing, we divide the generated curved rays into line segments and perform piecewise linear ray-object intersection tests.

Our method is also capable of performing multisampling ray tracing to obtain high quality anti-aliasing images. We first divide all the rays into equal numbers of line segments for subpixel sampling. When rendering using multisampling, a subpixel sampling point is determined by averaging the four neighboring points at the corresponding line segments with random weights. The final color can be derived by averaging the sampled colors within a pixel. We extend the last line segments as infinite linear rays to cast through the rest of the scene. Since the ray directions are altered, some of the rays might intersect with each other at certain points. This is acceptable because the intersections only occur outside the regions of interest and thus cause little effect on the projection.

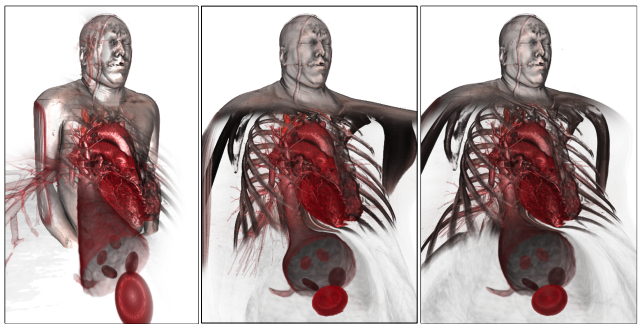
5.2 Volume Ray Casting

For volume data, we use ray casting to sample data values along with transfer functions (TFs) to achieve direct volume rendering. Ray casting, by nature, requires a traversal of the data structure, and thus simplifies the implementation of a non-linear ray marching algorithm. However, interesting features in volume datasets are usually hard to segment clearly, and in many cases important features are wrapped in opaque surrounding volumes. For example, in the human body dataset as shown in Figure 1, organs are embedded in the epidermis and other tissues. Even though the camera rays are bent to couple with the views at small scales, the first thing that the rays hit is the skin. In such cases, the multiscale sampling rays result in a magnified view of the outer skin, and the interesting features such as the heart and blood vessels are all occluded.

In order to clearly visualize the features at different scales, we apply different TFs to the same object while sampling rays with different mask values. TFs define RGBA colors corresponding to data intensities. In other words, TFs define how color and opacity represent data on the screen. In our system, users are allowed to specify separate TFs for each camera viewpoint so as to obtain better views at different scales. During ray casting, TFs between different views are interpolated based on the smoothed mask values to achieve continuous color transitions. The left three images in Figure 1 show three camera views with different TFs applied to the same dataset.

6 Discussion

We have introduced a new capability for creating novel views of 3D models. Figure 10 provides a direct comparison between the results using an image-based approach and our vector field and scalar field optimization. The image-based result in (a) is produced by separately rendering three views and then manually stitching them with the aid of the graph cut algorithm. As a result, the resulting image



(a) Image based approach (b) Vector field optimization (c) Scalar field optimization

Figure 10: *Comparison between the results using an image-based approach, the vector field optimization, and the scalar field optimization. (a) The image-based approach slightly changes the object's spatial relations due to the image overlaps and graph cut. (b) The vector field optimization can result in undesired distortion, e.g. the left shoulder and arterioles around the heart. (c) The two-step scalar field optimization provides a better control of the smoothness and exhibits good ray coherence.*

does not depict the true relations of the human body and the inner organs. As described in Section 4.4, direct vector field smoothing can easily result in distorted or deviated camera rays, which then reflect on a false camera projection, such as the left shoulder and arterioles around the heart in (b). The illustration in (c) is created by using our two-step scalar field optimization.

In our approach, the vector field is initialized from a set of Bézier curves, which smoothly connect the near clipping planes of each pair of successive cameras. It follows that if a smooth path can be traced between these planes, no intersection occurs as long as the directions of successive cameras do not differ by more than 90 degrees. This is not an actual limitation, since we can stack a number of cameras together to bend the rays in extreme ways. The final vector field is constructed from a weighted average of intermediate ray fields. But since the final streamlines are traced in this global vector field, the streamlines by definition should not intersect. However, in cases where the frusta of non-consecutive cameras intersect, critical points could appear, which may distort the image dramatically. One solution is to constrain the camera placement to avoid such cases.

6.1 Limitations

Although our approach can generate non-linear sampling rays for direct multiscale rendering simply based on a set of conventional pinhole cameras and image masks, the quality and beauty of the final image are highly dependent on users' efforts in selecting image masks and camera viewpoints. Here we provide four guidelines: 1) Objects projected in each view must have coherent spatial relationships to be able to create reasonable transitions. In other words, avoiding large difference in viewing directions between successive cameras is preferable. 2) Design of image masks directly affects relative positions of each view as well as available space for transitional regions. Even for the same set of camera views, resulting images can be quite different when different image masks are applied. As a result, the preserved regions in image masks should correspond to the relative positions of multiscale viewpoints. 3) Contradictory preserved views can result in undesirable images. For instance, rays in two preserved views that already intersect would cause unexpected problems while filling values in the scalar field. In such cases, users need to adjust the near-clipping and far-clipping distances of viewpoints to maintain the exclusivity of preserved views. 4) Since Bézier curves are constructed at each pixel position, rays would be pretty dense when passing through a small near-clipping plane of a camera and could increase the difficulty of accurately integrating streamlines in the vector field. So a larger near-clipping distance is usually better for vector field generation.

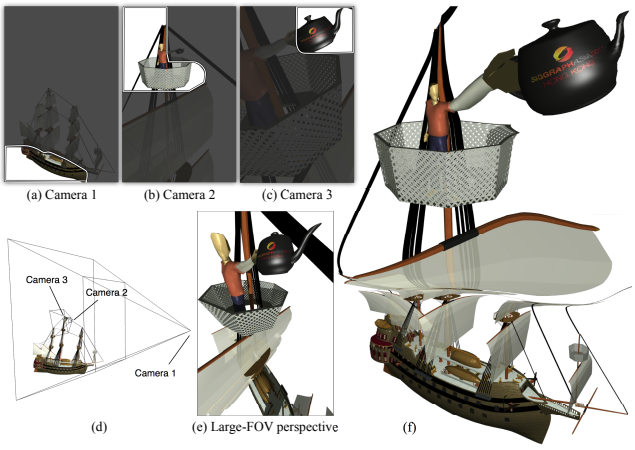


Figure 11: *Galleon dataset. (a)-(c) show three camera viewpoints and the corresponding masks at different scales, from an overview of a Spanish galleon to a teapot held by a pirate on the galleon. The brightened parts of the three images are the user-specified preserved regions, whereas the darkened regions are transition regions. (d) shows the relative positions of the three pinhole cameras and the galleon. The resulting multiscale image is shown in (f). (e) is a normal perspective view with a large field of view (FOV) which creates a fisheye-like effect. But even though the large FOV contains parts of the galleon body, the viewer can hardly tell the actual shape of the galleon due to the bad view angle.*

Adequate volume resolution for vector field calculation is also a crucial factor in producing high quality multiscale images using our method. Insufficient volume resolution would result in a lack of detail in streamlines at small scales, and noticeable errors in projecting preserved views. In Section 5 we mentioned the use of multiresolution volumes. However, discontinuous vector fields at junctions between different resolutions also introduce slight streamline perturbations, sometimes resulting in perceivable artifacts in projected images. In order to derive more coherent streamlines, higher-order vector sampling strategies at resolution junctions are needed to reduce streamline discontinuities.

Finally, our multiscale rendering framework supports both polygon models and volume datasets. However, most popular raytracers do not support non-linear ray tracing because ray-object intersection tests are much trickier for non-linear rays. Therefore, a custom raytracer is required to fully apply our framework to geometric scenes.

6.2 Applications

Presenting hierarchical structures. Our current design demonstrates attractive results and suggests several interesting uses. As mentioned earlier, multiscale images are especially useful in presenting objects with hierarchical structures. Showing continuous multiscale views in the same image helps viewers comprehend the spatial relationship between scales. Figure 4 illustrates three different scales: the upper body, the heart, and the aorta in a human body volume dataset, in three separate viewpoints. The brightened regions in the left three images are the corresponding preserved areas that are specified by the user in each view. Notice that the transfer functions applied to the dataset in each of the three views are

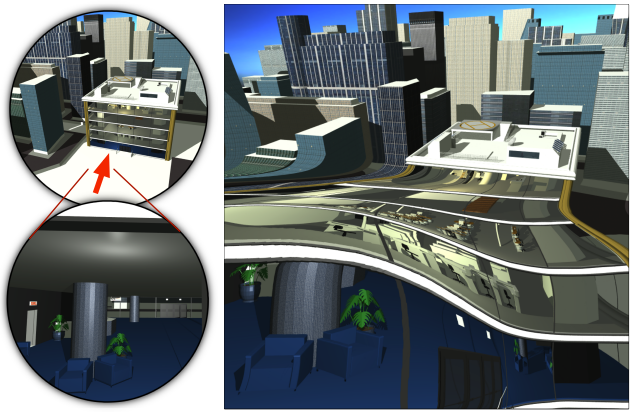


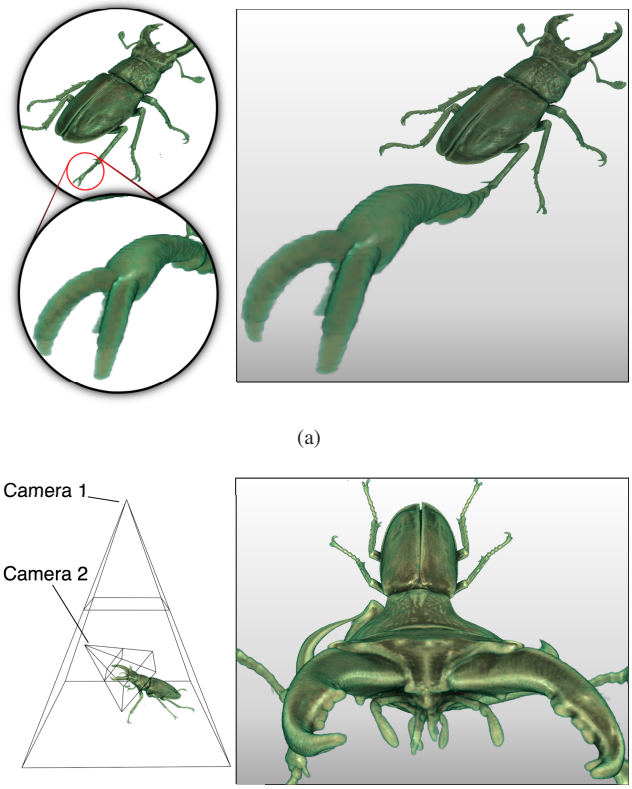
Figure 12: *Left: Two camera views that show a city skyline and a building interior, respectively. Right: our multiscale image of the same 3D city model. Inspired by M. C. Escher's Print Gallery [1956], our rendering mimics the multiscale nature of his drawing.*

slightly different to show interesting features more effectively. The resulting multiscale image is shown in Figure 1.

Focus+context effect. Most focus+context rendering algorithms redirect a portion of camera rays in order to achieve partial magnification. Since ray direction alterations in these models are based on single cameras, view directions of the magnified portions are basically the same as of the context. However, features at different scales usually require different viewpoints to best show interesting details. In our multiscale camera model, users can specify desired viewpoints for each scale. In addition, our model is also able to handle multilevel focus+context rendering, since users can set up more than two cameras, with each of them capturing different levels of detail. Creating multiple magnified regions is also possible, although our current rendering framework is designed for zooming along a single viewing path. We believe our ray generation approach can support multiple magnified regions by slightly modifying the way that pinhole cameras are connected so that camera rays can be cast differently in different image regions. But multiple magnified regions also imply more complex viewing frusta and a high potential for conflicting or intersecting viewing rays, and thus require a more thoughtful camera placement and mask design.

Figure 11 shows an example of focus+context rendering. In this 3D model, a pirate is standing in the crow's nest on top of the mast of the galleon holding a teapot. The top three images (a)-(c) in Figure 11 are three different camera viewpoints showing different levels of detail in the scene. The side view in Figure 11(d) provides an overview of the scene that depicts the relative positions and scales of the three cameras and the model. Based on the three camera views, the resulting image in Figure 11(f) clearly shows a continuous view from the whole galleon to the pirate on the mast and the teapot in his hand. Despite the significant difference in scales, our multiscale camera model makes it possible to show an overview of the galleon as well as a detailed view of the text on the teapot in a single continuous image. Figure 13(b) demonstrates focus+context rendering with greatly differing view angles between focus and context views. Camera 1 provides a bird's-eye view directly above the stag beetle, while Camera 2 shows a closeup view of the front of the beetle. The resulting image on the right combines both an overview of the beetle's body and a focused view of the beetle's oral cavity in a single, continuous image.

Artistic rendering. Multiscale rendering techniques may also be applied in many works of art in order to create novel views. In



(a)

(b)

Figure 13: *Stag beetle volume dataset. (a) The image on the right creates a focus+context effect, simultaneously and seamlessly showing both an overview of the stag beetle and a closeup view of one of its limbs. The multiscale view is created using the two camera viewpoints shown on the left. (b) Another focus+context view illustrates the body of the stag beetle from a bird's-eye view and zooms to its head in a front view.*

addition to showing different levels of detail, multiple scales in a single illustration help guide the viewer's eye towards interesting parts of the image. Many works by the mathematician and artist M. C. Escher are good examples of multiscale images. Figure 12 shows our rendering result using a 3D city model. In this figure, we try to mimic the multiscale nature of his well-known illustration *Print Gallery* [Escher 1956]. To generate this image, we combined views of the city skyline and the building interior to create a smooth, curved transition between indoor and outdoor areas. We placed image masks in the upper portion of the the first viewpoint (Figure 12, top left) to provide a perspective view of the surrounding buildings, and in the bottom left corner of the second viewpoint (Figure 12, bottom left) to capture interior details. Our framework then generates a smooth transition between the two viewpoints accordingly.

7 Conclusion and Future Works

We present a novel rendering framework model for creating continuous multiscale views of 3D geometric models and volumetric data. Almost all appealing multiscale illustrations are handcrafted by skilled artists. As we have shown, it is attractive to blend multiscale views of complex structures in a single static picture. Furthermore, even for a single object such as the stag beetle shown in Figure 13, multiscale views can be applied to achieve focus+context

rendering and to highlight interesting parts of the data. Finally, our multiscale rendering framework may also be used to emulate multiscale illustrations of objects and scenes that are previously only created by artists and physically impossible or expensive to create.

There are a number of directions for future work especially to increase the usability and robustness of this new rendering technology. First, the implementation of a custom raytracer would fully enable multiscale rendering of geometric scenes using our camera model. Automatic viewpoint placement is another interesting topic to explore. Instead of specifying viewpoints and corresponding image masks, users select features of interest with respect to properties such as shape, size, location, etc. With proper constraints, camera views could be automatically determined using the properties and spatial relations of user-specified features, without resulting in undesirable artifacts. Finally, it is also possible to create animated views by allowing selected view points to move.

Acknowledgements

This work was sponsored in part by the U.S. National Science Foundation through grant CCF 0811422, and the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777. Data courtesy of the National Institutes of Health, the Institute of Computer Graphics and Algorithms, the Vienna University of Technology, and 3D Warehouse of Google Inc.

References

BLOOM, F. E., NELSON, C. A., AND LAZERSON, A. 1988. *Brain, Mind, and Behavior*. W. H. Freeman and Company, 41 Madison Avenur, New York, NY 10010.

BÖTTGER, J., BALZER, M., AND DEUSSEN, O. 2006. Complex logarithmic views for small details in large contexts. *IEEE Transactions on Visualization and Computer Graphics 12*, 5, 845–852.

BROSZ, J., SAMAVATI, F. F., SHEELAGH, M. T. C., AND SOUSA, M. C. 2007. Single camera flexible projection. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, NPAR '07, 33–42.

BTTGER, J., PREISER, M., BALZER, M., AND DEUSSEN, O. 2008. Detail-in-context visualization for satellite imagery. *Computer Graphics Forum 27*, 2, 587–596.

CARPENDALE, M. S. T., CARPENDALE, T., COWPERTHWAITE, D. J., AND FRACCHIA, F. D. 1996. Distortion viewing techniques for 3-dimensional data. In *Proceedings of the 1996 IEEE Symposium on Information Visualization (INFOVIS '96)*, IEEE Computer Society, Washington, DC, USA, 46–53.

COSMIC VOYAGE, 1996. `http://www.imdb.com/title/tt0115952/`, National Air and Space Museum.

CUI, J., ROSEN, P., POPESCU, V., AND HOFFMANN, C. 2010. A curved ray camera for handling occlusions through continuous multiperspective visualization. *IEEE Transactions on Visualization and Computer Graphics 16* (November), 1235–1242.

ESCHER, M. C., 1956. Print gallery. `http://www.mcescher.com/Gallery/recogn-bmp/LW410.jpg`.

FAY, J. A. 1994. *Introduction to fluid mechanics*. MIT Press, Cambridge, MA.

GLASSNER, A. S. 2000. Cubism and cameras free-form optics for computer graphics. Tech. Rep. MSR-TR-2000-05, Microsoft.

GOOGLE, 2010. Google earth. `http://earth.google.com`.

HALL, C. A., AND PORSCHING, T. A. 1990. *Numerical Analysis of Partial Differential Equations*. Prentice Hall, Engelwood.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph. 22*, 3, 277–286.

MASHIO, K., YOSHIDA, K., TAKAHASHI, S., AND OKADA, M. 2010. Automatic blending of multiple perspective views for aesthetic composition. In *Proceedings of the 10th international conference on Smart graphics*, Springer-Verlag, Berlin, Heidelberg, SG'10, 220–231.

MCCRAE, J., MORDATCH, I., GLUECK, M., AND KHAN, A. 2009. Multiscale 3d navigation. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 7–14.

NYE, L., 2008. Zoom into the human bloodstream. Available at `http://www.nsf.gov/news/special_reports/scivis/winners_2008.jsp`.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Trans. Graph. 22*, 3, 313–318.

POPESCU, V., ROSEN, P., AND ADAMO-VILLANI, N. 2009. The graph camera. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, ACM, New York, NY, USA, 1–8.

SUDARSANAM, N., GRIMM, C., AND SINGH, K. 2008. Non-linear perspective widgets for creating multiple-view images. In *NPAR '08: Proceedings of the 6th international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, 69–77.

THE KNOWN UNIVERSE, 2009. `http://www.amnh.org/news/2009/12/the-known-universe/`, American Museum of Natural History.

WANG, L., ZHAO, Y., MUELLER, K., AND KAUFMAN, A. 2005. The magic volume lens: an interactive focus+context technique for volume rendering. 367–374.

WANG, Y.-S., LEE, T.-Y., AND TAI, C.-L. 2008. Focus+context visualization with distortion minimization. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2008) 14*, 6, 1731–1738.

WANG, Y.-S., WANG, C., LEE, T.-Y., AND MA, K.-L. 2011. Feature-preserving volume data reduction and focus+context visualization. *IEEE Transactions on Visualization and Computer Graphics 17*, 2, 171–181.

XU, L., LEE, T.-Y., AND SHEN, H.-W. 2010. An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics 16*, 1216–1224.

YE, X., KAO, D., AND PANG, A. 2005. Strategy for seeding 3d streamlines. *Visualization Conference, IEEE 0*, 60.

YU, J., AND MCMILLAN, L. 2004. A framework for multiperspective rendering. In *Rendering Techniques*, 61–68.

YU, J., AND MCMILLAN, L. 2004. General linear cameras. In *ECCV (2)*, 14–27.

YU, J., MCMILLAN, L., AND STURM, P. 2008. Multiperspective modeling, rendering, and imaging. In *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses*, ACM, New York, NY, USA, 1–36.