Spring 2015

Visibility computation through image generalization

Jian Cui
Purdue University

Graduate School Form 30
(Revised 08/14)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Jian Cui

Entitled

VISIBILITY COMPUTATION THROUGH IMAGE GENERALIZATION

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Voicu Popescu

Daniel Aliaga

Elisha Sacks

Christoph Hoffmann

Xavier Tricoche

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification/Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Voicu Popescu

Approved by: Sunil Prabhakar / William J. Gorman 03/12/2015

Head of the Department Graduate Program Date

VISIBILITY COMPUTATION THROUGH IMAGE GENERALIZATION

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Jian Cui

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2015

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

I would like to first acknowledge my major advisor Dr. Voicu Popescu. He introduced me to computer graphics and accompanied me along my learning journey providing patient guidance. I am grateful for the time and energy he devoted to me. I would also like to thank all professors from whom I have acquired invaluable knowledge and inspiration: Daniel Aliaga, Christoph Hoffmann, Elisha Sacks and Xavier Tricoche from Computer Graphics and Visualization Lab, Computer Science Department, Bedrich Benes and Nicoletta Adamo-Villani from Computer Graphics Technology Department; Lili Wang from Beihang University.

I would like to thank Paul Rosen and Zhiqiang Ma for their collaboration on research projects, and Ziang Ding, Meng-lin Wu, Liang Li and Zheng Zhou for brainstorming ideas. I would also like to thank all of my friends and colleagues throughout the Purdue Campus.

Finally, my deepest gratitude goes to my family for their love and support throughout these exciting, challenging, and successful years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABBREVIATIONS

ADI	Animated depth image
AGVS	Adaptive global visibility sampling
BRDF	Bidirectional reflectance distribution function
BSP	Binary space partitioning
CCD	Charge-coupled device
CoNUS	Coherent non-uniform sampling
COP	Center of projection
CRC	Curved ray camera
FPC	Flexible pinhole camera
FOV	Field of view
GC	Graph Camera
GLC	Genral linear camera
GPC	Genral pinhole camera
GPU	Graphics processing unit
GUI	Graphical user interface
GVS	Guided visibility sampling
IBR	Image based rendering
MCOP	Multicenter-of-projection
PPC	Planar pinhole camera
PVS	Potentially visible set
ROI	Region of interest

ABSTRACT

Cui, Jian PhD, Purdue University, May 2015. Visibility Computation through Image Generalization. Major Professor: Voicu Popescu.

This dissertation introduces the image generalization paradigm for computing visibility. The paradigm is based on the observation that an image is a powerful tool for computing visibility. An image can be rendered efficiently with the support of graphics hardware and each of the millions of pixels in the image reports a visible geometric primitive. However, the visibility solution computed by a conventional image is far from complete. A conventional image has a uniform sampling rate which can miss visible geometric primitives with a small screen footprint. A conventional image can only find geometric primitives to which there is direct line of sight from the center of projection (i.e. the eye) of the image; therefore, a conventional image cannot compute the set of geometric primitives that become visible as the viewpoint translates, or as time changes in a dynamic dataset. Finally, like any sample-based representation, a conventional image can only confirm that a geometric primitive is visible, but it cannot confirm that a geometric primitive is hidden, as that would require an infinite number of samples to confirm that the primitive is hidden at all of its points.

The image generalization paradigm overcomes the visibility computation limitations of conventional images. The paradigm has three elements. (1) Sampling pattern generalization entails adding sampling locations to the image plane where needed to find visible geometric primitives with a small footprint. (2) Visibility sample generalization entails replacing the conventional scalar visibility sample with a higher dimensional sample that records all geometric primitives visible at a sampling location as the viewpoint translates or as time changes in a dynamic dataset; the higher-dimensional visibility sample is computed exactly, by solving visibility event equations, and not through sampling. Another form of visibility sample generalization is to enhance a sample with its trajectory as the geometric primitive it samples moves in a dynamic dataset. (3) Ray geometry generalization redefines a camera ray as the set of 3D points that project at a given image location; this generalization supports

rays that are not straight lines, and enables designing cameras with non-linear rays that circumvent occluders to gather samples not visible from a reference viewpoint.

The image generalization paradigm has been used to develop visibility algorithms for a variety of datasets, of visibility parameter domains, and of performance-accuracy tradeoff requirements. These include an aggressive from-point visibility algorithm that guarantees finding all geometric primitives with a visible fragment, no matter how small primitives image footprint, an efficient and robust exact from-point visibility algorithm that iterates between a sample-based and a continuous visibility analysis of the image plane to quickly converge to the exact solution, a from-rectangle visibility algorithm that uses 2D visibility samples to compute a visible set that is exact under viewpoint translation, a flexible pinhole camera that enables local modulations of the sampling rate over the image plane according to an input importance map, an animated depth image that not only stores color and depth per pixel but also a compact representation of pixel sample trajectories, and a curved ray camera that integrates seamlessly multiple viewpoints into a multiperspective image without the viewpoint transition distortion artifacts of prior art methods.

1 INTRODUCTION

The visibility problem has been studied since the very early days of computer graphics, yet visibility remains an important an active research area. The computation of the set of geometric primitives that are visible from a viewpoint or a view region is central to many computer graphics, visualization, and computer vision applications. At first, due to computing power and storage limitations, visibility computation was limited to finding the geometric primitives visible from a given viewpoint, in order to render the geometric dataset from that viewpoint with correct hidden-surface removal. As graphics algorithms and their hardware implementation have advanced, research has begun to focus on visibility computation in order to achieve effects such as soft shadows, reflections and motion blur. Soft shadows can be rendered efficiently if one can quickly determine the geometric primitives to which there is direct line of sight from a point within an area light source. Reflections can be rendered efficiently if one can quickly compute the set of geometric primitives that can be reached by reflected rays. Motion blur requires finding all geometric primitives that are visible at an output image pixel over the exposure time interval of the virtual camera used render the scene.

As the size of computed and acquired data grows exponentially, solving visibility becomes both more challenging and more important. Technology advances enable the simulation and acquisition of complex phenomena with ever increasing accuracy. The resulting data volume increase surpasses our ability to analyze, transmit, and visualize data. Fortunately, many of the data queries benefit from data locality, which means that they can be answered by consulting only a small fraction of the entire dataset. In many instances, the visible set is several orders of magnitude smaller than the entire dataset. The reduced size of the visible set makes data analysis, transmission, and visualization tractable.

Another important motivation for studying the visibility problem is the proliferation of thin Internet-connected computing platforms, such as laptops, tablets, and smart phones, which have become the platform of choice for many applications. The platform does not have the storage and processing capabilities to handle a large

dataset and remote visualization is needed. One remote visualization approach is to render each frame at the server and to transfer the frame to the client where it is displayed. However, such an approach incurs the delay of traversing the network twice, once to communicate to the server the parameters of the desired frame, and once to receive the frame. The resulting latency limits the interactivity of the visualization. The network latency can be avoided if one computes and transfers a visible set sufficient to reconstruct visualization frames for a range of view parameters. Each frame is rendered at the client, from the visible set, with good interactivity.

Although the visibility problem can be solved, in theory, with computational geometry approaches such as the Aspect Graph, the $O(n^9)$ complexity for general scenes makes these solutions impractical for todays datasets which can contain billions or even trillions of geometric primitives. An image is a powerful way of approximating visibility. One can render an image quickly with the help of massively parallel GPUs, and each image pixel reports a visible geometric primitive. However, the visibility computation power of a conventional image is limited by the images uniform sampling rate, single viewpoint, and single time point.

We eliminate the visibility computation limitations of conventional images through image generalization. We have used the image generalization visibility paradigm to develop fast quality-guaranteed approximate visibility algorithms and efficient exact visibility algorithms (chapter 2), images with pixels that do not only store a color and geometry sample at each pixel but also the samples trajectory (chapter 3), camera models that allow modulating the sampling rate of the image to assign more pixels to data subsets with higher complexity or importance (chapter 4), and camera models with curved rays that circumvent occluders (chapter 5).

In this section, we first introduce the visibility problem and its importance (section 1.1), we then present a taxonomy of visibility problems and solutions (section 1.2), we describe the connection between visibility and images (section 1.3), we introduce the image generalization visibility paradigm (section 1.4), we state the dissertation that anchored this research (section 1.5), and we give a preview of our results (section 1.6).

1.1 Visibility problem and importance

The need to answer the question "what is visible?" arose for the first time in computer graphics in the context of rendering an image of a scene with correct visibility sorting of opaque surfaces. Due to the limited computation power, early rendering of polygons was usually done in wire frame. Without visibility culling, it is difficult to recognize objects and their spatial relationship in wireframe images. Hidden line removal algorithms solved this problem by not drawing the invisible line segments, thus wire frame objects appear to be solid in the output image. The hidden line removal algorithms were then extended to hidden surface removal algorithms for filled surfaces. When two or more surfaces project to the same image region, one needs to arbitrate between these surfaces and to give preference to the nearest, or visible, surface, over the farther, or invisible, surfaces. Hidden surface removal algorithm compute which surfaces and which surface parts should not be drawn to obtain a complete and correct output image. In addition to correct visibility, hidden surface removal algorithms also bring the efficiency of not drawing hidden surfaces. Once hardware performance has improved, painters style visibility algorithms have become prevalent. Such algorithms compute the order in which primitives should be drawn such that the visible surface is always drawn last, resulting in correct visibility, without the challenge of segmenting partially visible surfaces.

After a few more iterations of Moore's Law, a simple visibility algorithm became practical: the *z-buffer*. In addition to color, each pixel now also has a depth channel, which records the distance to the nearest surface encountered so far at that pixel as the current frame is rendered. A pixel is written only if the current triangle is closer at that pixel than the depth recorded in the z-buffer. Output image visibility is correct no matter in what order the surfaces are considered. The simplicity and versatility of z-buffering made hidden surface removal and painters style visibility obsolete.

Z-buffering provides pixel-level visibility sorting of the primitives that project inside the image frame, but in the case of large datasets, many primitives project outside the image, and eliminating these primitives one at a time can be too slow. Frustum culling is a type of visibility algorithm that discards a group of primitives if the projection of a conservative bounding box of the group does not intersect the image. For datasets with high complexity, performance might not be adequate even if one only draws the primitives that project inside the frame. Occlusion culling

algorithms discard groups of geometric primitives that project inside the output image frame but that are hidden by primitives that are closer to the output image viewpoint.

The visibility algorithms discussed so far consider only a single output image. However, in many computer graphics applications, view parameters change slowly. For example, when exploring a dataset, the viewpoint changes on a continuous curve as the virtual camera is translated through the dataset. The view parameter coherence is exploited by potentially visible set (PVS) algorithms, which attempt to find a set of geometric primitives that contains all the geometric primitives visible over a range of view parameters. If the camera translates from a point A to a point B, a PVS should contain all geometric primitives that are visible from an intermediate viewpoint on segment AB. In many cases the sets of geometric primitives visible from A and from B exhibit significant redundancy, the PVS for AB is only slightly larger than the union of the visible sets from A and B, and the PVS for AB is much smaller than the entire dataset. Consequently, rendering the frames for the intermediate viewpoints from the PVS brings efficiency comparable to rendering only the visible primitives, but without requiring that the visible set be computed for each frame. However, given a dataset, computing the PVS corresponding to a range of view parameters is a challenging open research question.

So far, the discussion has focused on computing visibility along rays that emanate from the output image viewpoint. For many computer graphics applications, visibility has to be evaluated for other types of rays. In the case of shadow rendering, one has to estimate whether an output image surface sample is visible from a light source, i.e. whether a light ray reaches that sample. Shadow rendering is simpler when the light source can be approximated with a point, and the resulting shadows are hard, i.e. with an abrupt transition from light to shadow. Interactive graphics applications approximate hard shadows by rendering the dataset from the light point. The resulting z-buffer is called a shadow map. An output image sample is reprojected onto the shadow map and it is shaded as in shadow if it is farther from the light than the shadow map sample at the reprojection point. However, the quality of the rendered shadows is limited by the resolution of the shadow map, which can miss thin features, and by the fact that the shadow map doesnt estimate visibility along output image sample light rays, but rather along light rays defined by the regular grid of the shadow map.

When the light source cannot be approximated by a light point, shadow computation becomes more challenging. Some output image samples see a fraction of the light source area, and the transition from light to shadow is gradual, over a penumbra region. Rendering soft shadows accurately and efficiently requires finding the geometric primitives that are visible from anywhere on the light source, a problem similar to PVS computation.

Rendering specular reflections accurately requires finding the first surface intersected by each reflected ray. Whereas output viewpoint and light rays are highly coherent, reflected rays typically exhibit great direction and sampling rate variability, as the incident rays are perturbed by the reflective surfaces. Whereas light rays can be approximated with conventional cameras, computing reflections accurately requires tracing individual reflected rays, a process that can be accelerated by computing the set of geometric primitives visible along the reflected rays.

The visibility problem is also central to the field of image based rendering (IBR). Image-based rendering appeared in the early 1990s as an approach for simplifying photo-realistic rendering. Unlike traditional computer graphics, which uses 3D geometric primitives as input, IBR uses pre-acquired or pre-computed images as rendering primitives. Rendering from reference images has recently received renewed attention in the context of remote visualization. A fundamental challenge in IBR is to decide which reference images are necessary and sufficient to render a given output image. In other words, one has to find the reference images that contain the samples visible in the output image. The IBR visibility problem computes a set of reference image samples and not a set of visible primitives. The visible samples are computed using a heuristic set of redundant images acquired from nearby viewpoints, using a layered depth image that eliminates the redundancy during a pre-processing step, or using a 4D database of rays called a light field.

Visibility has applications beyond traditional computer graphics. In computer vision, visibility computation aids in sensor placement adequate scene sampling; in robotics, visibility aids motion planning; in simulation, visibility accelerates wave propagation computation.

In conclusion, visibility computation is an open research problem at the core of many applications in graphics and beyond.

1.2 Visibility algorithm taxonomy

Visibility algorithms are classified based on the type of visibility problem they solve, based on the type of the visibility solution they compute, and based on the approach they take.

1.2.1 Visibility algorithm classification based on problem type

We distinguish between the following types of visibility problems:

- Visibility along a line
- Visibility from a viewpoint
- Visibility from a view segment
- Visibility from a view polygon
- Visibility from a view region
- Dynamic visibility

The *visibility along a line* problem has two variants: ray shooting and point-to-point visibility. In ray shooting, a ray is defined with an origin point and a direction and the goal is to find the first geometric primitive intersected by the ray. In point-to-point visibility the goal is to determine whether a segment defined by two points intersects a geometric primitive or not, i.e. whether there is a direct line of sight between the two points. Visibility along a line serves as a building block for more complex visibility problems.

Visibility from a viewpoint is a 2D set of ray shooting visibility problems. The most frequently used parameterization of the 2D set of rays is that defined by the planar pinhole camera model (i.e. the conventional perspective projection model). A ray is defined by the viewpoint and a point on the image plane. Since the viewpoint is constant, a ray is defined by two parametersthe two image plane coordinates. Typical uses of from viewpoint visibility include output image visibility computation and hard shadow computation.

Visibility from a view segment adds one viewpoint translation as the third dimension of the set of rays along which visibility is probed. A ray is defined with three

parameters: the parameter that defines the viewpoint location along the view segment, and the two image plane coordinates. From view segment visibility is useful, for example, in motion blur computation.

Visibility from a view polygon adds a second viewpoint translation as the fourth dimension of the set of rays along which visibility is probed. A ray is defined with four parameters: two parameters that define the viewpoint inside the view polygon, and the two image plane coordinates. If the polygon is a triangle, the two viewpoint parameters are typically the barycentric coordinates of the viewpoint inside the triangle. If the polygon is a rectangle, the two viewpoint parameters are the 2D coordinates that define the viewpoint inside the rectangle. From view polygon visibility is useful, for example, in the case of soft shadow computation where one has to find all geometric primitives visible from a rectangular area light source. Another application of from view polygon visibility is in the context of remote visualization, where it allows the user to translate the viewpoint at the client anywhere within a view rectangle. The current image is rendered efficiently from the from view polygon visible set, which contains all geometric primitives visible from anywhere inside the view polygon.

Visibility from a view region adds a third viewpoint translation as the fifth dimension of the set of rays along which visibility is probed. A ray is defined with five parameters: three parameters to define the viewpoint within the region, and the two image plane coordinates. The region is typically a box defined by the ranges of the three orthogonal translations of the viewpoint. Complex view regions are approximated with a bounding box. From a view box visibility is typically reduced to solving from-rectangle visibility for the six faces of the box and unioning the six visible sets with the geometric primitives inside the box. As a result, in practice, from view region visibility is only a 4D visibility problem. From-region visibility allows computing visibility in complex datasets by partitioning the viewing space into box-like cells.

Dynamic visibility is the problem of computing visibility in a dynamic dataset. The dataset time parameter adds another dimension to the visibility problem. Visibility along a given ray becomes 1D; as time changes and the geometric primitives of the dynamic dataset move, a ray can intersect multiple geometric primitives. From-point visibility becomes 3D. From-region dynamic visibility is a 5D visibility problem, using again the reduction of the view box to the union of the visible sets of the box faces.

A common prior work approach is to approximate higher-order visibility problems by sampling the high-dimensional space of visibility parameters and by unioning the visible sets computed at the sampling points. For example, from view box dynamic visibility can be approximated by computing static from-point visibility at each box corner, for several time steps, and by unioning the visible sets. Such an approach requires sampling the high-dimensional space of visibility parameters densely, and, even so, the approach remains heuristic, with no quality guarantee for the approximation computed.

The image generalization paradigm advocates computing visibility with an image and not with individual rays. The coherency of the rays in an image allows estimating visibility efficiently by projection followed by rasterization, which leads to a low per-ray amortized cost. We have demonstrated the power of the image generalization visibility paradigm by developing several algorithms for from-point, from-segment, from-polygon, and from-region visibility, for static and for dynamic datasets.

1.2.2 Visibility algorithm classification based on solution type

Visibility algorithms are also classified based on whether and how the visible set is approximated:

- Exact
- Aggressive
- Approximate
- Conservative

Exact visibility algorithms compute visible sets that contain all and only visible geometric primitives. Given a high dimensional input domain D of visibility parameter values, the exact set contains all dataset geometric primitives p for which there is a point d in D such that p is visible from d. For example, in the case of static from view segment visibility, the exact visible set contains a geometric primitive if and only if it is visible from somewhere on the view segment. The challenges of exact visibility algorithms are high computational complexity and lack of numerical robustness. We have used the image generalization paradigm to develop an efficient and robust from-point exact visibility algorithm. We have also developed from view segment and from

view polygon visibility algorithms that are exact under view translation. A visibility algorithm is exact under view translation if it finds all geometric primitives that are visible as the view translates.

Aggressive visibility algorithms underestimate the visible set, in the interest of simplicity and of performance. The aggressive visible set contains only but not all visible geometric primitives. Aggressive algorithms sample the input domain of visibility parameters and accumulate the visible geometric primitives they found. Prior art aggressive visibility algorithms sample visibility heuristically, which can lead to visible sets that are far from complete. The missing visible geometric primitives translate to objectionable artifacts when the output image is rendered from the aggressive set. We have developed aggressive visibility algorithms that provide a quality guarantee for the visible set computed. The quality guarantee results in output images that are virtually indistinguishable from images that are rendered from the original dataset.

Approximate visibility algorithms both overestimate and underestimate the set of visible geometric primitives. The visible set contains some geometric primitives that are never visible from the input domain of visibility parameter values, and it misses some that are visible. Image-based rendering visibility algorithms fall in this category. The visible samples computed are just a sampling of the set of visible geometric primitives. The sampling of the visible set is desired when the number of visible geometric primitives is itself too large for adequate application performance. Consider the case of a curved surface tessellated with 10,000 triangles. When the surface is seen from a distance, approximating the surface with a few samples (e.g. 10) is acceptable and preferred to finding all 10,000 visible triangles. IBR visibility algorithms not only solve visibility, but they also adapt the geometric level of detail to bound rendering cost, a problem that is notoriously difficult to solve with computational geometry approaches. We have used the image generalization paradigm to develop approximate visibility algorithms for static and dynamic datasets that bound the number of samples in the visible set.

Conservative visibility algorithms simplify visible set computation at the cost of including in the visible set some geometric primitives that are not visible from the input domain of visibility parameters. All visibility approximations are conservative in the sense that the approximations cannot lead to missing visible primitives. Conservative visibility algorithms usually compute an aggressive visible set and then discard

the primitives hidden by the aggressive set. The challenge is to avoid including too many hidden primitives in the visible set. The factor by which conservative algorithms overestimate the visible set is not bounded, so it can be arbitrarily large. Our aggressive visibility algorithms with quality guarantees enable conservative algorithms with small overestimation factors.

1.2.3 Visibility algorithm classification based on approach

Based on how visibility is analyzed over the input domain of visibility parameters, visibility algorithms can be categorized as:

- Continuous
- Sample-based

*Continuous*visibility algorithms compute a complete and accurate visibility subdivision over the input domain D of visibility parameters (e.g. image plane coordinates, viewpoint translation, dynamic dataset time). A single geometric primitive is visible at each visibility subdivision cell. The exact visible set is computed as the union of geometric primitives visible at the visibility subdivision cells. For example, in the case of from-point visibility of a dataset modeled with triangles, the visibility parameters are the two image coordinates. The visibility subdivision is a polygonal subdivision of the image plane. A cell is a polygonal region where a single triangle is visible. The exact visible set is defined by all the triangles visible in the polygonal regions of the visibility subdivision. In the case of from view segment visibility of a triangle dataset, each triangle is composed with the viewpoint translation vector to generate a 3D prism through extrusion. The visibility subdivision is the intersection of the extruded 3D prisms. The composition of the geometric primitive with the visibility parameters leads to high dimensional spaces where no algorithm for computing the visibility subdivision has proven to be practical. Another approach is to solve directly for the visibility subdivision boundaries, which have lower dimensionality. For example, the intersections of the boundaries of the 3D prisms in from view segment visibility are lines. The visibility event equations that describe the visibility subdivision boundaries have high degree. An additional challenge is the difficulty to account for all degeneracies and to achieve robustness.

Sample-based visibility algorithms sample the input domain of visibility parameters with visibility rays. Some approaches use individual rays, and some approaches use rays grouped in images. Approaches that use individual rays start out with a set of seed rays and shoot additional rays based on the results of the visibility queries gathered by the seed rays. The decision of whether and which additional rays to shoot is heuristic. Approaches that use images typically render images from a regular grid of viewpoints. The advantage of images is that the amortized cost of a single ray is much smaller than in the case of shooting individual rays. The disadvantage of images is that images rendered from nearby viewpoints are redundant. A sample-based visibility algorithm cannot promise an exact or even a conservative visible set. Whereas a ray confirms that the first geometric primitive it intersects is visible, it would take an infinite number of rays to verify that a geometric primitive is hidden. In the case of a triangle, a single ray can reveal that the triangle is visible, but it would require shooting a ray through each triangle point to confirm that the triangle is hidden.

Most of the visibility algorithms developed under the image generalization paradigm fall in the sample-based category. Unlike prior-art sample-based visibility algorithms, our algorithms sample the space of visibility parameters deterministically and not heuristically, which provides a quality guarantee. We have also developed a visibility algorithm that combines the rigor of continuous visibility analysis with the efficiency of sample-based visibility analysis. Our hybrid visibility algorithm constructs an initial visibility subdivision starting from a high-quality aggressive visible set provided by our sample-based algorithm. The initial visibility subdivision is used to find most hidden triangles and to suggest visibility rays for the sample-based algorithm to probe. Some of the rays find additional visible triangles, and the visibility subdivision is extended. The hybrid algorithm proceeds recursively, alternating between a continuous and sample-based stage. The visible set converges quickly to the exact set.

1.3 Image as a visibility solution

Modern computer graphics applications let the user explore a 3D dataset by computing raster images of the dataset. A raster image is a regular 2D array of color values, or pixels, that are mapped to the display pixels. The image is computed using a camera model whose rays sample the dataset. Most applications use the planar

pinhole camera model which approximates the human eye well, producing images that resemble what the user would see if the user were actually immersed into the dataset. Each pixel defines a ray from the pinhole, or eye, to the pixel center. The rendering algorithm computes a pixel by computing the color where the ray intersects the geometric primitives of the dataset.

There are two fundamental approaches for rendering an image of a 3D dataset. The feed-forward approach processes each geometric primitive in the dataset by projection followed by rasterization. Projection determines the pixels affected by the primitive, and rasterization computes the color of the pixels affected by the primitive. The second fundamental approach for rendering an image of a 3D dataset is ray tracing. Ray tracing computes one image pixel at the time by computing the intersection between the pixel ray and the geometric primitives of the dataset.

Ray tracing has the advantage of versatility. Whereas one can render opaque and diffuse surfaces by only taking into consideration the first order rays that leave the eye, effects such as refraction and reflection require considering higher order rays that appear when the first order rays intersect transparent and reflective surfaces. The ray tracing procedure can be applied iteratively or recursively to compute the color samples collected by higher order rays. However, higher-order rays do not exhibit the same amount of coherence as first-order rays, and there generally is no closed-form image plane projection of geometric primitives that are sampled by higher order rays. Consequently, the feed-forward approach cannot accurately render effects that generate higher-order rays. Consider a chrome teapot which defines reflected, second-order rays at the image pixels covered by the teapot. One cannot render the specular reflections on the teapot by projection followed by rasterization, because one cannot project the reflected geometric primitives directly onto the image plane. Feed-forward rendering approximates specular reflections with techniques such as environment mapping.

Feed-forward rendering has the advantage of efficiency: the projection stage quickly determines which pixels might be affected by a given geometric primitive, and avoids considering pixel-geometric primitive pairs that do not yield any intersection. A nave ray tracer considers all pixel-geometric primitive pairs, which is inefficient. Ray tracing acceleration schemes focus on limiting the data subset that is considered for each ray. However, ray tracing acceleration has proven to be challenging, and most com-

puter graphics applications where performance is at a premium rely on feed-forward rendering with hardware support.

1.3.1 The visibility computation potential of images

No matter how it is computed, an image has to record the color of the closest geometric primitive intersected by the ray of each pixel. In other words, rendering solves a visibility problem along the line of the ray of each pixel. Therefore, visibility computation is integral to image rendering, and an image is a solution to a visibility problem. The visible set computed by an image can be recovered in one of two ways: as a set of visible samples, or as a set of visible geometric primitives.

The set of visible samples is obtained directly from the image without any modification of the procedure by which the image is rendered. As discussed earlier, z-buffering is now the method of choice for solving visibility for rendering. The pixel sample stores not only the red, green, and blue color values, but also a depth value that indicates how far along the pixel ray the closest geometric primitive is located. The depth value together with the pixel ray provided by the camera model defines a visible sample, i.e. a 3D point on a geometric primitive that is known to be visible from the viewpoint of the image. The samples can be reprojected to novel viewpoints to render new images without incurring the cost of processing the original dataset in its entirety.

One can use an image to recover a set of visible geometric primitives by assigning a unique identifier to each primitive (e.g. the index in the array of primitives), and by keeping track of the identifier of the geometric primitive sampled at each pixel. The identifier is stored in an additional pixel channel and it is updated each time the z-value of the pixel is updated. Once the image is fully rendered, each pixel reports the identifier of a visible primitive.

We clarify here that the visibility information of an image has to be collected before antialiasing. To reduce the artifacts stemming from the non-zero pixel size, high-quality rendering computes first an intermediate image with multiple color samples per pixel. Then the multiple color samples of a pixel are blended to obtain the final color value at the pixel. Whereas blending colors alleviates aliasing, one cannot blend 3D points or geometric primitive identifiers. The visibility information has to be collected by considering all samples of a pixel, before they are blended.

The visibility paradigm presented in this dissertation capitalizes on the great visibility computation potential of images. We adopt the feed-forward approach of rendering images to benefit from the efficiency of the approach. We only consider first-order rays, that is we only attempt to find the geometric primitives that are directly visible, and we do not attempt to find the geometric primitives that are visible indirectly, for example by reflection in a specular scene surface. Although our visibility algorithms do not consider simultaneously first and higher order rays, our algorithms can be used to compute visibility for same-order bundles of rays. For example, our from view polygon visibility algorithm can be used to render efficiently the soft shadows cast by a rectangular area light source. The reflections of a diffuse object can be computed efficiently if the diffuse object is approximated with an image rendered with one of our visibility-optimized camera models.

A conventional image can quickly find visible geometric primitives in large datasets. However, the visibility solution computed by a conventional image is far from complete.

1.3.2 Visibility computation limitations of conventional images

When used to compute visibility, a conventional image suffers from four fundamental limitations: the field of view limitation, the uniform sampling rate limitation, the sample-based limitation, the single viewpoint/time point limitation.

The *field of view limitation* stems from the fact that conventional images are rendered with planar pinhole cameras whose field of view cannot exceed 180 degrees. The field of view limitation of conventional cameras has been encountered in computer graphics in other contexts, including in the context of building an omnidirectional panorama, or map, to approximate the environment of a 3D scene. For example, distant geometry in an outdoor scene (e.g. mountains, clouds) are approximated with a panorama which renders the environment correctly without incurring the cost of modeling and rendering the environment geometry. Initially, environment maps were built using a spherical parameterization of the 2D space of rays through a point. Spherical environment maps were later replaced by cube maps. A cube map is the equivalent of six images, one for each of the faces of a cube. The images are rendered with six planar pinhole cameras with 90 degree by 90 degree field of view, with the center of the cube as their eye, and with the cube faces as their image planes. Cube

maps are an effective solution to the field of view limitation of conventional images which we adopt.

The *uniform sampling rate limitation* stems from the fact that conventional images are rendered with a uniform grid, oblivious to the fact that geometric primitives have image footprints of different sizes. Consider a 3D dataset modeled with triangles. First, triangles can have an arbitrarily small image footprint due to a high dataset complexity, to a large distance to the eye, or to a grazing viewing angle. In Figure 1.1 (top), the visible set found by the reference image of the finely tessellated sphere is incomplete, which results in severe artifacts when the set is used to render an image from the same viewpoint but with a slightly different view direction. Increasing the resolution of the reference image is only palliative: the image footprint of a visible triangle can be arbitrarily small; therefore an infinite resolution would be needed to guarantee that all visible triangles are found.

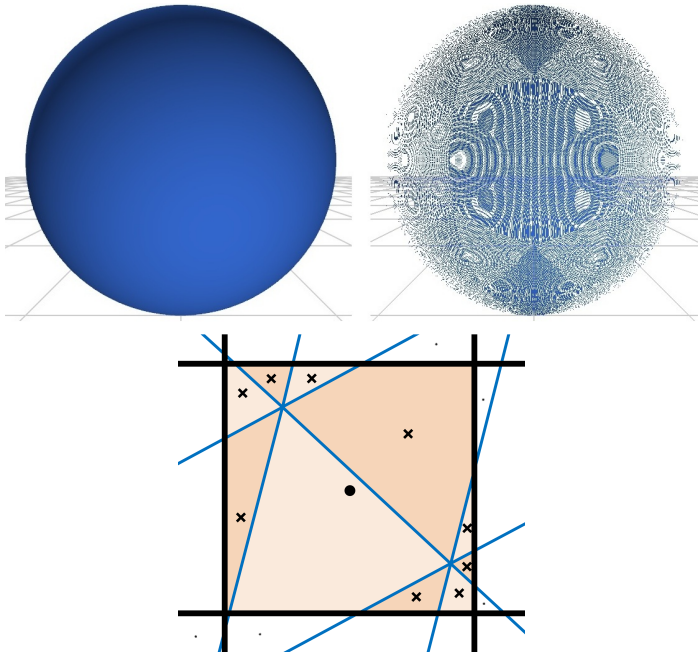

Figure 1.1. **Top**: reference image and frame rendered from the incomplete set of visible triangles found by the reference image. **Bottom**: sampling locations (crosses) added to a pixel to sample all the triangle fragments at the pixel.

The *sample-based limitation* is the fundamental limitation of sample-based approaches to visibility of not being able to verify that a geometric primitive is com-

pletely hidden by other geometric primitives. Sampling visibility with a ray might reveal that the geometric primitive is hidden along that ray, but an infinite number of samples is needed to confirm that the geometric primitive is hidden throughout.

The *single viewpoint/time point limitation* of conventional images when it comes to computing visibility is that a conventional image only finds visible geometric primitives from a single viewpoint and at a single time point. An image provides a visibility snapshot, whereas many applications would benefit from knowing which geometric primitives are visible over a range of viewpoint translations and over a time interval. Consider a remote visualization application that aims to let the user explore a massive dynamic dataset remotely. Given a box and a time interval, a visibility algorithm is asked to find all the geometric primitives that are needed to render the dataset from any viewpoint inside the box and at any time point inside the interval. The visible set is transferred to the client which can sustain an interactive visualization of the dynamic dataset from inside the box and overt the time interval. The user can decide to translate the virtual viewpoint inside the box for a given time point, or let time advance for a given viewpoint, or even translate the viewpoint to examine the dynamic dataset as time advances.

1.4 Image generalization paradigm for computing visibility

We propose to generalize images to overcome their visibility computation limitations described above. Our image generalization paradigm has three elements: *sampling pattern generalization, visibility sample generalization, and ray geometry generalization.*

1.4.1 Sampling pattern generalization

The first element of the image generalization paradigm is to abandon the uniform sampling of the image plane used by conventional images and to add sampling locations to the image plane where needed in order to improve the quality of the visibility solution computed. One option is to add sampling locations at image plane regions where the data subsets of higher complexity project. Another option is to add sampling locations such as to find all geometric primitives with a completely visible fragment. A fragment is the intersection between an image pixel and the image plane

projection of the geometric primitive. In other words, the fragment is the part of the pixel square covered by the geometric primitive. Sampling locations are added to make sure that all fragments are sampled, which guarantees finding all geometric primitives with a completely visible fragment, and which in turn guarantees finding all geometric primitives of a front surface, no matter how small their footprint. For our sphere example, sampling all fragments finds all visible triangles at the cost of one sampling location per fragment (Figure 1.1, buttom). Sampling locations are not added heuristically but rather deterministically, based on the dataset geometric primitives and based on the pixel grid. This sampling pattern generalization does not guarantee finding all visible geometric primitives, as some visible primitives that do not have a completely visible fragment are missed. However, the guarantee of finding all geometric primitives of a front facing surface is a strong quality guarantee which in practice results in aggressive visible sets that are close to complete.

1.4.2 Visibility sample generalization

A conventional image sample only stores a scalar visibility value, corresponding to a single visible geometric primitive. When the viewpoint translates or when time changes in the case of a dynamic dataset, the geometric primitive that is visible at an image plane sampling location can change. We propose to generalize the visibility sample to enable visibility computation in the context of dynamic datasets and of viewpoint translation. Visibility sample generalization proceeds in one of two directions. One direction is to increase the dimensionality of the visibility sample. A second direction is to enhance the sample with a record of its trajectory in the dynamic dataset. *Increasing the dimensionality of the visibility sample* brings two benefits: it enables visibility computation in the context of dynamic datasets and viewpoint translation, and it enables exact visibility computation. Consider the case of a dynamic dataset modeled with triangles rendered from a fixed view. As triangles move over time, multiple triangles can become visible at a given sampling location. What is needed is a 1D visibility sample that records all triangles visible at the sampling location over the time interval. The 1D visibility sample proposed by our image generalization paradigm is *not* a uniform 1D array of conventional samples. Instead, the 1D visibility sample is a subdivision of the time interval into subintervals such that a single triangle is visible for each subinterval. The visibility data stored by the

1D sample is non-redundant and complete, and the set of visible triangles at that sampling location is exact. Consider now the case of from view rectangle visibility of a static dataset modeled with triangles. A 2D visibility sample is needed to account for the triangles visible as the viewpoint translates with two degrees of freedom within the view rectangle. The 2D visibility sample stores a polygonal subdivision of the 2D viewpoint translation space. Each subdivision region corresponds to the viewpoint positions where a single triangle is visible. The 2D visibility sample stores all the triangles visible at a given sampling location as the viewpoint translates. Finally, consider the case of from viewpoint visibility of a static dataset modeled with triangles. A single 2D visibility sample is sufficient to compute an exact visibility solution. Like before, the visibility sample is a polygonal subdivision of a 2D space of visibility parameters. Whereas before the visibility parameters were the two viewpoint translations, now the visibility parameters are the two output image coordinates.

The second direction for visibility sample generalization is to *enhance the sample with its trajectory* as the geometric primitive it samples moves in the dynamic dataset. The geometric primitive carries the sample as the primitive moves over time. Instead of recording which geometric primitives are visible at a given sampling location over time, this visibility sample generalization records where each sample moves over time. The advantage of the dimensionality generalization of the visibility sample is a high-quality of the visibility solution computed. The advantage of the trajectory generalization of the visibility sample is lower redundancy of the visibility solution: consider a visible triangle moving with a constant velocity vector; dimensionality generalization will record the triangle as visible at all the pixels the triangle touches as it moves; trajectory generalization will record each triangle sample once along with a line segment to indicate the linear motion.

1.4.3 Ray geometry generalization

A conventional image is rendered with a planar pinhole camera whose rays connect the viewpoint to the pixel centers. Therefore, a conventional image can only find scene surfaces to which there is a direct line of sight from the viewpoint. We abandon the restriction that camera rays be straight lines. Generalizing the ray geometry to allow a ray to be any continuous curve enables designing cameras whose rays reach around occluders to sample surfaces that are not visible from the viewpoint. Rendering a

dataset from a reference point with a camera designed to have enhanced disocclusion capability results in a multiperspective image that stores sufficient samples to support quality reconstruction of output frames from viewpoints other than the reference point.

The non-linear rays are designed for the resulting camera to provide a fast projection operation. This way, the multiperspective image can be rendered efficiently with the feed forward approach. The rays are also designed to avoid ray intersections, as ray intersections lead to imaging a geometric primitive multiple times, and therefore to unnecessary redundancy of the set of visible samples.

Like the sample dimension generalization element of our paradigm, ray geometry generalization extends the visibility computation capability of images to support view translation. The two elements are orthogonal and they can be used in conjunction. Sample dimension generalization has the benefit of a higher quality visibility solution. Ray geometry generalization has the advantage of computing the visibility solution faster. Moreover, multiperspective images have a single layer and they are coherent, which makes it human interpretable. The multiperspective image can be shown directly to the user to support occlusion management in visualization applications. In other words, the user can directly see the data subset visible from multiple viewpoints.

1.5 Dissertation statement

This dissertation makes the following statement.

The accuracy and efficiency of sample-based 3D visibility computation using images is improved by generalizing the sampling pattern, the visibility sample, and the geometry of the rays of the camera model used to render the image.

1.6 Preview of results

We have proven this statement by developing several visibility algorithms. We give here a preview of our results.

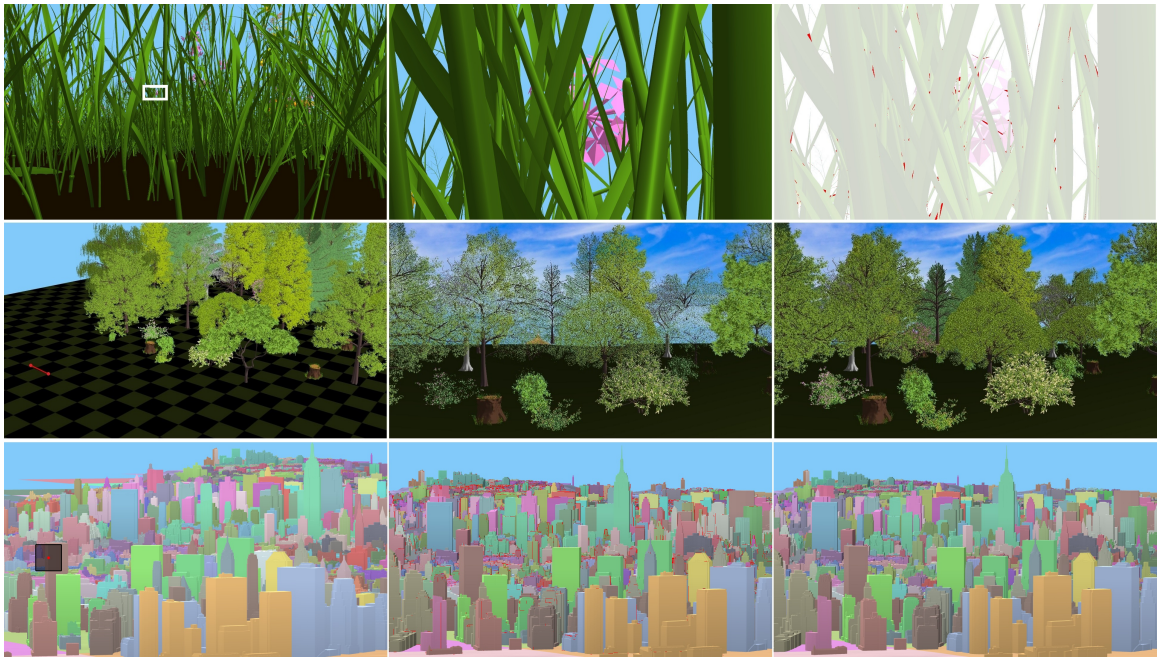

Figure 1.2. Preview of framebuffer generalization result

1.6.1 Framebuffer generalization

We have used the sampling pattern and the visibility sample generalization elements of our image generalization paradigm to develop four visibility algorithms.

1. We have used sampling pattern generalization to develop a quality-guaranteed aggressive from-point visibility algorithm. In Figure 1.2, top row, the algorithm computes the correct visibility over 99.93% of the reference image. The aggressive set yields high quality frames even under substantial magnification of the reference image. On the grass scene with 57 million triangles, the left image is computed with complete visible set, and the middle image shows a frame with 17× magnification rendered from the aggressive visible set, the pixel error is 0.26% and visualized in the right image. The few incorrect pixels are at surface boundaries, and never inside visible surfaces, so the frames are virtually indistinguishable from truth.

2. We have used the sampling pattern and the dimensionality of the visibility sample generalization to develop an algorithm for exact from-point visibility. Sampling all triangle fragments as prescribed by sampling pattern generalization does not guarantee finding all visible triangles since a partially visible fragment might be

sampled at a point where the fragment is hidden. Starting from the aggressive visible set, sampling locations are added iteratively to determine the visibility of triangles that are not hidden by the known visible triangles. The additional sampling locations are defined deterministically and not heuristically, based on the visibility subdivision of the image defined by the current set of visible triangles. No hidden triangle is ever added to the visibility subdivision. Because the starting set is almost complete, the visible set converges quickly to the exact set. The algorithm extends the aggressive set from Figure 1.2, top row, to the exact set in just three iterations, despite the complex occlusion patterns in the grass scene with 57 million triangles. The algorithm essentially computes a single 2D visibility sample that solves from-point visibility exactly. The 2D visibility sample is constructed iteratively, with the help of conventional 0D visibility samples that are placed according to sample pattern generalization principles.

3. The third algorithm is an aggressive visibility from a line segment algorithm that is exact under view translation. In other words, the algorithm finds all triangles that are visible in an image as the viewpoint translates on a segment, and the view does not rotate. The algorithm adds sampling locations aggressively and then computes the exact visible set directly over the entire view segment for each sampling location. A sampling location stores a 1D visibility sample that records all triangles visible as the viewpoint translates. When the view translates (and does not rotate or change focal length), the output image pixel centers remain a subset of the sampling locations, and the visible set is exact. In Figure 1.2, middle row, rendering an intermediate frame from the triangles visible at the endpoints of the view segment results in severe visibility artifacts, whereas the correct frame is obtained when using the visible set computed with our algorithm. The same algorithm also computes visibility over a time interval from a fixed viewpoint in a dynamic scene. Left image is the visualization of view segment, and middle image shows a frame rendered on the view segment with triangles visible from view segment endpoints; right image shows the correct frame rendered from our exact under translation algorithm.

4. The fourth algorithm computes visibility directly over a view rectangle, and the algorithm is exact under view translation. A sampling location stores a 2D visibility sample that records all triangles visible as the viewpoint translates over the view rectangle. The from-rectangle visibility problem is decomposed into a set of from-point visibility problems, one for each sampling location, which are solved

with our exact from-point algorithm. Since visibility is computed exactly for each sampling location, the resulting visible set contains all triangles visible as the view translates anywhere inside the view rectangle (Figure 1.2, bottom row). View region visualization is shown in left image, and middle image shows error visualization for a frame rendered from the triangles visible from the view rectangle corners; correct frame rendered from the visible set computed by our algorithm directly over the entire view rectangle is shown in right image.

1.6.2 Animated depth images

We have used the visibility sample generalization element of our image generalization visibility paradigm to support the interactive remote visualization of dynamic datasets. This work was published in the IEEE Transactions on Visualization and Computer Graphics [1]. The importance of remote visualization has grown and will continue to grow for the foreseeable future. One reason is that the amount of data obtained through observations and simulations increases much faster than our ability to transfer data from one geographic location to another. Another reason is that storing, processing, and displaying large datasets requires advanced capabilities which cannot and should not be replicated at all sites interested in a given dataset. Finally, the number of locations from where access to a given dataset is desired has increased with the proliferation of mobile computing platforms such as laptops, tablets, or even smartphones. One approach in interactive remote visualization is to send the visualization parameters from the client interested in the visualization to the server who stores the dataset of interest, to compute the desired visualization frame on the server, and to send the frame to the client where it is displayed. The approach requires no storage, computing, or visualization capabilities at the client and therefore it is suitable for any type of client that can display an image. However, for networks such as the internet, the approach can suffer from long latencythe network has to be traversed twice for each frame, once to send the visualization parameters and once to receive the image. Moreover, even though a single frame is much more compact than the entire dataset, sending a frame several times a second still requires aggressive compression or reduced resolution. Another challenge is achieving scalability with the number of clients, as each client sends frequent frame requests to the server. Based on the observation that a conventional image becomes obsolete with the slightest change

in visualization parameters, we take the approach of sending enhanced images, or superimages, from the server to the client. A superimage contains sufficient data to enable the quality reconstruction of thousands of frames at the client, without any additional data from the server. Reconstruction is fast and local, which greatly alleviates latency and achieves interactive rates without aggressive compression or reduced resolution. Moreover, client requests to the server are much less frequent, which improves scalability with the number of clients.

The challenge of the proposed approach is to construct a superimage that is robust to visualization parameter changes. In the case of dynamic geometric datasets, visualization parameters typically include three translations (x, y, z), three rotations (r_x, r_y, r_z), and the focal length f for the view, as well as the time parameter t. Whereas a frame is valid for a single point $(x_0, y_0, z_0, r_{x0}, r_{y0}, r_{z0}, f_0, t_0)$ in this multi-dimensional space of visualization parameters, it is our goal to construct a superimage that is valid for an entire volume $(x_0+\Delta x, y_0+\Delta y, z_0+\Delta z, r_{x0}+\Delta r_x, r_{y0}+\Delta r_y, r_{z0}+\Delta r_z, f_0+\Delta f, t_0+\Delta t)$. The superimage should allow reconstructing quality output frames for any visualization parameter values in the volume it covers.

We address the rotations and focal length changes naïvely by simply having a larger field of view and a higher resolution than the frame. This way the view can rotate and zoom in without running out of samples and without blurriness due to undersampling to a certain extend. The output image is computed through ray resampling and the result is correct, including for view dependent effects such volume rendering and reflections. We anticipate the view translations by using multiple from viewpoint images adaptively until an error threshold is reached while eliminating the redundancies of samples in between frames on the fly.

We address the problem of creating an image that covers a time interval as opposed to a time step. We introduce animated depth images, the rays of which not only sample depth and color samples, but also approximation of the trajectories of the samples over time. Like conventional depth images, animated depth images allow adapting the level of detail, provide occlusion culling, and bound the amount of data that has to be transferred. Unlike conventional depth images that can only capture a single snapshot of a dataset, animated depth images provide a quality approximation of a dynamic dataset for an entire time interval. Compared to a static/pre-computed video sequence, animated depth images have the advantage of interactivity, as the user

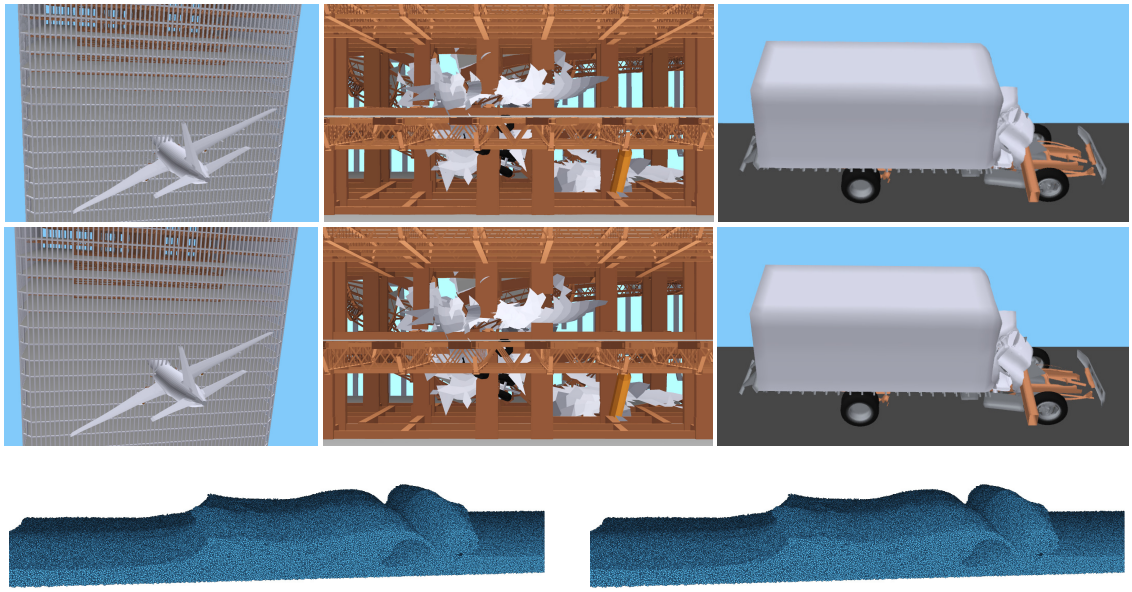

Figure 1.3. Animated depth image result preview.

is free to translate anywhere in the vicinity of a reference point, and the advantage of allowing to slow down the animation to any desired rate.

Animated depth images imply the following challenges. First, sample trajectories have to be stored compactly. Storing the trajectory of every sample is prohibitively expensive. Instead, we leverage sample coherence to assign trajectories to groups of samples based on a semi-rigid body decomposition of the image. Second, the reconstruction of output frames has to be done by taking into account sample connectivity that changes as samples move. The third problem is that of disocclusion errors, due not only to viewpoint translations, but now also due to motion within the dataset. We take the approach of sampling the dataset adaptively from multiple viewpoints and multiple time steps to derive on the server a good approximation of the set of necessary and sufficient samples, which are then transferred to the client. The approach is robust in the context of extremely challenging occlusion patterns. When the viewpoint desired at the client moves outside the neighborhood covered by the set of samples, or when the client desires to visualize the dataset at a time step outside of the interval covered, the adaptive sampling process is repeated, and a new set of animated samples are transferred from the server to the client.

We apply animated depth images in the context of interactive remote visualization of finite element analysis (FEA) datasets. FEA datasets are particularly challenging dynamic datasets since there are millions of degrees of freedom, with nodes moving independently, which creates complex occlusion patterns and massive data replication. Figure 1.3 top 2 rows show that animated depth images enable the reconstruction of quality visualization frames that are comparable to frames rendered directly from the original FEA dataset. Top row of Figure 1.3 shows frames rendered from animated depth images. Middle row shows frames rendered from the original FEA dataset, for comparison. The animated depth images approximate sample trajectories with a user-specified maximum error of 10mm, which represents 0.01% of the spatial extent of the aircraft dataset (left and middle), and 0.1% for the truck dataset (right) respectively. An adaptive sampling strategy reduces disocclusion errors below 1% for viewpoint translations of up to 10m around the reference viewpoint.

We also extend our approach to SPH datasets Fig 1.3, bottom row. Left image shows frame reconstructed from an animated depth image and right image shows frame obtained by rendering the original SPH dataset.The residual disocclusion error is 0.35% and thus virtually indistinguishable.

1.6.3 Flexible pinhole camera model

We have used the sampling pattern generalization element of our image generalization paradigm to develop a novel camera model that overcomes the uniform sampling rate limitation of conventional images. This work was published in IEEE Computer Graphics and Applications [2].

We introduce the flexible pinhole camera or FPC which allows for adjustments of the sampling rate according to the local importance or complexity of the data imaged. Like the planar pinhole camera (PPC), the FPC is defined by a viewpoint (i.e. center of projection or eye) and an image plane. However, the sampling locations are not defined by a uniform grid but rather by a sampling map that allows shifting sampling locations from one region of the image plane to another. The FPC image provides a coherent non-uniform sampling (CoNUS) of the dataset. The CoNUS image in Figure 1.4, left, samples the five faces at a higher rate. The sampling map has the topology of a 32×32 regular rectangular mesh but it is distorted to implement the sampling rate modulation.

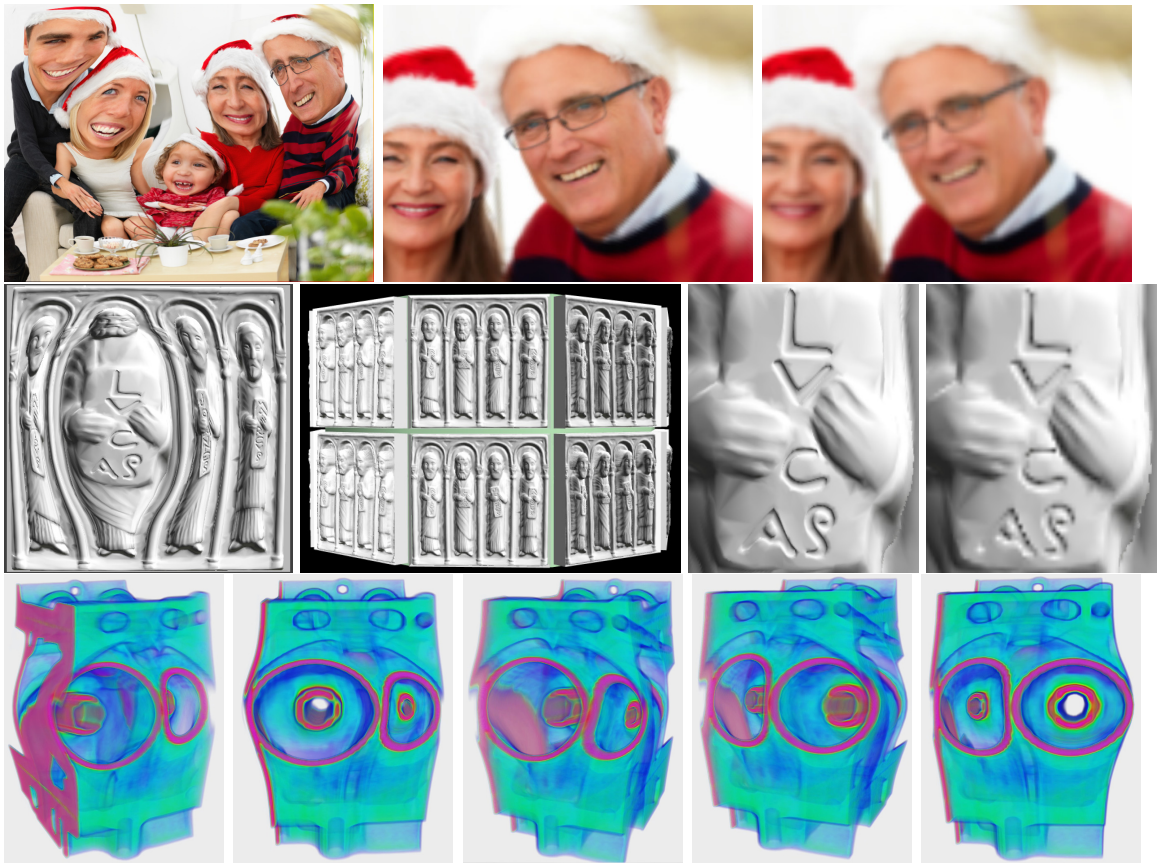

Figure 1.4. FPC result preview with CoNus images.

The sampling map underlying the FPC can be constructed from known regions of interest in a variety of ways. We build sampling maps in one of three ways: (1) interactively, using a physics-based mass-spring system, (2) by composing multiple sampling maps together, or (3) analytically. The FPC provides fast 3D projection which allows rendering CoNUS images quickly, in feed-forward fashion, by projection followed by rasterization, from many types of data. We demonstrate FPC rendering of CoNUS images from image, height field, geometry (i.e. 3D triangle meshes), and volume data. We explore the use of CoNUS images in the contexts of remote visualization, of focus-plus-context visualization, and of acceleration of expensive effects such as surface geometric detail and specular reflection rendering

Consider the example of a high resolution portrait photograph that has to be downsampled to be sent over the internet (Figure 1.4, top row). The recipient is likely to want greater detail on the faces, which, of course, cannot be provided by zooming into the conventional downsampled image. If instead the server sends a CoNUS image of same size but with a higher sampling rate at the known regions of interest, the user can zoom in with better results. Consider a second example where a height field is visualized remotely. The server sends periodically height field sections corresponding to the current user location. If instead the server sends CoNUS height fields, the fidelity of the output frame increases considerably. Like any height field, the CoNUS height field samples the ground plane orthogonally, avoiding occlusions. However, the sampling pattern is defined analytically to match the sampling rate requested by the output frames. Left image shows a coherent non-uniform sampling (CoNUS) image that allocates more samples to the face regions, and middle image shows output frame reconstructed from CoNUS image. It is clear that CoNUS image reconstruction is better by comparing with output frame reconstructed from a conventional image of same size shown in the right image.

Many techniques employ depth images in order to accelerate expensive rendering effects. In relief texture mapping a depth image is used to enhance a surface with geometric detail. In specular reflection rendering the environment mapping approximation errors are avoided by modeling objects close to reflectors with depth images. The main reason depth images accelerate these effects is that one can compute the intersection between a ray and a depth image faster than one can compute the intersection between a ray and the original geometry. A CoNUS depth image brings sampling flexibility (Figure 1.4, middle row), without increasing the cost of the intersection operation. Left image shows a CoNUS relief texture that allocates more samples to a tablet of interest; middle image shows the texture mapping with this relief texture, and comparison between frames that zoom in on tablet of interest and are rendered with CoNUS and with conventional relief textures of same size are displayed in the right two images.

Whereas in the examples presented so far the need for non-uniform sampling is to improve an auxiliary data representation from which conventional output images are computed, in the case of focus-plus-context visualization the CoNUS image is shown directly to the user. The FPC approach enables a versatile focus-plus-context visualization technique that can handle any type of data and that provides good

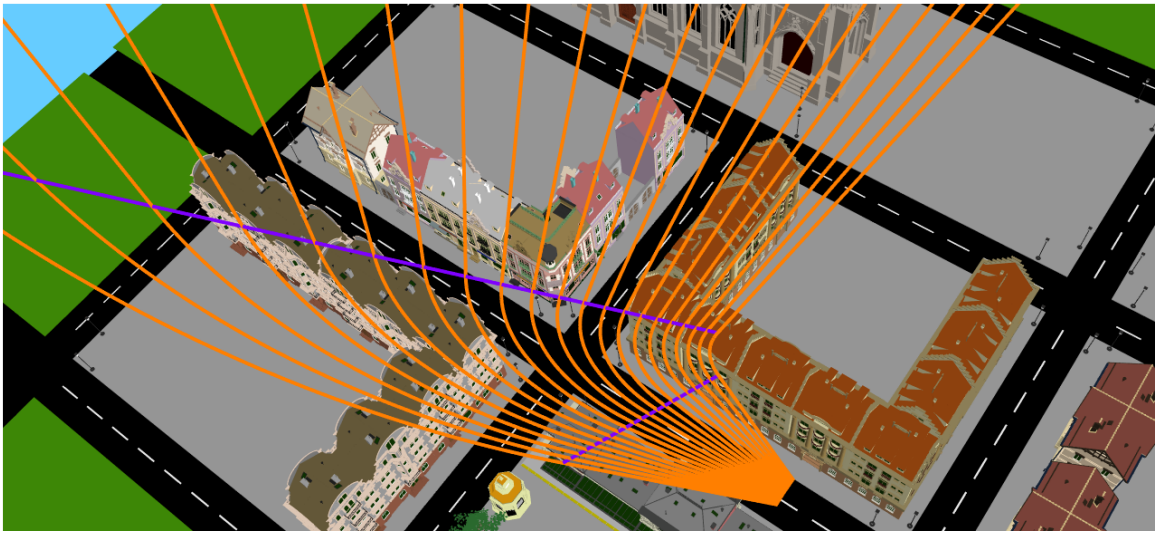

Figure 1.5. The curved ray camera ray visualization.

control over the focus regions. The CoNUS images are rendered directly from the dataset (e.g. volume data, geometry) using the FPC, and they are not obtained by first rendering and then downsampling a high resolution conventional image. A focus-plus-context visualization for volume rendering example is shown in Figure 1.4, bottom row, emphasizing on two cylinder housings.

1.6.4 Curved ray camera model

We have used the ray geometry generalization element of our image generalization visibility paradigm to develop the curved ray camera (CRC), a camera with C^1 continuous rays that reach around occluders to gather sufficient samples for a comprehensive multiperspective visualization of the dataset. This work was published in the IEEE Transactions on Visualization and Computer Graphics [3].

A conventional image can only sample surfaces to which there is a direct line of sight. An elegant solution for integrating into the image samples that are not visible from the reference viewpoint is to design the set of rays to include rays that bend to go past occluders to reach data subsets of interest.

In some applications, the goal is to capture samples that are not visible from the reference viewpoint, but are visible from nearby viewpoints. Occlusion cameras [4–

6] are a family of cameras that gather such barely hidden samples to generate a multiperspective image that has sufficient samples to provide good reconstructions of output frames from nearby viewpoints.

In other applications, the rays have to reach deep into the dataset to expose data subsets that are hidden from anywhere near the reference viewpoint. Consider a scientific visualization application where the goal is to search for a feature of interest in a large dataset. The conventional approach is to navigate interactively a conventional camera in search of the feature of interest. The user has to be able to retrace their steps in order to achieve a systematic exploration of the dataset. The navigation approach is even more problematic in the case of dynamic datasets, where it can happen that the user reaches the locale of a transient phenomenon of interest too late for the phenomenon to be observed. Dynamic datasets could be explored in parallel with a set of several fixed conventional cameras, but the parallel visualization based on multiple discontinuous and redundant images is confusing. The user has to spend significant cognitive effort to adapt to each one of the contexts, and the efficiency of such parallel visualization is only marginally better than that of serial visualization.

A promising approach for parallel visualization is to integrate multiple conventional images, each with its own viewpoint, into a continuous multiperspective image. The graph camera [7] achieves such an integration with C^0 continuity between the individual perspectives. Although the resulting multiperspective image is continuous, meaning that nearby 3D points project to nearby image locations, the abrupt transition from one viewpoint to the next introduces a significant visual artifact that lowers the quality of the parallel visualization. The artifact is due to the fact that the rays of the graph camera are piecewise linear. When the ray switches from one perspective to the other, the ray breaks from a line through the first viewpoint to a line through the second viewpoint.

The CRC alleviates the abrupt perspective change artifact of graph camera images by transitioning from one viewpoint to the next gradually, over a transition region. A CRC ray is a sequence of line segments interconnected with conic arcs. A conic arc is tangent to both line segments it connects (Figure 1.5, top). The gradual transition from one viewpoint to the next eliminates the distortion of objects as they switch perspective (Figrue 1.6).

Although the simplest curve that can meet the endpoint tangency requirements is a Bézier curve, modeling the CRC rays with Bézier arcs does not provide a closed-

Figure 1.6. The curved ray camera C^1 continuous transition (right) eliminates the abrupt change of perspective (left).

form projection operation. We use general conic arcs which do provide closed-form projection. The CRC frustum is a chain of PPC subfrusta connected by transition subfrusta. The rays of a PPC subfrustum do not intersect other than at the PPC viewpoint. The rays of a transition subfrustum do not intersect as they are family of conic arcs defined by a linear parameterization of the entry and exit rectangles of the PPC subfrusta the rays connect. Finally, the subfrusta of the CRC do not intersect by construction.

We have developed three CRC constructors. One constructor allows the user to add and modify viewpoints interactively. The CRC has many more degrees of freedom than a conventional camera. Each of the viewpoints can be placed to achieve the desired disocclusion effect. A second constructor builds a CRC to keep a target data subset disoccluded. Given a 3D dataset, a desired viewpoint, and a target, the target tracking constructor builds the CRC rays to find a path to the target. If the target is visible in a conventional image, the CRC reverts to a PPC. Target tracking works in multiple contexts, including a moving target and a stationary viewpoint, a moving viewpoint and a stationary target, and a moving target and a moving viewpoint. A third constructor builds a CRC that follows a given path. The constructor adds viewpoints to guide the rays of the CRC along the path. The CRC visualization straightens the path and shows what is visible along the path, avoiding the occlusions that the turns of the path would cause in a conventional visualization.

2 FRAMEBUFFER GENERALIZATION

A conventional image is a powerful visibility computation tool. An image is computed efficiently using graphics hardware, and every image pixel finds a visible triangle. However, the visible set found by a conventional image is far from complete because of its *uniform and finite sampling rate*. First, triangles can have an arbitrarily small image footprint due to a high scene complexity, to a large distance to the eye, or to a grazing viewing angle. In Figure 1.1 (top), the visible set found by the reference image of the finely tessellated sphere is incomplete, which results in severe artifacts when the set is used to render an image from the same viewpoint but with a slightly different view direction (0.2°).

Increasing the resolution of the reference image is only palliative: the image footprint of a visible triangle can be arbitrarily small, therefore an infinite resolution would be needed to guarantee that all visible triangles are found. Second, an image can rule a triangle as visible, but it cannot rule a triangle as hidden. Whereas a single sample can be sufficient to verify that a triangle is visible, an infinite number of samples would be needed to verify that a triangle is hidden at all its points. Third, a conventional image finds visible triangles from a viewpoint and not from a view region, missing triangles that become visible as the viewpoint translates. Moreover, a conventional image only computes visibility for a single time point, missing triangles that becomes visible as time changes.

We generalize images to remove these shortcomings in three ways: (1) by adding sampling locations to sample all triangles fragments, which guarantees that all triangles with a completely visible fragment are found, no matter how small their image footprint; (2) by adding sampling locations based on a visibility subdivision of the image, which completes the set of visible triangles efficiently; (3) and by generalizing the ray sampling at a sampling location from 0D to 1D and to 2D, which supports viewpoint translations and time changes. We have used the paradigm to develop a quality-guaranteed aggressive from viewpoint visibility algorithm, an efficient exact from viewpoint visibility algorithm, a from view segment and a from view rectangle

visibility algorithm that are exact under view translation, and an over time interval visibility algorithm that is exact for a constant view.

In this chapter, the PVS algorithms are introduced in the order of problem domain: from viewpoint aggressive, from viewpoint exact, from view segment, over time interval and finally from a view polygon. This chapter ends with conclusion after our experimental result on a variety of complex static and dynamic datasets, including 3D surfaces, FEA dataset and SPH dataset.

2.1 Prior work

Visibility algorithms are classified based on the visible set they compute. *Conservative visibility algorithms* overestimate visibility, such that no visible triangle is omitted. The benefit is an accurate image, but the number of hidden triangles unnecessarily included in the solution can be substantial [8, 9]. *Aggressive visibility algorithms* underestimate the set of visible triangles, which leads to image errors. The goal of aggressive visibility research is to reduce and control the error [10, 11]. *Exact visibility algorithms* find only and all visible triangles, which avoids the cost of rendering unnecessary triangles as well as any image error.

Aggressive Visibility. We distinguish between probing visibility by casting individual rays, and by rendering entire images. Algorithms in the first category use randomly generated rays [12, 13], or heuristics to shoot rays that are likely to find visible triangles, and subsequent sampling is guided by what the initial rays find e.g. [11, 14]. The advantage is the flexibility to cast precisely the rays deemed necessary, which limits sampling redundancy. Some approach shoots bundles instead of rays in order to exploit the spatial coherency of rays [15]. However, it is difficult to place error bounds on the results. Moreover, casting individual rays can only be done efficiently with a hierarchical scene subdivision, which is difficult to extend to dynamic scenes. For dynamic scenes, 4D ray tracing is firstly proposed with the assumption that the majority of geometric primitives do not change [16, 17].

Algorithms in the second category leverage the fact that the amortized cost of rays in an image is lower than that of individual rays. Our aggressive visibility algorithms fall in this category. An image only captures samples visible from its viewpoint, which provides exact visibility for that viewpoint and rays only [18, 19]. One option is to use images from additional viewpoints [10, 20, 21], which are highly redundant, or to

eliminate redundancy as a pre-process [22, 23]. The challenge of these approaches is to decide which images are needed for a sufficient sampling of the visibility parameter space. The usual strategy is to sample uniformly "as densely as possible", and thus the visibility error is not bounded. Multiperspective images capture in a single shot more than what is visible from a single viewpoint through innovation at the camera model level [3, 24], but there is no quality guarantee on the visible set they gather.

Specialized visibility algorithms have been developed for many computer graphics contexts. The algorithms are typically aggressive, focusing on finding the visible triangles of highest relevance in the particular context. *The semi-analytical visibility* algorithm [25], developed for motion blur, samples the image with lines as opposed to points, an idea borrowed from temporal antialiasing [26]. Visibility is analyzed continuously over time for each line sample. The algorithm is aggressive because the analysis is restricted to a uniform grid of image lines. Line samples are a brute force approach for improving uniform point sampling. The line parameter adds an expensive second dimension to the 1D motion blur visibility problem. The uniform line sample pattern is heuristic, so even after solving the higher-dimension visibility problem, there is still no guarantee for the quality of the solution. We propose deterministic point sampling that guarantees a quality visible set without increasing the visibility problem dimensionality.

Exact Visibility. Early work focuses on from-point visibility for anti-aliasing. The solution was to compute a visibility partition for each pixel, defined by the triangle fragments visible at each pixel [27–29]. The solution is inefficient because fragments of hidden triangles are added and then removed from the visibility partitions. The exact from-point visibility algorithm we describe computes pixel visibility partitions exclusively from visible triangles.

Pixel-free from-point visibility algorithms are also inefficient because they compute occluded intersections (e.g. [30]); typical running times are $O((n+k)\log n)$ or $O(n \log n + k + t)$ for n triangles with k edge intersections and t triangle intersections on the image plane. Output sensitive algorithms are restricted to special input [31, 32]. From-point visibility was implemented on the GPU [33], but with a running time quadratic in the number of triangles.

Beam tracing [34] analyzes visibility in 3D by partitioning the 2D space of rays defined by the viewpoint using conical [35] or frustum-like beams. The unsampled gaps between rays are avoided, but beam-triangle intersection is costly. Beam-tracing

has more recently been used for shadow [36, 37] and sound [38] rendering, using acceleration schemes based on adaptive beam splitting. Our paradigm bypasses the need for beams in from-point visibility: the beam is replaced with the smallest number of rays needed to capture the visible triangles over a solid angle. Moreover, we do not trace multi-dimensional rays but rather evaluate visibility over their sampling domain by projection.

One strategy for from-rectangle visibility is to decompose the 4D non-Euclidean space of lines in the dataset according to visibility criteria. *Aspect graph algorithm* [39] uses visibility events to subdivide the 3D view space, which has a $O(n^9)$ complexity in computation. A more efficient model with $O((n^3 + k) \log n)$complexity and output sensitive construction is yet to be practical [40]. Another strategy is to compute visibility between pairs of polygons [41–45], using constructive solid geometry in the 5D ambient space of the Plucker coordinate representation of lines. The algorithms have high computational complexity and are not output sensitive.

In order to reduce complexity, some researches reduce 3D visibility problem to 2D by restricting the viewpoints on the ground of a 2.5D scenes, such as urban and terrains [46–52]. For indoor scenes, with the clear definition of windows and portals, visibility problem can be solved efficiently [53, 54]. However, the complexity of these approaches converges to other 3D algorithms when the dataset does not meet the prerequisites.

Conservative Visibility algorithms are exact algorithms that run on a visibility problem that was conservatively simplified, e.g. through extended projections [55], or occluder erosion [56]. Our aggressive visibility algorithms produce a visible set that is almost complete, so adding to the set all triangles not hidden by the aggressive set yields a good conservative visible set (i.e. with only a small number of hidden triangles). Per-frame occlusion culling improves rendering performance by batch discarding triangles that are hidden in the current output frame [57–61], which is further optimized with the help of graphics hardwares [62–64]. Triangles are grouped inside containers with simple geometry, the containers are rendered on a partial z-buffer of the output frame obtained from known big blockers, and the triangles of hidden containers are discarded.

Occlusion culling methods can also be aggressive by fusing blockers [59]. Grouping occluded and occluding geometry is heuristic, and it is particularly challenging in

the case of dynamic scenes. For each time step, the occlusion mask needs to be updated [65] and becomes obsolete when the occluding pattern changes drastically.

Irregular Framebuffers. Our paradigm advocates abandoning the uniform sampling of conventional images in favor of adding sampling locations deterministically to guarantee that all visible triangles are found. The benefits of irregular framebuffers have been noted before in contexts that include: pixel-accurate shadow mapping [66, 67], where the shadow map estimates light visibility precisely at the point samples captured by the output image; point-based rendering [68], where projected reference image samples are not clamped to the output image pixel grid but rather located precisely within the output image pixel using a pair of offsets; and focus plus context visualization where focus regions are sampled at a higher rate [69].

2.2 From viewpoint algorithm

We have developed two from viewpoint visibility algorithms with the ray pattern generalization: one aggressive with a quality guarantee, and one exact.

2.2.1 Aggressive

The aggressive algorithm finds all triangles that have a completely visible fragment. This includes all triangles of a front surface, no matter their image footprint. The algorithm takes three passes over the scene triangles (Algorithm 1).

(Line 1) *The first pass* is a conventional rendering pass over a pixel grid with one sampling location at each pixel center. After the first pass, each pixel center reports one visible triangle. In Figure 2.1, the centers of pixels 0-3 are indicated with dots. The conventional image rendered finds triangles a and c, but not b, even though b is completely visible.

(Lines 2-9) *The second pass* adds sampling locations to make sure that all triangle fragments are sampled. A fragment f is defined as the intersection between the projection of a triangle t and a pixel p. If f doesn't contain any of the sampling locations of p, a new sampling location s is generated at the centroid of f, unless t is hidden at s by the triangle t_0 found by the first pass as visible at the center of p. In Figure 2.1, a generates one and b four sampling locations (crosses). Triangle d generates no sampling locations: the fragments of d in pixels 1 and 2 already contain

Algorithm 1 Aggressive from viewpoint visibility

Input: scene triangles S, viewpoint o, conventional pixel grid G
Output: aggressive set of visible triangles V_0

```
 1: Render S from o over G
 2: for all triangles t in S do
 3:     for all pixels p covered by the projection t' of t do
 4:         fragment f = t' ∩ p
 5:         if f does not contain a sampling location then
 6:             sampling location s = Centroid(f)
 7:             t_0 = triangle visible at the center of p
 8:             if t is closer to o than t_0 at s then
 9:                 Add s to p
10: Render S from o over G
11: for all sampling locations s in G do
12:     V_0 = V_0 ∪ {s.t}
13: return V_0
```

sampling locations (which were added for b); the fragment of d in pixel 0 does not contain a sampling location; however, d is hidden at the centroid of its fragment in pixel 0 by a, which was found at step 1, and adding such a sampling location is unnecessary since it is already known that d is not visible there.

(Line 10) *The third pass* renders the scene over the grid of pixels, which now have a variable number of sampling locations. Triangle projection proceeds as usual. Rasterization has to evaluate projected triangle edge equations and depth for each sampling location of each pixel touched by the triangle projection. After the third pass, each sampling location reports a visible triangle, which are collected into the visible set (lines 11-12) .

A sampling location is useful only if it reveals that its triangle is visible at that point. If the triangle is not visible at the sampling location, one cannot rule the triangle as hidden, and the sampling location is wasted. The first pass computes a preliminary set of visible triangles efficiently, then the second pass avoids creating sampling locations that are already known not to be useful based on the preliminary set. If a pixel is completely covered by a visible triangle, the pixel will have only its initial sampling location at its center; the visible triangle is found at the first pass, and all candidate sampling locations are discarded by the triangle at the second pass.

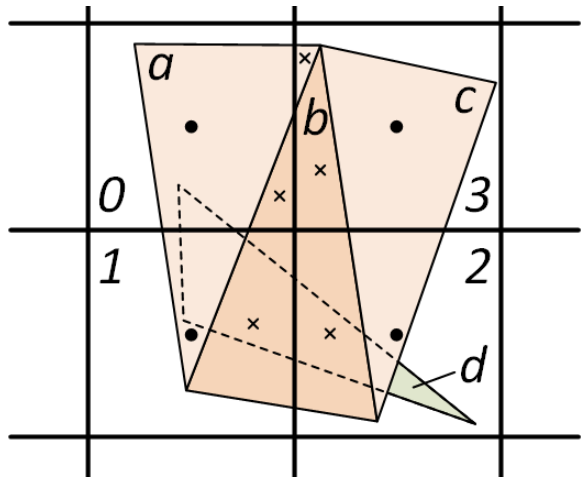

Figure 2.1. Quality-guaranteed aggressive from viewpoint visibility.

2.2.2 Exact

We have developed an efficient algorithm that extends the aggressive visible set iteratively to the exact set (Algorithm 2).

Algorithm 2 Exact from viewpoint visibility

Input: scene triangles S, viewpoint o, aggressive visible set V_0
Output: exact set of visible triangles V

```
Initialize V = V_0, U = S − V_0
Construct visibility subdivision VS from V_0
while U is not empty do
    for all triangles t in U do
        if t is hidden by VS then
            remove t from U
        else
            add sampling locations for t
    Render triangles in U
    Collect newly found visible triangles V_i
    Update VS with V_i and set U = U \ V_i, V = V ∪ V_i
return V
```

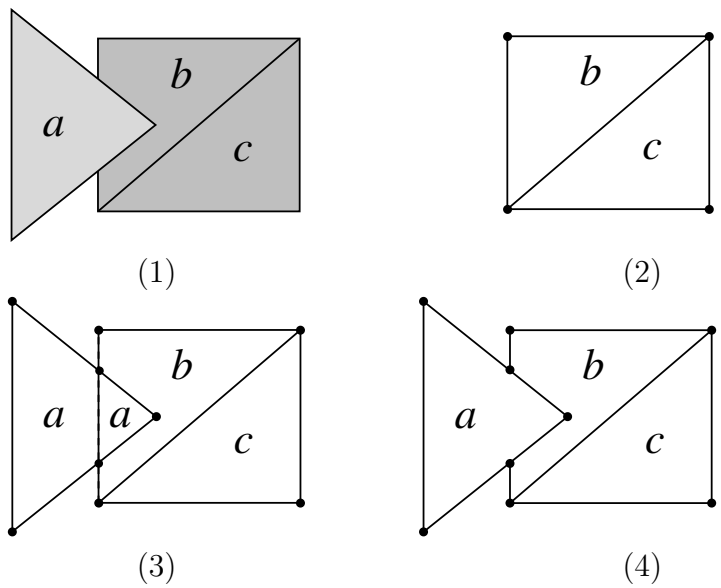

Figure 2.2. Incremental construction of visibility subdivision. The scene consists of three triangles, a, b, and c, with a partially occluding b (1). The visibility subdivision constructed from b and c (2) is updated according to a (3-4).

(Line 1) The algorithm first initializes the visible set V to the aggressive visible set V_0 received as input, and all other scene triangles are added to the set of undecided triangles U.

(Line 2) Then the algorithm constructs an initial visibility subdivision VS of the image from the aggressive visible set V_0 received as input (2). VS is a subdivision of the image plane into polygonal regions where one or no triangles are visible. The region boundaries are the visible portions of the projected triangle edges. VS is constructed incrementally, by adding one triangle at the time. In the example in Figure 2.2, the visibility subdivision built for triangles b and c contains three visibility regions, where b, c, and no triangles are visible. The visibility subdivision is updated with triangle a in two steps. First, the subdivision is intersected with a which results in shrinking the region where b is visible and in creating two regions where a is visible. Then the two regions for a are merged into one region to obtain the updated visibility subdivision.

(Lines 3-11) The algorithm iterates until there are no more undecided triangles.

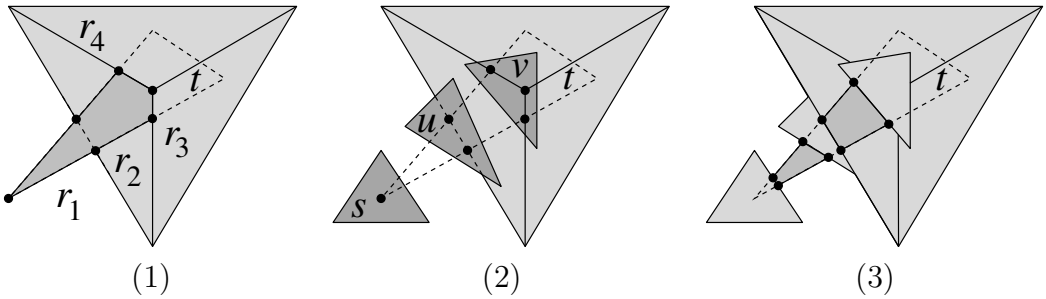

Figure 2.3. (1) six sampling locations, shown with dots, created for undecided triangle t at first iteration, (2) triangles s, u, and v visible at those sampling locations, and (3) eight sampling locations created for t at second iteration.

Each iteration first processes the undecided triangles (lines 4-8). If an undecided triangle t is hidden by the visibility subdivision, t is removed from further consideration. If t is not hidden in region r of the subdivision, sampling locations are created at the vertices of t that project in r, at the vertices of r inside the projection of t, and at the intersection points of the edges of r and the projected edges of t. In Figure 2.3.1, t is hidden in regions r_3 and r_4, but not in r_1 and r_2, and six sampling locations are created.

After the undecided triangles are processed, the remaining undecided triangles are rendered over the new sampling locations (line 9), a step identical to step 3 of the aggressive algorithm. The visible triangles found by the sampling locations (line 10) are used to update the visibility subdivision, and they are transfered from the undecided set to the visible set (line 11).

The algorithm is fast because the initial visible set is almost complete and most remaining triangles are hidden by the initial visibility subdivision. The visibility subdivision is built exclusively from visible triangles. Figure 2.3 illustrates an unlikely scenario where a triangle remains undecided after an iteration: all sampling locations created for t (1) are won by other undecided triangles (2) and more sampling locations are needed to decide t (3). In our experiments, the algorithm converged in at most three iterations even for scenes with tens of millions of triangles and complex occlusion patterns.

2.3 From view segment visibility

We have developed an algorithm that computes visibility from a view segment directly. The input is a scene S modeled with triangles, a view segment o_0o_1, and a conventional grid of pixels G. The output is an aggressive approximation of the set of triangles visible from points on the view segment, with the following quality guarantees:

1. The visible set contains all triangles that have a fragment f that is completely visible from any viewpoint on o_0o_1 where f exists. In other words, if a triangle t has a fragment f in a pixel p and f is completely visible from any viewpoint from where the projection of t touches p, the algorithm finds t. Like before, this guarantees that all triangles of a front surface are captured, no matter how small their footprint.

2. The visible set contains all triangles visible at the pixel centers as the view translates from o_0 to o_1. This makes the algorithm exact under view translation: the visible set produces correct frames from any intermediate viewpoint.

Algorithm 3 From view segment visibility

Input: scene triangles S, view segment o_0o_1, pixel grid G
Output: aggressive visible set V, exact under view translation

```
for all triangles q in S do
    h = Footprint(q, o0o1, G)
    for all pixels p touched by h do
        fragment f = p ∩ h
        if f doesn't contain a sampling location add one
for all triangles q in S do
    h = Footprint(q, o0o1, G)
    for all pixels p touched by h do
        for all sampling locations s of p do
            q.VI = ComputeVisibilityIntervals(q, s, o0o1)
            s.VI = MergeVisibilityIntervals(s.VI, q.VI)
for all pixels p in G do
    for all sampling locations s in p do
        for all visibility intervals vi in s do
            V = V ∪ vi.triangle
return V
```

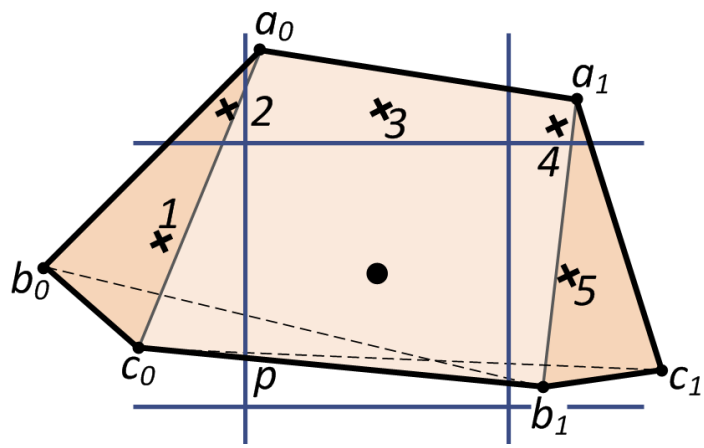

Figure 2.4. Triangle footprint as the view translates.

The from view segment algorithm has two main stages (Algorithm 3): enhancing the pixel grid with additional sampling locations (lines 1-5), and computing visibility at each sampling location (lines 6-11).

Lines 1-5 In the first stage, sampling locations are added to sample all triangle fragments. This is done like before by computing the image footprint of the triangle, by computing its fragments, and by adding a sampling location at the centroid of a fragment that does not already contain one. The only difference is that now the footprint of the triangle is the union of all triangle projections as the view translates. We use a fast, tight, and conservative approximation of the triangle footprint as the convex hull of the six extremal vertex projections. In Figure 2.4 the projection of the triangle moves from $a_0b_0c_0$ to $a_1b_1c_1$ as the viewpoint moves from o_0 to o_1, and the convex hull $a_0b_0c_0b_1c_1a_1$ has six fragments, each with one sampling location.

Lines 6-11 The second stage takes another pass over the scene and updates visibility at each sampling location touched by each triangle. Whereas in the case of from viewpoint visibility there is a single triangle visible at a sampling location, in the case of from view segment visibility multiple triangles can become visible at a sampling location as the viewpoint translates. The translation along the view segment defines a one-parameter visibility change. The visibility sample of a sampling location is generalized from 0D to 1D to store a list of visibility intervals that define which triangle is visible at the sampling location and for which view sub-segment.

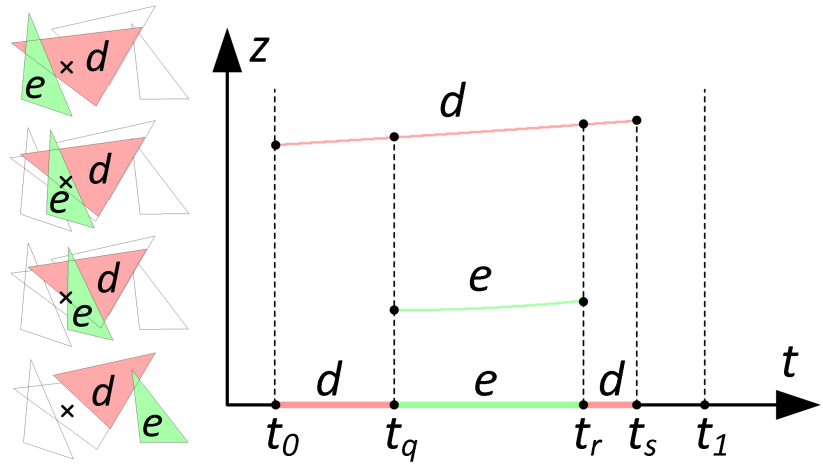

Figure 2.5. **Left**: triangles d and e move over a sampling location (cross). **Right**: visibility intervals at the sampling location.

In Figure 2.5, left, two triangle projections e and d move over a sampling location as the viewpoint translates along the view segment. The translation is linear with parameter t. The graph (right) shows the depths z from the viewpoint to the triangles, at the sampling location, as the viewpoint translates. The visibility intervals stored are shown on the abscissa. Initially the visibility sample of the sampling location is empty. After d is processed, there is one visibility interval $[t_0, t_s]$ for d. After e is processed, there are three visibility intervals: $[t_0, t_q]$ for d, $[t_q, t_r]$ for e, and $[t_r, t_s]$ for d.

Updating visibility at a sampling location s for a triangle q is done in two steps: the first step computes the visibility intervals for q (line 10), and the second step intersects and z-buffers the visibility intervals of q with the visibility intervals already stored as s (line 11).

The visibility intervals of q are the intervals of the translation parameter t when the projection of q covers s. They are computed by computing the values of t when s crosses the moving projections of the edges of q. Given two triangle vertices a and b and a sampling location s with image plane coordinates (u_s, v_s), we derive the condition that the projection of triangle edge ab contains s. The coordinates (u_a, v_a) of the projection of a are computed by writing a as a 3D point on the ray through (u_a, v_a) at w from viewpoint o:

$$a = o + M \begin{bmatrix} u_a \\ v_b \\ 1 \end{bmatrix} w, \begin{bmatrix} u_a \\ v_b \\ 1 \end{bmatrix} w = M^{-1}(a - o) \tag{2.1}$$

$$u_a = \frac{m_0 \cdot (a - o)}{m_2 \cdot (a - o)}, v_a = \frac{m_1 \cdot (a - o)}{m_2 \cdot (a - o)} \tag{2.2}$$

where M is the camera matrix, and the rows of M^{-1} are m_i. s is on the projection of ab if:

$$\frac{u_s - u_a}{u_s - u_b} = \frac{v_s - v_a}{v_s - v_b} \tag{2.3}$$

By plugging in u_a and v_a from Eq. 2.1 into Eq. 2.3, we obtain:

$$\frac{(u_s m_2 - m_0)(a - o)}{(u_s m_2 - m_0)(b - o)} = \frac{(v_s m_2 - m_1)(a - o)}{(v_s m_2 - m_1)(b - o)} \tag{2.4}$$

M is constant as the view translates along the view segment. Since m_i, v_s, u_s, a and b are constant, and since o is linear in t, Eq. 2.4 is quadratic in t. Eq. 2.4 is solved for each of the three edges of the triangle, the solutions in $[t_0, t_1]$ are kept and sorted to define the visibility intervals of the current triangle q.

In the second step, the visibility intervals of q are z-buffered with the previous visibility intervals at the sampling location s. The depth z from o to the intersection of a triangle abc and camera ray r_s through s is given by:

$$z = \frac{(a - o) \cdot n}{r_s \cdot n}, n = (b - a) \times (c - a) \tag{2.5}$$

Since a, b, c, n, and r_s are constant, and since o is linear in t, z is linear in t. For scenes where triangles do not intersect, z-buffering two overlapping visibility intervals is simply done by evaluating the two depth functions at a t inside the overlap. When triangles can intersect, the possible intersection between two triangles is found by solving the quadratic equation that results from setting their z's (Eq. 2.5) to be equal at the sampling location.

The visible set is obtained by unioning the visible triangles found by each visibility interval of each sampling location of each pixel (lines 12-15).

2.4 Over time interval visibility

The from view segment visibility algorithm can be used to compute the triangles visible over a time interval in a dynamic scene. The input is a dynamic scene modeled with triangles whose vertices move linearly over a time interval $[t_0, t_1]$, and a view V. Whereas in the case of from view segment visibility all vertices move relative to the viewpoint according to the same translation vector, now each vertex moves independently according to its own translation vector. The output is the set of triangles visible at the pixels of V at any time in $[t_0, t_1]$. The visible set is exact for view V.

Eq. 2.4 used to compute the visibility events remains a quadratic in t, as now vertices a, b, and c are linear in t and o is constant. Eq. 2.5 used to z-buffer the overlapping visibility intervals of two triangles is now a cubic in t over a quadratic in t, as n is quadratic in t. For applications where moving triangles do not intersect, such as for the finite element analysis and the smoothed particle hydrodynamics simulation applications used as examples in this chapter, the depth functions are trivially evaluated to arbitrate between the two triangles. When triangles intersect, depth function equality results in an order five equation in t, which has to be solved numerically.

The visible set is exact for the given view V because visibility is computed exactly for each sampling location, including the pixel centers of V. Visibility at a sampling location is computed continuously over the time interval by solving visibility event equations. The algorithm computes not only what triangles are visible for $[t_0, t_1]$, but also what triangles are visible at a given t in $[t_0, t_1]$, which enables exact occlusion culling.

2.5 From view rectangle visibility

We have developed an algorithm for computing visibility directly over a view rectangle (Algorithm 4). The visible set provides the same guarantees as before: the set contains all triangles that have a completely visible fragment, and the algorithm is exact under view translation. Like the from view segment algorithm, the algorithm has three stages.

Lines 1-5 Stage one adds sampling locations to a conventional pixel grid G to ensure that all triangle fragments are sampled. The triangle footprint is approximated with the convex hull of the twelve extremal projections of the triangle vertices (one

Algorithm 4 From view rectangle visibility

Input: scene triangles S, view rectangle $o_0o_1o_2o_3$, pixel grid G
Output: aggressive visible set V, exact under view translation

```
1: for all triangles q in S do
2:     h = Footprint(q, o0o1o2o3, G)
3:     for all pixels p touched by h do
4:         fragment f = p ∩ h
5:         if f doesn't contain a sampling location add one
6: for all triangles t in S do
7:     h = Footprint(q, o0o1o2o3, G)
8:     for all pixels p touched by h do
9:         for all sampling locations s of p do
10:            q' = CoverageTriangle(q, s, o0o1o2o3)
11:            s.A = s.A ∪ {q'}
12: for all pixels p in G do
13:    for all sampling locations s in p do
14:        V = V ∪ ExactFromPointVisibility(s.A)
15: return V
```

for each of the four corners o_0, o_1, o_2 and o_3 of the view rectangle, for each of three vertices).

Lines 6-11 Stage 2 decomposes the from view rectangle visibility problem into one from viewpoint visibility problem for each sampling location. This is done with another pass over the scene triangles. The triangle footprint is approximated as described for stage one (line 7). For each sampling location s covered by the footprint of a triangle q, a *coverage triangle* q' is computed and stored at s (lines 10-11).

The coverage triangle is defined as follows. The viewpoint translation inside the view rectangle is described with two parameters v and t. The visibility of q changes at s when the projection of an edge of q crosses s. These visibility events occur on lines in the visibility parameter plane (v, t). The three edges of q define three visibility event lines, which define the coverage triangle q' of q. q' defines the v and t values when the triangle projection covers s. q' can also be thought of as the orthographic projection of q with the set of parallel rays obtained by translating the sampling location ray with two degrees of freedom.

The table in Figure 2.6 (left) shows two triangle projections that move over a sampling location s as the viewpoint translates over a view rectangle. The 3D graph

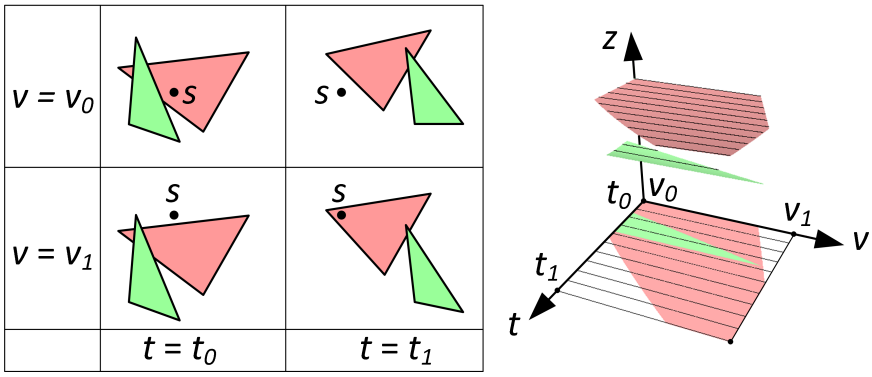

Figure 2.6. Left: two triangles projections moving over sampling location s as the view translates over a view rectangle $[v_0, v_1] \times [t_0, t_1]$. Right: z planes of the two triangles as they move over the sampling location, and 2D visibility sample in the (v, t) plane.

(right) shows the depth z at the sampling location as a planar function of the two translations, for each triangle. The (v, t) plane (bottom of graph) shows the two coverage triangles clipped to the $[t_0 t_1] \times [v_0 v_1]$ domain and depth-composited to define the 2D visibility sample stored at the sampling location.

Lines 12-14 Stage 3 computes visibility at each sampling location s by running our exact from viewpoint algorithm (Section 3.2) on the set A of coverage triangles stored in stage 2. The coverage triangles are already projected so no projection is needed. The visible set is the union of the sampling location visible sets.

2.6 Spherical particles as visibility primitives

We have described our visibility algorithms for scenes modeled with triangles. The algorithms support any geometric primitive that can be tessellated. We have extended our aggressive from viewpoint , from view segment, and over time interval visibility algorithms to support spherical particles directly, without the cost of increasing the number of primitives through tessellation (Figure 2.7 and isosurface in Figure 2.8). The extension has to solve four problems:

(1) deciding whether a particle covers a sampling location, which is done in 3D by checking whether the distance d from the particle center to the sampling location ray is less than or equal to the particle radius r;

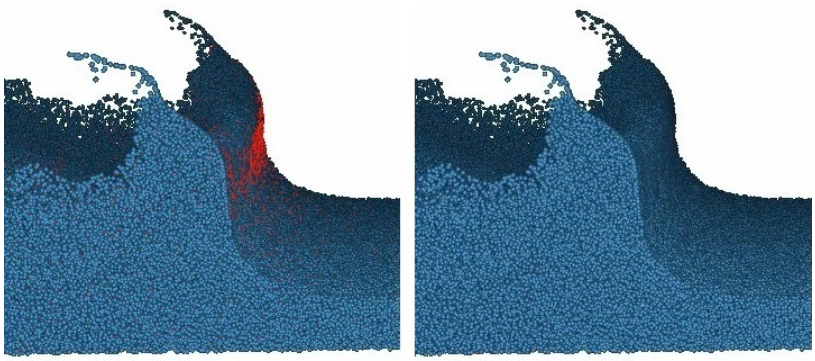

Figure 2.7. Over time interval visibility in a dynamic scene: frame rendered from the particles visible at the time interval endpoints, with missing particles shown in red (top), and correct frame rendered from the visible particles computed by our algorithm directly over the entire time interval (bottom).

(2) approximating the footprint of the image projection of a particle as the viewpoint translates or the particle moves, which is done by computing the bounding box of conservative approximations of the extremal projections of the particle;

(3) finding the visibility parameter values (translation or time) when a visibility event occurs, which is done by solving a quadratic equation that results from setting d equal to r;

(4) finding the centroid of a particle "fragment", which is the projection of the particle center if the projection is inside the pixel, or else the average of the intersection points between the pixel frame and the particle projection.

The exact from-point algorithm cannot be easily extended to handle particles directly because the particle projections have curved edges which complicates visibility subdivision construction.

2.7 Results and discussion

We have tested our visibility algorithms on a variety of scenes: *Manhattan* (4.0 million triangles, Figure 1.2, bottom), *Grass* (55 million triangles, Figure 1.2, top), *Forest* (47 million triangles, Figure 1.2, middle), *Isosurface* (500 million triangles, Figure 2.8), *Water* (2.1 million spherical particles over 80 states, Figure 2.7), *Impact* (2 million triangles over 134 states, Figure 2.8), *Fusion* (500 thousand spherical par-

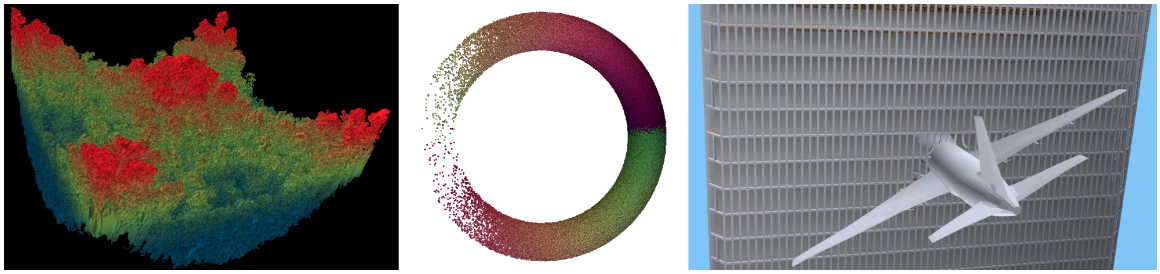

Figure 2.8. Isosurface with 500M triangles *(top)*, Fusion with 500K particles over 100 states *(middle)*, *and* Impact 2M triangles over 134 states *(bottom) scenes.*

ticles over 100 states, Figure 2.8). We organize the presentation and discussion of our results in four subsections: quality, efficiency, comparison to prior art methods, and limitations.

2.7.1 Quality

from viewpoint visibility. The quality of the visible set computed by our aggressive from viewpoint visibility algorithm is summarized in the top three rows of Table 2.1. Visibility completeness (row 1) is defined as the percentage of image area for which the aggressive algorithm finds the visible triangle. Let S^* and S be the visibility subdivisions of the image induced by the aggressive and exact sets. Then visibility completeness is computed as the sum of the areas of the regions in S for which S^* provides the correct visible triangle. Visibility completeness is 98.1% for $Forest$ which has both high depth complexity and small triangle footprint, and 99.7% or better for the other scenes.

We have also estimated the quality of the algorithm by measuring the average and maximum per frame percentages of incorrect pixels over typical paths with thousands of frames. Both the reference image and the output frames have a resolution of $1,280 \times 720$. The errors are small, even though the paths include zooming in. The maximum zoom-in factors for the *Manhattan*, *Grass*, and *Forest* paths were 7×, 17×, and 10×. The few incorrect pixels only occur in between surfaces, which makes for frames that are hard to distinguish from truth frames.

Table 2.1. Quality and efficiency of from viewpoint visibility algorithms.

	Manhattan	**Grass**	**Forest**	**Isosurface**	**Water**
1. Completeness	99.9%	99.9%	98.1%	99.7%	N/A
2. Frame err. (avg)	0.03%	0.11%	2.11%	0.27%	0.09%
3. Frame err. (max)	0.08%	0.20%	3.0%	0.61%	0.13%
4. S.L. / pix (avg)	1.6	5.7	24	209	2.0
5. S.L. / pix (max)	57	1,490	1,245	1,670	27
6. Time A [s]	8.8	47	53	1,677	8.1
7. Iterations	2	3	3	3	N/A
8. Time E-A [s]	5.8	22	1,078	146	N/A

The exact from viewpoint visibility algorithm supports rendering correct frames with any view direction and any zoom factor. We have developed a robust implementation of the algorithm based on a perturbation technique that evaluates control logic predicates correctly and without special handling of degeneracy [70].

We start by perturbing the triangle vertex coordinates by an amount that is negligible in terms of visibility but sufficient to avoid degeneracies. We evaluate predicates with floating point interval arithmetic, which provides an interval that contains the true value of the predicate. The sign of the predicate is determined unless the interval contains zero. We resolve such ambiguous cases by increasing the precision of the interval arithmetic, and thus shrinking the interval, until zero is excluded. The extended precision arithmetic is implemented with the MPFR library [71]. Although it is costly, ambiguity is rare due to perturbation, so the overall efficiency is high.

1D visibility. Our from view segment visibility algorithm is exact for view translations, and our over time interval visibility algorithm is exact as time changes and the view is fixed. Table 2.2 gives the error if instead of using our algorithms one approximates the visible set with the union of the two visible sets computed for the endpoints of the visibility parameter domain (view segment or time interval). The error is given as the average and maximum percentage of incorrect pixels per frame over a sequence of one thousand frames uniformly sampling the visibility parameter domain.

The from view segment algorithm supports any segment length. The longer the segment, the bigger the benefit of the algorithm compared to simply computing vis-

Table 2.2. Percentage of incorrect pixels per frame when the visible set in 1D visibility is approximated with the union of the endpoint visible sets. Our algorithms are exact in these cases.

	From segment endpoints			**From time interval endpoints**		
	Forest	**Grass**	**Manhattan**	**Impact**	**Water**	**Fusion**
Avg.	26%	46%	2.5%	0.093%	0.17%	0.66%
Max.	34%	50%	5.2%	0.34%	0.35%	0.91%

ibility at the segment endpoints, but also the less efficient the algorithm becomes as the same visible triangle is found at a larger number of sampling locations. The over time interval algorithm is run for time intervals that are short enough for the motion of triangle vertices to be approximately linear over the interval. The scenes used here were animated by numerical simulation codes that define these intervals implicitly in between simulation states.

from view rectangle visibility. Table 2.3 reports the quality of the visible set computed by our from view rectangle visibility algorithm. The scene is $Manhattan$ and the output resolution is $1,280 \times 720$. The algorithm is compared to running the exact from viewpoint algorithm from the corners of the view rectangle and union the resulting four visible sets. The error is given as the average and maximum number of incorrect pixels over two paths of ten thousand frames each, which sample the view rectangle. For the first path the view only translates, and for the second path the view translates and rotates. The view-rectangle algorithm is exact under view translation and it has substantially smaller errors than the four-corners algorithm when the view also rotates. The algorithm is also compared to *guided visibility sampling* (GVS), a state of the art visibility algorithm, as discussed in Section 8.3.

2.7.2 Efficiency

The efficiency of our aggressive from viewpoint visibility algorithm is summarized in rows 4-6 of Table 2.1. The number of sampling locations per pixel (rows 4-5) depends on the average image footprint of the triangles, and on the presence of large blockers, and therefore it is small for scenes like $Manhattan$ and large for scenes like $Isosurface$. All running times reported in this chapter were measured by running

Table 2.3. Incorrect pixels per frame for our from view rectangle visibility algorithm compared to computing visibility at the rectangle corners, and compared to prior-art guided visibility sampling (GVS).

	Tanslation only		**Translation + rotation**	
	Avg.	**Max.**	**Avg.**	**Max.**
from view rectangle	0	0	0.01	1
From rectangle corners	202	362	274	948
GVS (10M rays)	805	1,554	643	1,516

parallel implementations of our algorithms on a workstation with 24 2.27GHz X7560 Intel cores. The algorithms were parallelized by tiling the reference image with a uniform 2D grid, by assigning scene triangles to tiles based on triangle footprint, which is computed as described for each algorithm, and by assigning the tile visibility sub-problems to cores in round-robin fashion. The running times for the aggressive from viewpoint visibility algorithm (row 6) is below one minute except for *Isosurface* where the average number of sampling locations per pixel is 209.

The efficiency of our exact from viewpoint visibility algorithm is summarized in rows 7-8 of Table 2.1. The algorithm converges in at most three iterations (row 7), even for the *Grass* and *Forest* scenes which have tens of millions of triangles and complex occlusion patterns. The sampling locations generated based on the visibility subdivision (Figure 2.3) are very likely to resolve the visibility of an undecided triangle t by verifying that t is visible or by finding the triangle that occludes t. The running time (*Time E-A*, row 6) does not include the time for computing the starting aggressive visible set (*Time A*).

The running times of our from view segment and our over time interval visibility algorithms are given in Table 2.4. Interval z-buffering uses a binary search tree and has an $O(n \log n)$ running time for a sampling location with n intervals. For the over time interval algorithm the times given are the sum of all times over all intervals (i.e. 134, 80, and 100 intervals for the *Impact*, *Water*, and *Fusion* scenes).

The running time of our from view rectangle visibility algorithm was 23 hours for the *Manhattan* scene.

Table 2.4. Running times for our 1D visibility algorithm.

	from view segment			**over time interval**		
	Forest	**Grass**	**Manh.**	**Impact**	**Water**	**Fusion**
Time [min]	4.2	25	9.2	6.9	2.5	15

2.7.3 Comparison to prior art methods

We compare the visibility computation capability of our image generalization paradigm to two prior art approaches.

The first prior art approach is the use of conventional images to aggregate an approximate visible set by uniformly sampling the high-dimensional space of visibility parameters. We compared our from viewpoint visibility algorithms to computing visibility with a conventional image of ultra-high resolution of $32 \times 1,280 \times 32 \times 720$. This corresponds to a 32×32 uniform supersampling of the reference image used by our visibility algorithms. Even at the prohibitive cost of 1,024 samples per pixel, the conventional image fails to find all visible triangles: the average pixel errors for the cases used in Table 2.1 are 0.015%, 0.0078%, 0.11%, 0.25%, and 0.0001%. Our exact from viewpoint visibility algorithm produces correct frames. For the *Manhattan* scene our aggressive algorithm yields a 0.03% frame error with 1.6 sampling locations per pixel, compared to the 0.015% error for the high resolution image with 1,024 sampling locations per pixel.

We compared our 1D visibility algorithm to sampling the visibility parameter domain, i.e. the view segment or the time interval. Even when the domain is sampled densely, the quality of the aggregate visible set is low. For example, for the *Forest* (Table 2.3), aggregating the visible set from 40 points on the view segment reduces the average error only to 4.9%, whereas our algorithm yields error-free frames. Similarly, we have compared our from view rectangle visibility algorithm to aggregating visibility from 50×50 conventional images supersampled by a factor of 32×32, rendered from viewpoints that sample the view rectangle uniformly. Again, the dense uniform sampling of the visibility parameter domain failed to find all visible triangles (average and maximum frame errors of 2.5 and 41 pixels). Our algorithms do not search for the visible triangles blindly, through uniform sampling of the visibility parameter

domain, but they rather find the visible triangles directly, by computing the visibility parameter values where visibility changes occur, through visibility event equations.

The second prior art approach to which we compare our image generalization paradigm for visibility is the heuristic sampling of the space of visibility parameters using individual rays. We considered two state-of-the-art algorithms that take this approach:*guided visibility sampling* (GVS) [11], which provides an aggressive solution to from view rectangle visibility, and *adaptive global visibility sampling* ($AGVS$) [14], which provides an aggressive solution for multiple view cells at once.

GVS shoots visibility rays from a rectangle in an iterative process. Early rays guide the generation of subsequent rays based on two heuristics: *adaptive border sampling*, which looks for new visible triangles adjacent to visible triangles that were already found, and *reverse sampling*, which looks for visible triangles in unsampled but accessible space defined by depth discontinuities. The iterative search for visible triangles is terminated heuristically based on a predetermined ray budget or on a lower threshold for the rate of visible triangle discovery.

Table 2.3 shows that when shooting ten million rays, which corresponds to an average of 10 rays per pixel for our 1,280 $\times$ 720 resolution, GVS is outperformed by simply aggregating the visible sets computed at the corners of the rectangle. GVS results in substantially larger visibility errors compared to our from view rectangle visibility algorithm, which defines on average fewer than two sampling locations per pixel, and then computes visibility at each sampling location deterministically. We have also run GVS on the sphere example (Figure 1.1, top), tessellated to 1.3Mtris, and 40 million rays (i.e. 40 rays per pixel) were not sufficient to complete the visible set, whereas our aggressive from viewpoint algorithm finds all visible triangles with an average of 3.18 sampling locations per pixel. In all these comparisons, ray shooting for GVS was restricted to the ray subspace sampled by our algorithms, and therefore all rays were given a chance to contribute to the visibility solution.

$AGVS$ records a visible triangle not just with the view cell that contains the origin of a visibility ray, but with all view cells traversed by the ray until the intersection with the visible triangle. New rays are generated based on the visibility probing outcome of earlier rays and based on proximity to visibility events using a set of heuristics called *adaptive mixture distribution*. Solving visibility for several view cells at once could be done with GVS or with our algorithms and it is an issue orthogonal to the visible set quality comparison at hand. We do compare against $AGVS$ because

Table 2.5. Number of incorrect pixels per frame for the prior approach of heuristic visibility sampling using individual rays, and for our approach.

	GVS(6 faces)	**AGVS**	**Image Generalization paradigm**		
			8 corners	**12 edges**	**6 faces**
Avg.	527	435	21	4.9	0.56
Max.	5,536	3,198	210	124	21

of the *adaptive mixture distribution* heuristics which affect the visible set computed for individual view cells.

Table 2.5 compares our approach to *GVS* and to *AGVS* for a box-shaped view cell for the *Manhattan* scene. *GVS* approximates from view cell visibility by aggregating the visible sets computed by shooting ten million rays for each of the six faces of the box. *AGVS* computes visibility directly for the entire cell using 40 million rays that originate from inside the cell. We rendered ten thousand frames with random viewpoints inside the cell and with random view direction, and we measured the average and maximum number of incorrect pixels per frame. Our paradigm performs substantially better even when we only compute visibility at the eight corners of the view cell. The errors are further reduced when our more powerful visibility algorithms are used for the edges and the faces of the view cell.

2.7.4 Limitations

The current implementation of our aggressive from viewpoint visibility algorithm has a running time quadratic in the number of sampling locations per pixel, as a new sampling location is created only after checking that none of the existing sampling locations is inside the current fragment. This is acceptable when the average number of sampling locations is small, but can become a bottleneck for scenes where the average triangle image footprint is small (e.g. *Isosurface*). In all our experiments the resolution of the reference image where visibility was computed was always the same as the resolution of the output image. Choosing a reference image resolution commensurate with the average triangle footprint will control the average number of sampling locations per pixel. Another option is to subdivide pixels hierarchically, e.g. with a quadtree, to find the sampling locations inside a fragment in logarithmic time.

Our over time interval visibility algorithm relies on the assumption that triangle vertices move with a constant velocity over the time interval, which reduces the complexity of the visibility event equations. The assumption requires subdividing time into many short intervals and running the visibility algorithm for each interval. Generalizing our from view rectangle visibility algorithm to a from view segment *and* over time interval algorithm is challenging. When one visibility parameter is translation and the other one is time, the visibility events occur on curves instead of on lines and so the visibility subdivision becomes nonlinear.

Our algorithms rely on sampling locations to eliminate two visibility parameters. For example, from view rectangle visibility is reduced to solving a from viewpoint visibility problem at each sampling location. However, this comes at the cost of redundancy–the same triangle is found as visible by many sampling locations. The bigger the span of the visibility parameter domain (e.g. view segment, time interval, or view rectangle), the higher the redundancy, which hurts performance.

2.8 Conclusions and future work

We have described a novel approach to visibility based on image generalization. The image is enhanced with sampling locations defined by scene geometry. A small number of sampling locations are sufficient to reveal most visible triangles. Our approach couples a sample-based and a continuous visibility analysis of the image plane to complete the visible set efficiently, in a remarkably small number of iterations. 1D and 2D visibility domains are handled directly, by solving visibility event equations, which reveals visibility changes without trial and error.

So far we have used our visibility algorithms to precompute visibility for complex scenes off-line. One direction of future work is to accelerate our visibility algorithms to support applications where visibility has to be computed in real time, such as antialiasing of minified geometry, motion blur, or soft shadow rendering. One option is to leverage the programmability of current graphics hardware (e.g. through CUDA); another option is to devise hardware extensions that bring native support to rendering over framebuffers with a variable number of sampling locations per pixel, and with more complex sampling locations (e.g. a list of intervals, a 2D map).

A second direction of future work is to use the image generalization paradigm to develop more general visibility algorithms. For example, an exact from view rectangle

visibility algorithm, which also provides exact from view box visibility when applied to the six faces of the box, requires the construction of a two-parameter dynamic visibility subdivision of the image; a quality-guaranteed aggressive from view rectangle *and* over time interval visibility algorithm requires 3D visibility samples, which could be implemented by voxelizing the translation by translation by time visibility parameter volume.

3 ANIMATED DEPTH IMAGE

Remote visualization has become both a necessity, as dataset sizes have grown faster than computer network performance, and an opportunity, as laptop, tablet, and smart-phone mobile computing platforms have become ubiquitous. However, the conventional remote visualization approach of sending a new image from the server to the client for every view parameter change suffers from reduced interactivity. One problem is high latency, as the network has to be traversed twice, once to communicate the view parameters to the server and once to transmit the new image to the client. A second problem is reduced image quality due to aggressive compression or low resolution.

We address these problems by constructing and transmitting unconventional images that are sufficient for quality output frame reconstruction at the client for a range of view parameter values. The client synthesizes thousands of frames locally, without any additional data from the server, which avoids latency and aggressive compression. By generalizing the ray sampling of regular image, we introduce the *animated depth image*(ADI), the ray of which not only samples the color and depth, but also the temporal trajectory of the samples for a given time interval. Sample trajectories are stored compactly by partitioning the image into semi-rigid sample clusters and by storing one sequence of rigid body transformations per cluster. Animated depth images leverage sample trajectory coherence to achieve a good compression of animation data, with a small and user-controllable approximation error. This work has been published on IEEE Transaction on Visualization and Computer Graphics [1].

3.1 Prior work

We review prior work in remote visualization and prior work aimed at overcoming the problem of disocclusion errors.

3.1.1 Remote visualization

We classify remote visualization approaches based on the computational load distribution between client and server. At one end of the spectrum is the approach of doing all the work on the server and of sending visualization frames to the client, which acts like a simple terminal that displays images [72–76]. The approach is appealing because it doesnt require any storage or computation capability at the client, which is particularly beneficial when the client runs on limited hardware such a smartphone [77]. Moreover the approach is general - any visualization algorithm and any type of dataset are supported as long as the server can produce the visualization frames which are displayed at the client. The approach suffers from the disadvantage of limited interactivity due to network bandwidth limitations and latency, which can be addressed by reducing resolution or by aggressive compression. Moreover, the approach implies frequent requests from the client to the server, i.e. once for every change in the visualization path requested by the user, which can lead to server overload, and to poor visualization service quality when connection to the server is lost. The problem has been addressed in the context of virtual environments by anticipating user interactions [78, 79] which however comes at a loss of generality and which is difficult to extend to the context of remote visualization where the visualization target might not be known a priori.

At the other end of the spectrum is the approach of reducing the dataset to a manageable size, to transfer the reduced dataset to the client, and to run the visualization algorithm at the client. There is a large variety of techniques for reducing dataset size, including multi-resolution and level of detail [80, 81], feature extraction [82, 83], progressive refinement [84,85], occlusion culling [86,87], and data compression [88,89] techniques. One technique [90] targets dynamic datasets specifically and reduces the dataset by compressing the trajectories of the simulation nodes (i.e. vertices of finite element geometry) through rigid body decomposition. The strength of the general approach of reducing the dataset at the server is that once the reduced dataset is transferred to the client, the visualization doesnt depend on the network anymore. One weakness of the approach is the need for data reduction algorithms for specific data types and visualizations. Another weakness is the challenge of reducing large datasets aggressively while preserving features of interest that are typically not known a priori.

In the middle of the spectrum are hybrid approaches: most of the work is done at the server while the client also shoulders part of the burden with the reward of improved interactivity. One example are approaches that use sophisticated compression schemes on the server that require decompression at the client in parallel [91,92], or with the help of GPUs [93]. Another example are systems that send enhanced images, or superimages, from the server to the client, such as panoramas [94–101], depth images [23,102,103], multiple center of projection (MCOP) images [104,105], or non-uniformly sampled images [106]. Depth images can be used to synthesize images with point based rendering technique, such as splatting [107,108] or meshing [109].

View-dependent effects such as volume rendering or reflections are challenging for such approaches since it requires either computing the expensive effect at the client, or increasing the size of the representation considerably to include view-dependent color. The visualization by proxy framework [110] succeeds at decomposing and translating a static volume dataset into a compact set of proxy depth and attenuation images that serve as an intermediate representation in the context of volume rendering. Occlusions are addressed using a single-pole occlusion camera [4], which creates multi-perspective proxy images that avoid simple occlusion patterns between a small number of features.

Visualization by proxy provides a general framework where volume rendering operations can be quickly approximated, without accessing the original dataset. The coherent visualization of a time-varying volume dataset without access to the entire dataset, as needed for example in the case of remote visualization, has been proposed using ray attenuation functions [111]. The method has the limitations of not allowing viewpoint changes and of restriction to exploratory use due to approximation errors. Our method focuses on viewpoint changes in opaque surface rendering for dynamic FEA datasets, it handles arbitrarily complex occlusion patterns, and it enforces a user selected error bound on sample trajectory approximation.

The animated depth image method we introduce falls in the category of hybrid approaches. Hybrid approaches are general as far as the client is concerned - like conventional images, superimages insulate the client from the complexity and variety of visualization algorithms and dataset types. Hybrid approaches also improve interactivity - like reduced datasets, superimages are sufficient to reconstruct frames at the client without any additional data from the server. The challenge of the hybrid approach is to devise superimages that can cover a large volume of the multidimensional space of visualization parameters. Previous work was concerned with view rotations

and focal length variations [106] and with viewpoint translations [102]. Animated depth images target the changes in the time parameter for time-varying datasets.

Like the previous method for dynamic dataset reduction through rigid body decomposition discussed above [90], our method compresses animation data by leveraging motion coherence of animated depth image samples. However, the previous method works at dataset level and does not scale with dataset size. Animated depth image are a hybrid remote visualization approach, with cost independent of dataset extent or resolution, and only dependent on output image resolution. In order to achieve this, the animated depth image approach contributes solutions to the problems of fast, hierarchical rigid body decomposition of the animated depth image samples, of adaptive sampling to avoid disocclusion errors due to viewpoint translation and sample motion, and of visualization output frame reconstruction from the animated depth image samples.

3.1.2 Alleviating disocclusion errors

The idea of using a depth image as a rendering primitive dates back to early image-based rendering work [20], However, a single depth image is not sufficientthe slightest viewpoint translation creates disturbing disocclusion errors. Disocclusion errors have been addressed by combining multiple depth images at run time [20, 68, 112] or off-line [22, 23, 113]. Another approach is to render the depth image with a non-pinhole camera model, such as a multiple-center-of-projection camera [105], an occlusion camera [4], a general linear camera [24], or a graph camera [7]. Similar approaches distort the dataset instead of the camera rays in order to eliminate the occlusions [114].

The advantage of the non-pinhole camera approach is that the samples needed for the view region are arranged in a single-layer depth image with good pixel to pixel coherence, which is compact and compresses well. This is of great importance in our context where we aim to reduce the amount of data that has to be transferred from the server to the client. However, constructing non-pinhole camera models that capture all samples needed in the context of the complex occlusion patterns that arise in FEA datasets is challenging. We opt instead for the approach of pre-combining multiple depth images. This suits the remote visualization scenario wellthe work of

rendering multiple depth images and of combining them into a non-redundant set of samples is done at the server, before transmission.

3.2 Animated depth image definition

Animated depth image is similar to regular depth image [115]. Each ray in the ADI is generalized to sample not only a spatial 3D point, but also its trajectory over a time interval continuously. An ADI stores:

(a) *Color and depth samples* to approximate the color and geometry of the dataset

(b) *Sample trajectories* to approximate the motion in the time-varying dataset

(c) *Sample connectivity* to enable a quality triangle-mesh-based reconstruction of visualization frames

(a) Like a conventional depth image, an animated depth image is an image that stores color and depth per pixel, obtained by rendering the dataset for a given view PPC_0 and at a given time step t_0. The pixel data can be unprojected to a 3D point with color using PPC_0. In Figure 3.1, the viewpoint of PPC_0 is E, and pixels a, b, and c are unprojected to 3D points A, B, and C using the PPC_0 rays Ea, Eb, and Ec and the depths z_a, z_b, and z_c stored at the three pixels.

(b) Unlike a conventional depth image, an animated depth image also encodes the trajectories of its samples. Consider a sample A that belongs to a dataset triangle $V_0V_1V_2$ (Figure 3.1). The sample is defined by its barycentric coordinates α, β, and γ:

$$A = \alpha V_0 + \beta V_1 + \gamma V_2 \tag{3.1}$$

As the triangle vertices move to V_{01}, V_{11}, and V_{21}, respectively, the sample moves to A_1 which is found using the sample's barycentric coordinates:

$$A_1 = \alpha V_{01} + \beta V_{11} + \gamma V_{21} \tag{3.2}$$

In complex time varying datasets, such as for example FEA datasets, triangles have complex trajectories modeled with hundreds of time steps. Storing the trajectory of each of individual sample of an animated depth image results in a large data size that precludes applications such as remote visualization. We leverage the local

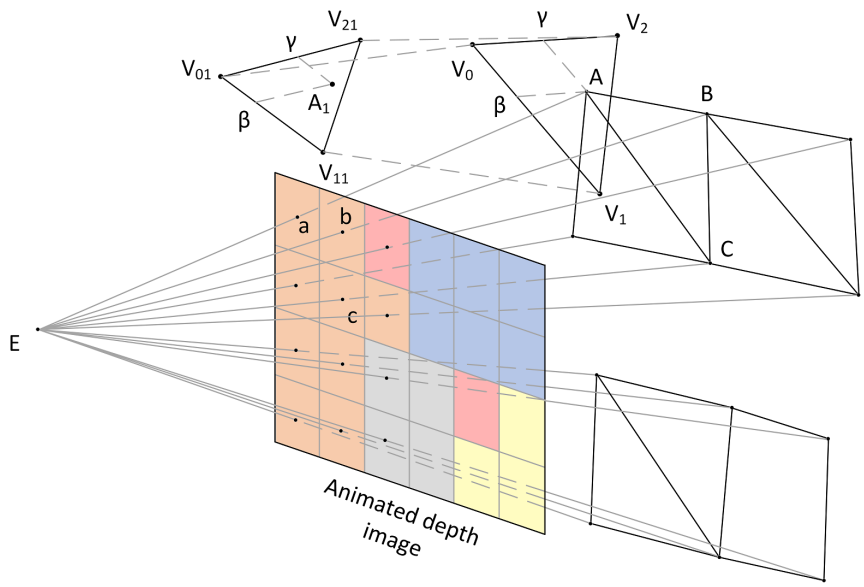

Figure 3.1. Animated depth image illustration.

coherence of motion in time-varying datasets and group nearby animated depth image samples into rigid bodies. A rigid body is a cluster of samples whose motion is approximated well with a single sequence of rigid body transformations X_1, X_2, , X_{n-1}, where X_i is a conventional 4×4 transformation matrix that encodes a rotation and a translation but no scaling, and n is the number of time steps. Instead of storing the trajectory A_0, A_1, , A_{n-1} of each sample in the rigid body, the animated depth image only stores the initial position A_0 of each sample and the sequence of transformations X_1, X_2, , X_{n-1}. The subsequent positions A_i of the sample are approximated with:

$$A_i^* = X_i A_0 \tag{3.3}$$

Consider for example a piece of an axle of the truck in the FEA simulation shown in Figure 1.3(right). When the truck impacts the barrier the piece breaks off and flies away spinning. The samples of the piece can be grouped into a rigid body because their motion throughout the simulation can be approximated with the same sequence of rigid body transformations. The piece can also bend slightly as it breaks off, as long as a user imposed maximum trajectory approximation error is not exceeded. If

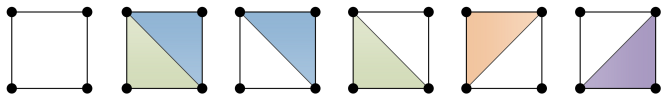

Figure 3.2. Illustration of the six possible connectivity scenarios for a neighborhood of 2×2 samples.

the piece bends significantly and the error threshold would be exceeded when using a single rigid body, the piece is approximated with two or more smaller rigid bodies. In Figure 3.1 there are four rigid bodies highlighted with green, blue, purple and yellow. Not all samples are assigned to rigid bodies (red in Figure 3.1). Such unassigned samples have their trajectory encoded explicitly.

(c) The animated depth image samples are used to reconstruct output visualization frames from novel views. One approach is to resort to a point-based rendering technique that does not require explicit sample connectivity. A high-quality reconstruction approach is to connect samples in a triangle mesh leveraging the connectivity defined implicit by the regular grid of pixels [109]. However, not all four adjacent samples should be connected by two triangles. Like for a conventional depth image, samples should be disconnected if they are on opposite sides of a depth discontinuity: the silhouette samples of a foreground object should not be connected to their neighboring samples that belong to the background object.

Animated depth images also require that samples be disconnected when they belong to surfaces that move apart over the course of the simulation. The animated depth image stores at sample A the connectivity in the 2×2 sample neighborhood that has A as its top left sample (Figure 3.1). The issue of connectivity for the purpose of reconstruction is of course orthogonal to the issue of rigid body decomposition for sample trajectory approximation (Figure 3.2).

3.3 Animated depth image construction

Consider a time-varying dataset D modeled with triangles whose vertices move on piecewise linear trajectories over n time steps from t_0 to t_{n-1}. Given a reference view PPC_0 and a sample trajectory approximation error threshold ε, an animated

depth image of D is constructed in three major steps, with each step computing one of the main components of the animated depth image (Section 3.2):

(a) Compute samples by rendering D at t_0 from PPC_0.

(b) Compress sample trajectory through rigid body clustering.

(c) Compute sample connectivity.

The first step (a) computes a conventional depth image of the dataset by rendering the triangles in D at their t_0 position. The triangle color could originate for example from materials or from false color schemes, and could be encoded for example with a color per vertex or with textures. No matter what the origin of the color or encoding mechanism, the color is transferred to the depth image which stores a color sample per pixel. In addition to color and depth, each pixel stores the index(ID) of the dataset triangle it samples. The triangle ID is used to compute the barycentric coordinates of the pixel sample using Equation 3.1.

3.3.1 Rigid body clustering

The second step (b) of animated depth image construction computes a compact representation of sample trajectories by clustering samples into rigid bodies. This clustering is based on the reasonable assumption that samples that move together like a rigid body are close together in model space, and hence in image space. The sample trajectory approximation error is bound by the threshold ε. The clustering proceeds in bottom-up fashion with the following steps:

(b.1) Seed rigid bodies in 2×2 sample neighborhoods.

(b.2) Merge rigid bodies recursively.

(b.3) Finalize rigid bodies.

Step b.1 takes a pass over the depth image computed at Step (a) and forms initial rigid bodies of 2×2 neighboring samples, whenever possible. Let S_0, S_1, S_2 and S_3 be the four samples of the 2×2 neighborhood. The samples form a rigid body if a sequence of rigid body transformations X_1, X_2, , X_{n-1} places each of the four samples for each time step within epsilon of its true dataset position.

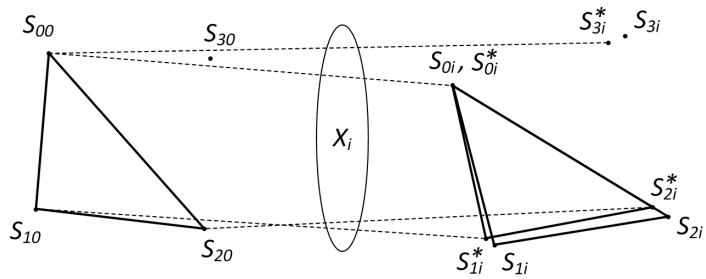

Figure 3.3. Construction of rigid body transformation X_i.

$$S^*_{ji} = X_i S_{j0} \tag{3.4a}$$

$$|S_(ji)^* - S_{ji}| < \varepsilon \tag{3.4b}$$

$$0 \leq j \leq 3, 1 \leq i \leq n-1 \tag{3.4c}$$

In Equation 3.4, S^*_{ji} is the position of sample S_j at time step i as approximated using the rigid body transformations (Equation 3.3), S_{ji} is the true position of sample S_j at time step i as given by the dataset using the barycentric coordinates computed at Step (a) (Equation 3.2), and the Euclidean distance between S^*_{ji} and S_{ij} has to be smaller than ε for each sample j and for each time step i.

We construct the rigid body transformations X_i one at the time, starting with X_1 and ending with X_{n-1}. X_i is constructed using three samples by adapting a previously developed method [116] as shown in Figure 3.3. X_i is constructed by combining a translation that takes the initial position S_{00} of sample S_0 to its position S_{0i} at time step i, with a rotation that aligns the planes of triangles $S_{00}S_{10}S_{20}$ to triangle plane $S_{0i}S_{1i}S_{2i}$, and with a rotation about the normal of the common triangle plane that aligns edges $S_{00}S_{20}$ and $S_{0i}S_{2i}$. Once X_i is constructed, the approximate sample positions at time step i are computed by applying the transformation X_i to the initial sample positions.

The approximation error for S_{0i} is 0 since transformation X_i is constructed such that S^*_{0i} and S_{0i} coincide. If the distance between the true and approximated position of any of the other three samples, including S_3, exceeds ε, the four samples cannot form a rigid body and the iterative construction of the sequence of transformations X_i stops. If all errors are within ε, the algorithm proceeds with constructing trans-

formation X_{i+1}. Once X_{n-1} is constructed, the four samples define a rigid body. Figure 3.4, top, illustrates the 2×2 sample rigid bodies constructed by Step b.1. For the 2×2 neighborhoods where rigid body construction fails, the four samples remain unassigned (white in Figure 3.4, top).

Step b.2 reduces the number of rigid bodies through merging. Merging proceeds in bottom-up quadtree fashion. The rigid bodies of four neighboring nodes at the current level of the quadtree are merged to form the rigid bodies of the parent node at next level up. The inner horizontal and vertical boundaries are traversed one pair of samples at the time. If the two samples of a pair belong to different rigid bodies, the algorithm attempts to merge the two rigid bodies.

Given two rigid bodies A and B, rigid body B could be merged into A if the sequence of rigid body transformations of A approximates the trajectories of all the samples in B within ε. When B is merged into A, the rigid body B is abandoned while the transformations in A will represent all samples in both A and B. Figure 3.4, middle, shows the rigid bodies obtained after Step b.2. The rigid bodies are larger and in smaller number compared to the starting seed rigid bodies (top).

Step b.3 improves the rigid body partitioning of the animated depth image by overcoming limitations of Steps b.1 and b.2. First, Step b.1 only attempts to form a rigid body between the four samples of a 2×2 neighborhood. If the attempt fails, the four samples remain unassigned, whereas, for example, it could be that sample S_0 can be assigned to the rigid body to the left of the 2×2 neighborhood, or even to a distant rigid body. To overcome this limitation, Step b.3 tests each unassigned sample for possible inclusion into each of the existing rigid bodies. Second, Step b.2 only attempts to merge rigid bodies that are adjacent. Step b.3 attempts to merge all pairs of rigid bodies.

Figure 3.4, bottom, shows the final rigid bodies. Some large rigid bodies are formed by merging non-adjacent rigid bodies. As expected, the few remaining unassigned samples are concentrated around the impact region where samples move chaotically. The trajectories of the unassigned samples are approximated using the greedy polyline simplification algorithm of Ramer [117] and Douglas-Peucker [118], conforming to the same error threshold ε.

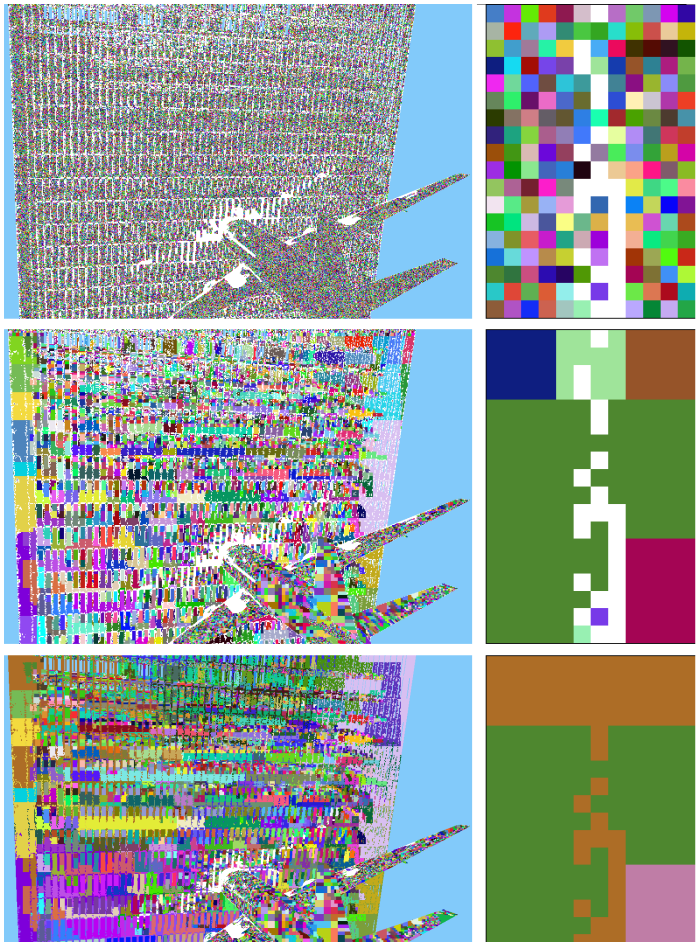

Figure 3.4. Visualization of rigid bodies and magnified fragment after Step b.1 (top), Step b.2 (middle) and Step b.3 (bottom) of the animated depth image construction algorithm. The number of rigid bodies and the percentage of unassigned samples for each of the three images are 125,752 and 20.72%, 12,945 and 20.72%, and 6,322 and 1.45%.

3.3.2 Sample connectivity computation

The third and final step (c) of the construction of the animated depth image computes sample connectivity to enable triangle-mesh-based reconstruction of output visualization frames.

For conventional depth images, connectivity is computed using the second order derivative of the depth map. Values larger than a threshold indicate depth dis-

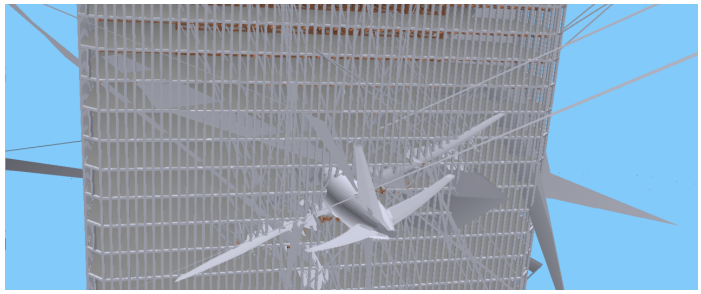

Figure 3.5. Incorrect reconstruction that does not take into account the temporal changes in sample connectivity.

continuities, and reconstruction triangles spanning across depth discontinuities are removed [68]. However, naively using such connectivity data computed at t_0 for all time steps of an animated depth image results in severe artifacts (Figure 3.5) due to sample motion and erosion. When a finite element, e.g. a piece of a structural steel beam, undergoes excessive stress, the element "erodes", i.e. it is eliminated from the FEA simulation for the subsequent time steps. When an element erodes, all the dataset triangles used to represent the element erode as well, as do all samples contributed by the eroding dataset triangles. A reconstruction triangle connecting three samples should clearly not outlive its first eroding sample, but this is not always sufficient.

Consider a structural steel beam in the aircraft impact simulation shown in Figure 1.3. Let's assume that the beam is modeled with six dataset triangles with vertices V_0 to V_7 (Figure 3.6), and let's assume that dataset triangles $V_1V_2V_6$ and $V_2V_5V_6$ erode at time step i. If the beam is far from the viewpoint of the perspective camera PPC_0 used to construct the animated depth image, or if the beam is seen by PPC_0 at an angle, it can happen that neither $V_1V_2V_6$ nor $V_2V_5V_6$ has a sample in the animated depth image. In Figure 3.6, samples S_0, S_1, S_2 and S_3 skip $V_1V_2V_6$ and $V_2V_5V_6$. Simply checking for erosion at the vertices of the reconstruction triangles will lead to the erroneous conclusion that the reconstruction triangles do not erode.

The correct eroding time step of each reconstruction triangle is set as follows. Consider reconstruction triangle $S_0S_1S_2$. For each of its edges, e.g. S_1S_2, compute the shortest path P_{12} between the dataset triangles of the two samples, i.e. $V_0V_1V_7$ and $V_2V_3V_5$, respectively. The shortest path is computed in the dataset triangle adjacency

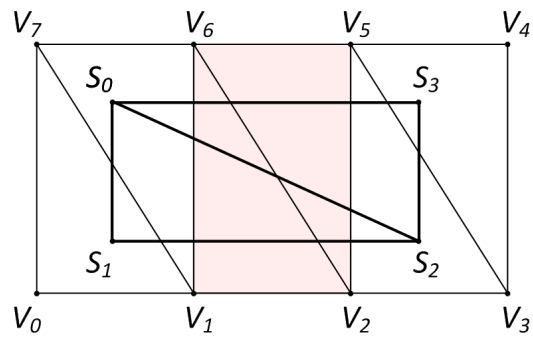

Figure 3.6. Reconstruction triangles $S_0S_1S_2$ and $S_2S_3S_4$ should erode when dataset triangles $V_1V_2V_6$ and $V_2V_5V_6$ (red) erode, even though samples S_i belong to dataset triangles that do not erode.

graph at t_0. The dataset triangle adjacency graph is an undirected graph with nodes corresponding to dataset triangles and with edges corresponding to dataset triangles sharing a vertex. The eroding time step of edge S_1S_2 is set as the first time step where P_{12} is interrupted. A path is interrupted when one of the dataset triangles it enumerates erodes. Finally, the eroding time step of the reconstruction triangle is set to be the earliest of the eroding time steps of its three edges. For the example in Figure 3.6, each reconstruction triangle has two edges that erode at time step i and one edge that does not erode, and thus the eroding time step for the two reconstruction triangles is i.

3.4 Adaptive sampling in space and time

Like a conventional depth image, an animated depth image suffers from disocclusion errors when the viewpoint translates. Moreover, disocclusion errors also occur when sample motion uncovers new samples. We alleviate disocclusion errors by sampling the dataset adaptively from multiple viewpoints and at multiple time steps (Section 3.4.1), and by eliminating redundant samples (Section 3.4.2).

3.4.1 Adaptive sampling

Given an animated depth image ADI_0 with view PPC_0 covering nt time steps starting at t_0, the goal is to enhance ADI_0 with sufficient samples to ensure a dis-occlusion error free reconstruction from anywhere in a neighborhood of PPC_0, and at any time in $[t_0, t_0 + n_t]$. We define the neighborhood of PPC_0 with an equilateral triangle of radius q (i.e. the radius of its circumscribed circle), perpendicular to the view direction of PPC_0, and centered at its viewpoint. The length q is an input parameter. The larger the triangle, the bigger the viewpoint translation range at the client, but also the bigger the data size.

Algorithm 5 Adaptive sampling algorithm

Input: Dynamic dataset D, view triangle R, timer interval T
Output: ADI tiles set S

```
Initialize the set S of animated depth image tiles to empty.
Initialize view parameter queue q with (R, T)
while q is not empty do
    Dequeue the first pair (R_now, T_now) from q
    for all Viewpoint e, time step t in a discretized subset of in (R_now, T_now) do
        Render a depth image DI_et
        Computer non-redundant samples DI_et − S
        S*_et = Tile(DI_et − S)
        S_et = ConstructADI(S*_et)
        S = S ∪ S_et
    if The last processed |DI_et − S|/|DI_et| > g then
        Subdivide (R_now, T_now) and enqueue the subregions
Return S
```

Our adaptive sampling scheme renders conventional depth images from various locations inside the viewpoint triangle and at various time steps. The sampling process stops when the percentage of non-redundant samples contributed by a new depth image drops below a threshold g. In order to enable the elimination of redundant samples, depth images and animated depth images are split into square tiles of size $t_w \times t_w$ (we use $t_w = 4$). Figure 3.7 shows the tiles containing the non-redundant samples contributed by a depth image. Tiles, like complete images, allow storing connectivity information compactly. Given a tile size t_w and a threshold g for the percentage of new samples contributed by a new image, our adaptive sampling algorithm proceeds as follows.

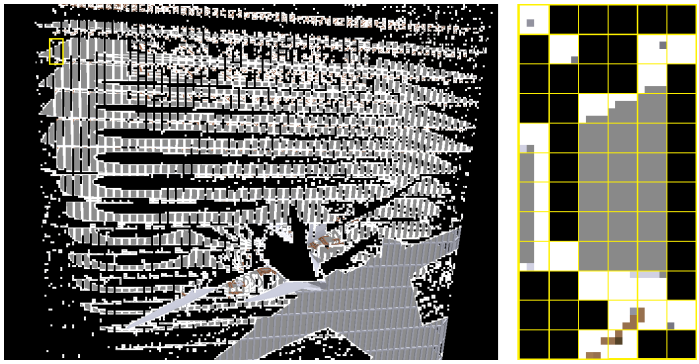

Figure 3.7. Non-redundant samples gathered by our adaptive algorithm.

The algorithm samples the space of possible viewpoints and time steps recursively, in the form of a stack implementation in Algorithm 5. The recursion start from the input view triangle polygon R and time interval T. For each pair of (R, T), a predefined viewpoints subset are used to represent the visibility of this entire region.

In our implementation, we use the three vertices plus the centroid of the view triangle, and the two end points of the time interval. For each viewpoint and time pair (e, t) in this space, a new depth image DI_{et} is rendered at *line 6*. *Line 7* checks for each sample in DI_{et} whether it is redundant with the samples already collected in S. The non-redundant parts of DI_{et} are partitioned into tiles (*line 8*), an animated depth image tile is computed for each depth image tile (*line 9*), the new set of tiles is added to S (*line 10*).

The subdivision continues recursively if the last examined pair (e, t) contributes sufficient number of non-redundant samples (*lines 11-12*). When the algorithm finishes, the set of ADI tiles are returned (*line 13*). The first depth image computed by the adaptive sampling algorithm corresponds to the center of the viewpoint triangle (also the viewpoint of PPC_0) and to t_0. Since S is initially empty, S will contain the entire animated depth image constructed for PPC_0, which guarantees highest-quality reconstructions for the view PPC_0.

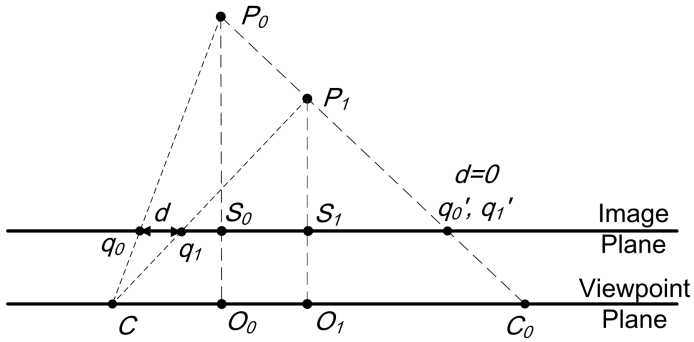

Figure 3.8. Projections of samples P_0 and P_1 onto the image plane as the viewpoint translates on the viewpoint triangle plane. The distance d between the projections is 0 at C_0 where q_0' and q_1' coincide, and it increases away from C_0.

3.4.2 Sample redundancy

The first task is to define sample redundancy. Depth images contain point samples and two samples will in general not correspond to the same 3D point. We define two samples as redundant if and only if they project within one pixel in all views PPC_i, where PPC_i is identical with PPC_0 except that the viewpoint can be anywhere inside the equilateral viewpoint triangle with radius q. If two samples belonging to different surfaces happen to project at nearby locations from a viewpoint, motion parallax separates the two samples when seen from a different viewpoint, and the samples are correctly labeled as non-redundant. Our definition of redundancy ignores view rotations which only introduce a negligible variation of the distance between the projections of two samples. Given two samples defined at different time steps t_a and t_b, the samples are redundant if they are redundant when the second sample is brought to time step t_a.

The second task is to find a method for quickly checking for sample redundancy, given our definition. Figure 3.8 illustrates the projection of a pair of two samples P_0 and P_1 as the viewpoint translates on the plane of the viewpoint triangle. The image plane is constant as there are no view direction rotations. Let C_0 be the intersection between P_0P_1 and the viewpoint plane. For a viewpoint C, the square of the distance d between the image plane projections q_0 and q_1 of P_0 and P_1 is given by:

$$d^2 = |q_0 - q_1|^2 = (w_0 - w_1)^2 \cdot x^2 \tag{3.5}$$

where $w_i (i = 0, 1)$ and x are defined as:

$$w_i = \frac{|q_i - q'_i|}{|C - C_0|} = \frac{|P_i - S_i|}{|P_i - O_i|} \tag{3.6a}$$

$$x = |C - C_0| \tag{3.6b}$$

Since w_0 and w_i do not depend on the current viewpoint C, d_2 is a quadratic function in x, with the minimum value of 0 reached when $C = C_0$. Distance d increases away from C_0. Over the entire viewpoint triangle, the maximum distance occurs at one of the three vertices of the viewpoint triangle. Consequently, given two samples, the largest distance between the projections of the samples over all viewpoints inside the viewpoint triangle can be easily computed as the maximum over the three distances obtained at the viewpoint triangle vertices. If the maximum is less than one pixel, the new sample is discarded as redundant.

3.5 Visualization frame reconstruction

Given a set of animated depth image tiles S, an output image view PPC_i, and time parameter value t_j, the client reconstructs the corresponding visualization frame by rendering each tile in S.

A first step computes the positions at time t_i of each 3D sample stored in the tiles in S. If a sample belongs to a rigid body, the position is reconstructed by applying to the initial position of the sample the appropriate transformation from the rigid body trajectory. If the sample is unassigned, that is it does not belong to a rigid body, the current position is inferred from the trajectory of the sample which is stored explicitly in the animated depth image representation. The time value ti need not be one of the simulation time steps: the simulation can be visualized at arbitrarily slower speeds by interpolating simulation computed sample positions.

Once the current position of the 3D samples is established, the frame reconstruction problem is reduced to the well-studied problem of reconstructing output images with novel viewpoints from input depth images. Any prior work method developed for rendering from depth images can be used, including splats [108], surfels [107], forward rasterization [119], and triangle-mesh reconstruction [20].

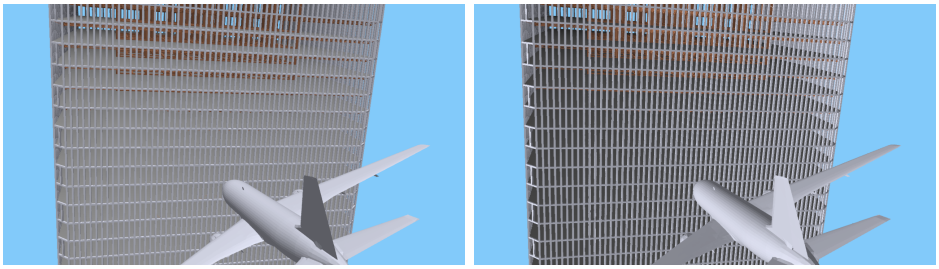

Figure 3.9. Frames with lighting computed at the client.

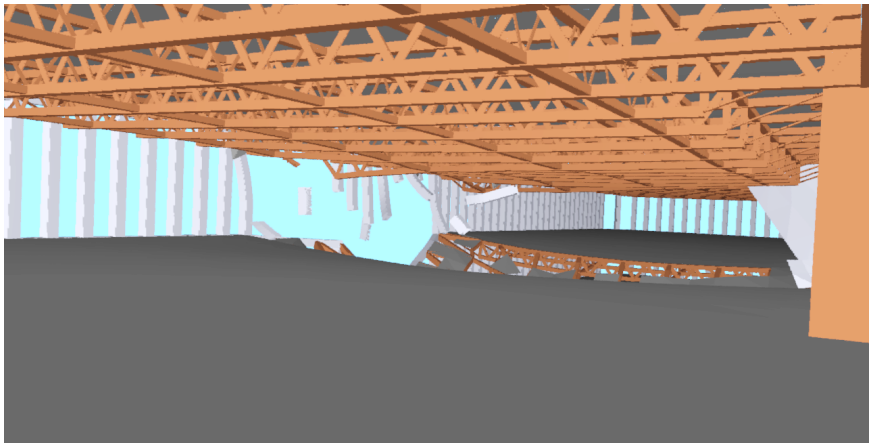

Figure 3.10. Side view of the aircraft dataset.

We use two reconstruction modes: an efficient point-based rendering approach where each sample is rendered with a 2×2 output image pixel splat, and a high-quality triangle-mesh reconstruction. A $t_w \times t_w$ tile defines a mesh of up to $(t_w - 1) \times (t_w - 1) \times 2$ triangles by connecting any 4 neighboring samples with 2 triangles. The connectivity information stored by the tiles is used to avoid defining triangles across depth discontinuities. Samples not connected in any triangle are drawn as points. Depending on the rendering capability at the client, more shading flexibility can be supported by incorporating into the animated depth image additional per-sample shading parameters. We have extended animated depth images to also store per-sample normal, which allows for dynamic relighting at the client (see Figure 3.9).

3.6 Extension to SPH datasets

The animated depth image is a general compact representation of time-varying color and depth datasets. So far we have demonstrated animated depth images in the context of triangle meshes resulting from FEA simulations. Animated should be extendable to other types of time-varying datasets, with certain modifications. Consider, for example, a smoothed-particle hydrodynamics (SPH) dataset where the 3D position of each particle center is recorded over a sequence of time steps. One option for visualizing such a dataset is to render each moving particle as an opaque, shaded sphere. Tessellating each sphere would result in a dataset of moving triangles that can be handled as described. However, tessellating each particle results in a one hundred-fold explosion in the number of primitives. Instead we modify our method to handle particles directly.

Like before, the animated depth image stores color and depth samples. A pixel sample is a reference to its particle and not a 3D pointthe triangle ID is replaced with the particle ID and no barycentric coordinates are needed. Rigid bodies are computed using the center of the particle to which each sample belongs, and not the sample's 3D point. The animated depth image encodes the trajectories of the centers of the particles and not the trajectories of individual samples. Sample connectivity is not needed, as output visualization frames are reconstructed by rendering each particle as an (independent) sphere. The adaptive sampling in space and time remains the same. Sample redundancy detection is now replaced with a simple test for particle ID uniqueness. For the example in Figure 1.3 the SPH dataset was captured using the animated depth image approach. The trajectory approximation error threshold is 1% of the radius of the sphere modeling the particle, the average disocclusion error rate is 0.17%, and the compression factor is 40.

3.7 Results and discussion

We have applied animated depth images to multiple reference views in two FEA datasetsthe truck dataset (Figure 1.3, right) and the aircraft dataset (Figure 1.3 left and middle), as well as to an SPH dam break simulation dataset (Figure 1.3, bottom). The truck dataset has 81 time steps and covers a region of $15m \times 5m \times 3.3m$. The truck dataset contains 0.63M triangles and 0.28M vertices, for a total of 23M vertex positions. The aircraft dataset has 170 time steps, it is segmented into 3 segments

Table 3.1. Data size variation with trajectory approximation error for the aircraft dataset.

ε	**[mm]**	5	10	25	50	100	500	1000
	[%]	0.005	0.01	0.025	0.05	0.1	0.5	1.0
	[pix]	0.04	0.09	0.21	0.43	0.85	4.25	8.50
Size [MB]		39	29	22	19	18	17	16

Table 3.2. Adaptive sampling performance for various convergence threshold g values for the aircraft dataset.

g[%]	**Number of Sampling Depth Images**	**Data Size[MB]**	**Residual Disocclusion Error Rate [%]**	
			Max.	**Avg.**
0.05	208	24	0.31	0.084
0.1	184	23	0.31	0.11
0.2	136	23	0.37	0.14
0.3	124	23	0.46	0.16
0.5	76	20	40.0	0.65

of 58, 58, and 56 time steps, and it covers a region of $110m \times 90m \times 60m$. The aircraft dataset contains 2.08M triangles and 2.01M vertices, for a total of 342M vertex positions. The SPH dataset has 82 time steps, it covers a region of $100 \times 18 \times 20$ and it contains 2.17M particles for a total of 178M particle center positions. The particles are rendered as spheres with radius 0.1.

3.7.1 Quality

We investigate quality along three directions: sample trajectory approximation, residual disocclusion, and reconstruction errors.

Sample trajectory approximation error

Animated depth images approximate sample trajectories with a user-controlled maximum approximation error. Larger error bounds ε lead of course to fewer rigid

bodies, fewer unassigned samples, and a more compact representation. Table 3.1 shows how the data size decreases as the approximation error ε increases for the aircraft dataset. ε is given in absolute values (e.g. 10mm), in approximate relative values (e.g. 10mm/100m = 0.0001 = 0.01%), and in maximum image plane error values (e.g. 0.01 pix). The error in the image plane is estimated by projecting a segment of length ε on the view PPC_0 of the animated depth image. The segment is parallel to the image plane and it is located at the depth of the closest sample in the animated depth image, which provides a conservative upper bound of the image plane error. The image resolution is $1,280 \times 720$.

Residual disocclusion error

Given an output frame F, we measure the residual disocclusion error rate in F as the percentage of samples in the corresponding truth frame F_0 that are not present in F. Table 2 shows the number of depth images the adaptive sampling algorithm uses, the size of the resulting set of animated depth image tiles, and the maximum and average residual disocclusion error rates for various values of the convergence threshold g. The maximum and average residual disocclusion error rates are computed over a visualization sequence of 50,000 frames, which were reconstructed from viewpoints and at time steps that sample the viewpoint triangle and time step interval densely and comprehensively. As expected, a smaller g value yields fewer residual disocclusion errors at the cost of sampling the dataset more. Disocclusion errors are not linear, as seen in the jump of the maximum error rate when the g value changes from 0.3% to 0.5%. The size of the resulting representation decreases slowly as the residual disocclusion error is small (i.e. up to $g = 0.3\%$) and then it decreases rapidly indicating that samples are missed. For all the images shown in this chapter, the value for g is 0.1%.

Table 3.3 reports typical residual disocclusion error rates for the truck and aircraft datasets. The 3 regions of the aircraft dataset that were investigated are shown in Figure 1.3 left (outside), Figure 3.10 (side), and Figure 1.3 middle (reverse). The residual disocclusion errors is small in all cases.

The graph in Figure 3.11 shows the variation of the average and maximum residual disocclusion error rates over all viewpoints as a function of time step for the outside, side, and reverse regions of the aircraft dataset. The maximum graph line for the

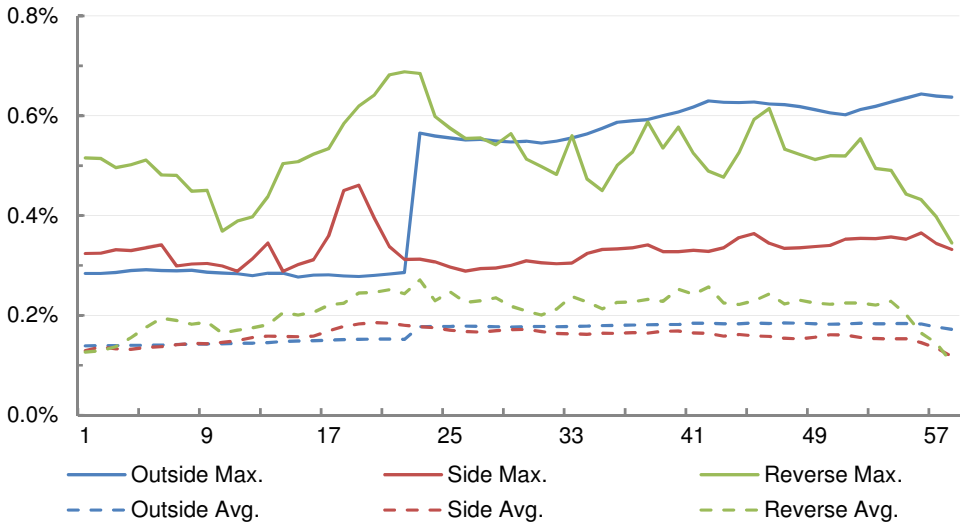

Figure 3.11. Variation of residual disocclusion error rates over time steps.

Figure 3.12. Variation of residual disocclusion error rates over time steps.

reverse region (solid green) varies considerably due to the fast and chaotic motion in that dataset region and the proximity of the reference viewpoint.

Reconstruction error

Even if all samples needed are captured, the reconstructed frame will differ slightly from a frame rendered directly from the original dataset. Figure 3.12 shows that such

differences are small. The largest errors are seen at residual disocclusion errors and at edges. The reasons for the differences include:

- The additional resampling introduced by the intermediate animated depth image representation; the output frame has the same resolution as the input animated depth image, whereas, for conservative reconstruction, the input should have twice the resolution of the output;
- The undersampling caused when the screen footprint of samples increases from the reference view due to view changes or sample animation;
- The conservative early elimination of a triangle between samples that erode at different time steps, as opposed to splitting the triangle into fragments each eroding at a different time.

3.7.2 Performance

Data size

Table 3.4 shows the data size variation for the animated depth image representation as a function of the length q of the viewpoint triangle side. The data was measured for the outside region of the aircraft dataset, the output resolution is $1,280 \times 720$, and the number of simulation time steps is 58. As can be seen in the relative size row, the ratio of the data size to the viewpoint triangle edge length decreases as the viewpoint triangle gets bigger, which indicates that the animated depth image representation is more efficient as the viewpoint triangle grows. We chose a triangular

Table 3.3. Maximum and average residual disocclusion error rates.

Dataset		**Residual Disocclusion Error Rate** [%]	
		Max.	**Avg.**
Truck		0.23	0.05
Aircraft	**Outside**	0.64	0.17
	Side	0.46	0.16
	Reverse	0.69	0.21
SPH		1.1	0.17

Table 3.4. Data size for various viewpoint triangle sizes.

q[m]	1	2	3	4	5	8	10
Size[MB]	16	19	21	25	26	36	42
Rel. Size	16	9	7	6	5	4.5	4
C.F.	17	15	15	14	14	13	13

viewpoint region for simplicitymore complex regions can be built from multiple triangles. Moreover, the adaptive sampling algorithm can be easily extended to more complex 2D or 3D regions (e.g. a cuboid sampled in octree fashion).

The viewpoint triangle is the set of viewpoints used by the adaptive sampling algorithm to capture all samples needed. However, the viewpoint triangle is a conservative approximation of the set of viewpoints from where the animated depth image representation has sufficient samples. Other viewpoints close to the triangle are likely to have sufficient samples, such as points off the triangle plane behind and in front of the center of the triangle, or points on the triangle plane just beyond the triangle. The image in Figure 3.13 shows the viewpoint triangle (solid orange) enlarged (orange triangular contour) and extruded (blue and yellow).

Reconstructions from 66% of the viewpoints inside the prism defined by the blue and yellow triangles have a smaller residual disocclusion error than reconstructions from the viewpoint triangle. Consequently the user can navigate the viewpoint away from the triangle viewpoint, and good reconstructions are obtained even at a considerable distance from the viewpoint triangle (compare the size of the prism to that of the viewpoint triangle in Figure 3.13). When the viewpoint leaves the viewpoint triangle, the user (or the system) can request a new animated depth image representation. Visualization continues using the current representation, with good results, until the new representation arrives from the server.

The last row of Table 4 gives the compression factor achieved by the rigid body decomposition and the compression of the trajectories of unassigned samples. The compression factor is computed by comparison to storing the trajectory of every sample uncompressed, with one position per simulation time step. Tables 3.1 and 3.2 report the variation of the size of the animated depth image representation with the trajectory approximation error threshold ε and with the convergence factor g. For the SPH dataset, the animated depth image representation requires 49.4MB of storage

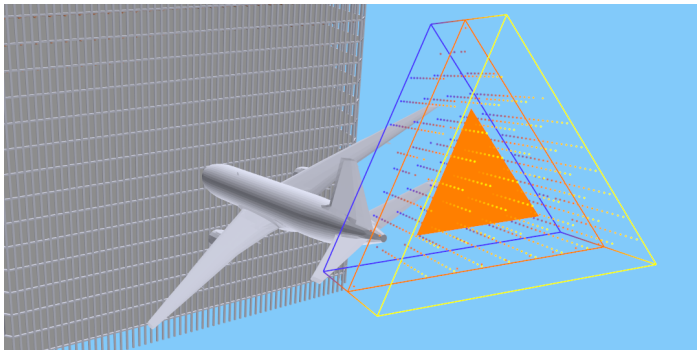

Figure 3.13. Visualization of viewpoints outside of viewpoint triangle (solid orange) with conforming residual disocclusion error.

space, a 41.67 compression factor over the original dataset (2.01GB) that stores each particle position for each time step.

Frame rate

Table 3.5 gives the average rate at which frames are reconstructed at the client from the animated depth image representation. The measurements were performed on an Intel i7 workstation with an nVidia GTX660 graphics card. Four output frame rendering modes are investigated. For static the simulation time step is fixed. For dynamic the simulation time advances from frame to frame. PB corresponds to a straight forward point-based reconstruction with 2×2 splats (Figure 3.14). TM corresponds to triangle mesh reconstruction. The primitives (points and triangles) are sorted in descending order based on their erosion times; this way the primitives needed at a time step are simply determined by choosing the appropriate prefix of the connectivity array without having to enable and disable individual triangles. As expected, higher frame rates are obtained for lower output resolutions, since that implies fewer samples during reconstruction, for the point-based reconstruction mode which is less expensive than the triangle mesh reconstruction, and for the static visualization mode since it does not imply updating the geometry for every frame. For all aircraft dataset experiments at the $1,280 \times 720$ resolution, the minimum, average and maximum frame rates are (98, 145, 212), (13, 18, 23), (16, 28, 40), and (9, 14, 20) for Static PB, Static TM, Dynamic PB, and Dynamic TM , respectively.

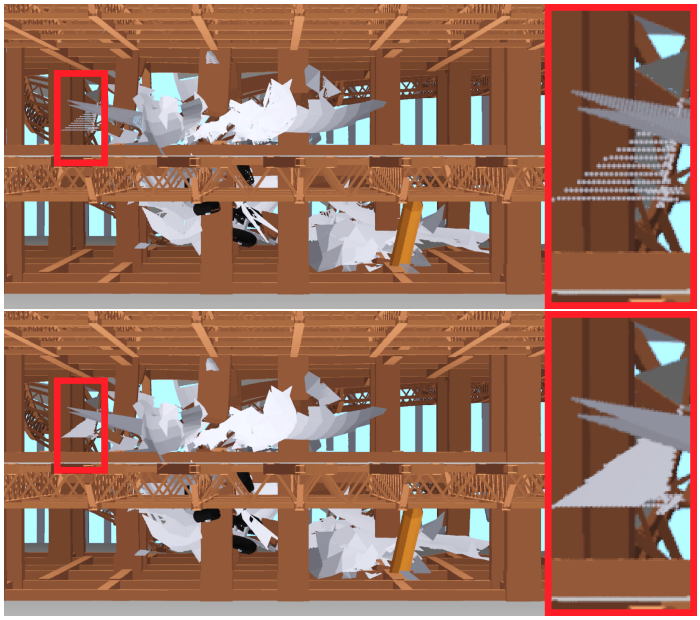

Figure 3.14. Comparison between frames reconstructed using 2×2 pixel splats (left), and using a triangle mesh (right)

Before the client can reconstruct output visualization frames, the animated depth image representation has to be decompressed. Decompression time ranges between 0.4 and 2.5s (Table 3.5), which is comparable to the transmission time in the case of high bandwidth networks, and negligible in the case of low bandwidth networks, as discussed in Section 3.7.2.

Scalability

Animated depth images inherit from conventional depth images the desirable property of cost independence from dataset size. Dataset size depends on two factors: extent and resolution. The animated depth image performs occlusion culling and geometry resampling to achieve cost independence from both factors. Consider an FEA simulation of an earthquake in a city with thousands of buildings. In output views that focus on one or a few buildings, the buildings not visible are culled away. In output views that show the entire city, the resolution of the animated geometry is reduced. The number of samples remains capped by the animated depth image

Table 3.5. Frame rate for various visualization modes.

Dataset	**Sequence**	**Region**	**Frame Resolution**	**Decomp-ression [s]**	**Frame Rate[fps]**			
					Static		**Dynamic**	
					PB	**TM**	**PB**	**TM**
Truck	0-80	N/A	1280×720	0.60	566	106	100	68
Aircraft	0-57	Out.	1280×720	1.4	201	23	38	18
			960×640	0.66	266	31	50	27
			640×480	0.42	480	56	111	45
		Side	1280×720	1.2	111	14	23	12
			960×640	0.88	160	22	31	17
			640×480	0.48	311	46	60	33
		Rev.	1280×720	1.2	135	22	25	15
			960×640	0.85	194	28	56	21
			640×480	0.45	394	74	113	45
	58-115	Out.	1280×720	1.4	212	21	38	20
			960×640	0.65	275	37	49	26
			640×480	0.40	505	57	75	47
		Side	1280×720	1.2	109	13	23	12
			960×640	0.89	157	21	30	15
			640×480	0.51	312	45	59	34
		Rev.	1280×720	1.8	119	16	22	12
			960×640	1.3	165	24	30	17
			640×480	0.72	319	40	53	36
	116-171	Out.	1280×720	1.3	211	22	40	17
			960×640	0.55	298	36	54	28
			640×480	0.32	550	68	114	52
		Side	1280×720	1.1	112	16	24	12
			960×640	0.91	167	22	34	18
			640×480	0.51	315	42	58	31
		Rev.	1280×720	2.5	98	14	16	9
			960×640	1.9	137	18	22	12
			640×480	1.0	270	40	48	26
SPH	0-81	N/A	1280×720	0.33	3280	20	352	20

resolution. Although the cost of a single animated depth image is capped, the adaptive sampling algorithm uses multiple animated depth images to prevent disocclusion errors.

The total number of samples depends on the complexity of the occlusion patterns and on the size of the viewpoint triangle. Although, to the limit, in a dataset with an infinite number of infinitely small particles every two viewpoints gather disjoint sets of samples and thus the number of samples is infinite, for FEA datasets there is

Table 3.6. Relative cost increase as output image resolution changes from 640×480 to $1,280 \times 720$.

	Sequence								
	0-57			**58-115**			**116-171**		
	Out.	**Side**	**Rev.**	**Out.**	**Side**	**Rev.**	**Out**	**Side**	**Rev.**
Samples	2.35	2.92	2.97	2.40	2.90	2.92	2.52	2.97	2.82
Data size	2.13	1.96	2.13	2.17	2.16	2.24	2.56	2.19	2.13
Frame rate	0.38	0.43	0.38	0.29	0.43	0.49	0.32	0.37	0.43

substantial sample redundancy between neighboring viewpoints and the number of samples needed for a given viewpoint triangle is bounded.

The cost of the animated depth image does depend on output image resolution. As the output image resolution grows, a quality reconstruction requires that the resolution of the animated depth images increases as well. In all our experiments the resolution of images increases as well. In all our experiments the resolution of the animated depth images equals the resolution of the output frame.

Table 3.6 reports the relative change in number of samples, in data size, and in reconstruction frame rate as the resolution increases from 640×480 to $1,280 \times 720$, which corresponds to an increase in number of pixels by a factor of 3. Data is provided for each sequence of the aircraft dataset and for each region. The number of samples never increases by a factor greater than 3. The storage size increases with the number of pixels sub-linearly, which is a strong point of the compression based on rigid body decomposition employed by our approach. The higher the resolution, the more coherent neighboring samples are, the more samples per rigid body, and the more effective the compression. The frame rate at higher resolution is typically higher than a third of the frame at lower resolution, which indicates good scalability.

Comparison to other remote visualization approaches

The animated depth image approach enables remote visualization of dynamic datasets and produces high-quality frames that are very close to frames rendered directly from the dataset. Compared to transferring a conventional depth image, our representation requires a larger initial transfer, but then supports changing the view and advancing the simulation time at the client.

We now compare our approach to the conventional remote visualization approach of computing each frame on the server. The performance of a remote visualization system is characterized by three quantities: the startup time t_0, defined as the time it takes for the first frame F_0 to be displayed, the frame to frame latency l, defined as the average time elapsed from when a frame $F_i (i > 0)$ is requested by the user to when F_i is displayed, and the total amount D of data transferred for a remote visualization sequence. To estimate these quantities we have to further define the visualization context.

First, we define how the frame is compressed for the conventional approach. We investigate two scenarios: each frame is compressed individually using jpeg, and each frame is compressed by taking into account that previous frames have already been sent to the client. We approximate conservatively the second scenario by compiling off-line a video file for all the frames from a visualization sequence with the state of the art H.264 codec. This provides an upper bound on the compression performance that live streaming can achieve. For the aircraft dataset, the average per frame data size is 422kB for $1,280 \times 720$ resolution and individually compressed frames, 58kB for $1,280 \times 720$ resolution and streaming. For 640×480 resolution the same numbers are 45kB and 4kB, respectively.

Second, we estimate the ping time between the server and the client, defined as the time it takes a short message to be transferred from the client to the server and back, and the network download bandwidth, defined as the amount of data that can be transferred from the server to the client per second. Upload speed is not a concern since the request for a new frame implies small data amounts. Average ping times from our Purdue University laboratory to various domestic and international servers ranges from 57ms to 276ms. In our comparison we use the values of 50ms and 300ms for a short and a long ping time. The download bandwidths we measured in West Lafayette IN for 4G, 4G LTE, residential broadband, and wide area network (WAN) are 2.5, 10, 20, and 100 Mbps, respectively. In our comparison we use 1Mbps and 100Mbps for high and low bandwidth values.

Finally, we need to define a visualization sequence as the series of consecutive visualization frames requested by the user for a particular region of the dataset and for a particular interval of simulation time steps. In the case of the animated depth image approach, a visualization sequence is reconstructed from the same set of animation depth image tiles. The number of frames in such a visualization sequence depends on

Table 3.7. Comparison between conventional and animated depth image remote visualization for various network scenarios.

Resolution	**Conventional**		**Animated Depth Image**		
	l[**ms**]	D[**MB**]	t_0[**s**]	l[**ms**]	D[**MB**]
Scenario A: ping 300ms, bandwidth 1Mbps					
$1,280 \times 720$	603 - 3,447	580 - 4,220	192	2 - 111	24
600×480	300 - 502	40 - 450	88	2 - 38	11
Scenario B: ping 300ms, bandwidth 100Mbps					
$1,280 \times 720$	300	580 - 4,220	1.92	2-111	24
600×480	300	40 - 450	0.88	2 - 38	11
Scenario C: ping 50ms, bandwidth 1Mbps					
$1,280 \times 720$	478 - 3,322	580 - 4,220	192	2 - 111	24
600×480	56 - 377	40 - 450	88	2 - 38	11
Scenario D: ping 50ms, bandwidth 100Mbps					
$1,280 \times 720$	50 - 58	580 - 4,220	1.92	2-111	24
600×480	50	40 - 450	0.88	2-111	11

the dataset, on the region of the dataset, and on the time interval. We have observed the civil engineers in our project examine each of the outside, side, and reverse regions of the aircraft dataset for over 10 minutes, which at 30Hz implies sequences of 18,000 frames. In our comparison we assume visualization sequences of 10,240 frames.

Table 3.7 gives the performance of the animated depth image remote visualization approach for the aircraft dataset and compares it to that of conventional remote visualization. Four scenarios are investigated: long ping time and low bandwidth (A), long ping time and high bandwidth (B), short ping time and low bandwidth (C), and finally short ping time and high bandwidth (D). For each scenario two output resolutions are investigated. For the animated depth image approach, the data size is estimated by averaging the representation size over the three regions outside, side, and reverse, for each resolution (resulting in the values of 24MB and 11MB). The startup time t_0 is computed by dividing the data size to the bandwidth to obtain the 192s and 88s values.

The frame to frame latency l is computed by inverting the reconstruction frame rate, and it is given as a range, using the fastest and slowest frame rates given in Table 3.5 for the same resolution, and over all reconstruction modes (i.e. 566fps and 9fps for $1,280 \times 720$, and 550fps and 26fps for 640×480). l does not depend on the

network parameters (i.e. ping and bandwidth). In fact the visualization can continue at the client even if the connection to the server is lost after the initial transfer.

For the conventional approach, the total amount of data transferred D is obtained by multiplying the average frame size by the number of frames in the sequence (i.e. 10,240). A frame size range is used, from H.264 sequence compression to compression of individual frames, as discussed above. The conventional approach transfers substantially more data. The breakeven points are 424 and 58 frames for $1,280 \times 720$, and 2,816 and 250 frames for 640×480.

For the conventional approach, the time for the first frame is the same as for any other frame, thus $t_0 = l$. We estimate l as follows:

$$l = max(t_{ping}, \frac{t_{ping}}{2} + \frac{f}{b}) \tag{3.7}$$

Where t_{ping} is the ping time, f is the size of the frame, and b is the bandwidth. If b is sufficiently large for the network to transport a frame in half the ping time, l is given by t_{ping}. In scenario A, the advantage of the animated depth image (ADI) approach over the conventional remote visualization (CRV) approach is substantial, for both resolutions, and even when the highest quality reconstruction is used for ADI and the most aggressive frame compression is used for CRV. In scenario B, the high bandwidth reduces l for CRV to ping time, which still exceeds even the highest quality reconstruction time for ADI. In scenario C, ADI has substantial advantage for the $1,280 \times 720$ resolution. In scenario D, which corresponds to a very highly performing network, ADI has an advantage only for the faster reconstruction modes (i.e. PB static and dynamic, see Table 3.5). For all scenarios, the fastest reconstruction (i.e. 2ms) gives at least a 25 fold advantage for ADI over CRV.

We conclude that, compared to the conventional remote visualization approach of sending each frame from the server to the client, the animated depth image approach improves frame to frame latency in all but in the case of a very highly performing network, and the advantage increases with output frame resolution. Moreover, the frame to frame latency does not depend on the network condition. These advantages come at the cost of a longer startup time. The conventional approach will always be limited by the ping time, a network characteristic which whose improvement is challenging and costly.

3.7.3 Limitations

One of the limitations of animated depth images as of all hybrid approaches to remote visualization is the large startup time. Progressive refinement schemes such as transferring a lower resolution animated depth image for the reference view could help alleviate this problem. Another approach is to further reduce the size of the animated depth image, for example by compressing the color and depth maps. The trajectories of the unassigned samples currently take up to 50% of the overall storage requirement, so further improving the compression of those trajectories will translate in sizable storage gains. Although the amount of residual occlusion errors is small even for the complex occlusion patterns in the dataset regions explored, the adaptive sampling algorithm proceeds nonetheless in greedy fashion. A global optimization approach could be developed to bound residual disocclusion errors.

A second limitation of animated depth images is that they do not support volume rendering. This limitation is inherited from conventional depth images which can model opaque surfaces by capturing the first surface sample seen along a ray, but cannot model transparency. This does not mean that animated depth images cannot be used to visualize opacity data. Visualizations of opacity data often take the first step of computing a surface of interest (e.g. isosurface) which can then be remotely visualized with our method. Volume rendering is just one example of the more general challenge brought to sample-based rendering by view dependent effects. Another example is rendering reflections. One option is to render reflections at the client, rendering capability permitting. We will also investigate the extension of animated depth images to store view-dependent color in a compressed form, leveraging the fact that color variability is limited by the targeted range of reconstruction viewpoints.

Finally, the rigid body decomposition of the set of samples stored by an animated depth image is done non-optimally in the interest of performance. The heuristic used is based on the reasonable assumption that samples whose motion is well approximated by a rigid body transformation are also samples that are close to each other in model space and thus in image space.

3.8 Conclusions and future work

We have described animated depth images, a novel type of image that not only stores color and depth samples but also stores sample trajectories. An animated

depth image covers a time interval as opposed to the single time point covered by conventional depth images. The difference between an animated depth image and a set of conventional depth images comes from the fact that the trajectories of the samples are stored in a compact way that leverages sample trajectory coherence.

The approximation is efficient: tight user-selected error bounds are met while achieving considerable storage savings. The approach does not rely on sample trajectory simplicity, but rather on similarity of trajectories of nearby samples. The approach uses one rigid body transformation per time step which allows modeling complex trajectories with little or no time step to time step coherence, as those arising in the impact and dam break simulations considered in this chapter. As we have shown, despite the complexity of the motion in these simulations, trajectories do exhibit sample to sample coherence. As the spatial resolution of simulations continues to increase, so will the sample to sample coherence and thus the efficacy of our approach.

Compared to a video segment, an animated depth image affords interactivity. We have demonstrated the benefits of animated depth images in the context of remote visualization of FEA datasets, which exhibit complex occlusion patterns. We have shown that the approach can be extended to SPH simulation datasets. Since animated depth images are a general approximation of animated geometry, we anticipate that the approach can be extended to other representations. Like for any type of image, animated depth image size is independent of dataset size and their relative benefit increases with dataset size.

In computer graphics applications such as, for example, urban simulation or games, most of the dynamic scenes used are static where most of the moving triangles are part of explicitly defined rigid bodies (e.g. cars through the city) whose trajectories are encoded collectively with a single sequence or tree of transformations, and the motion of most non-rigidly moving triangles is governed by the motion of underlying skeletons (e.g. computer animation characters). This allows animating the triangles economically by skinning the skeleton for each new configuration.

Animated depth images are not suitable for such scenes. Animated depth images are specifically designed for datasets where millions of simulation nodes move independently resulting in millions of triangles with small screen footprint whose vertices move independently from frame to frame. Such datasets are common in scientific visualization and they cannot be handled with the conventional scene graph,

transformation hierarchy, and skeleton conceptualizations of motion used in computer graphics.

One direction for future work is incorporating animated depth images into an actual remote visualization system, with one or a few powerful servers, and with many thin clients. Output frame reconstruction is equivalent to straight forward triangle rendering, and even smart-phones can render meshes with a number of triangles commensurate to their screen resolution, so we do not foresee problems with rendering performance at the client. One of the potential advantages of our hybrid method is to decrease the frequency of requests from the client to the server, compared to conventional remote visualization where each client makes a request for each frame displayed.

However, the request for a new set of animation depth image tiles is more complex than simply rendering a single frame. In order to minimize the response time for such requests, we will investigate parallelizing the computation of an animated depth image and the adaptive space/time sampling over the set of processors available at the server. Prior work has accelerated the computation on the server side using a light field [120] – the approach could be adapted to pre-compute all animated depth images possibly needed by clients by tiling the viewing space. Another direction for future work is to address other types of visualization, such as volume rendering, computation that is harder to cache in sample sets [121]. Finally, animated depth images introduce sampling robustness with respect to variations of the time parameter of time-varying datasets. In the future we will investigate other image generalizations that bring robustness with variations of other visualization parameters such as, for example, an isovalue used in isosurface extraction.

4 THE FLEXIBLE PINHOLE CAMERA

In this chapter, we present a pinhole camera model that allows modulating the sampling rate over the field of view with great flexibility via ray pattern generalization. This flexible pinhole camera or FPC is defined by a viewpoint and by a sampling map that specifies the sampling locations on the image plane. The sampling map is constructed from known regions of interest with interactive and automatic approaches. The FPC provides an inexpensive 3D projection operation which allows rendering complex datasets quickly, in feed-forward fashion, by projection followed by rasterization.

The FPC supports many types of data, including image, height field, geometry, and volume data. The resulting image is a coherent non-uniform sampling (CoNUS) of the dataset that matches the local variation of the importance of the dataset. We demonstrate the advantages of CoNUS images in the contexts of remote visualization, of focus-plus-context, and of acceleration of expensive rendering effects such as rendering of surface geometric detail and of specular reflections. This work has been published in IEEE Computer Graphics and Applications [2].

Most computer graphics and visualization applications employ images computed using the planar pinhole camera (PPC) model. The PPC is a good approximation of the human eye which makes it uniquely well suited for applications where the goal is to show users what they would see during an actual exploration of the scene. However, there are applications where the reduced field of view, the single viewpoint, and the uniform sampling rate limitations of the PPC model are a severe disadvantage.

In this chapter we address the uniform sampling rate limitation of the PPC model. We introduce the flexible pinhole camera or FPC which allows for adjustments of the sampling rate according to the local importance or complexity of the data imaged. Like the PPC, the FPC is defined by a viewpoint (i.e. center of projection or eye) and an image plane. However, the sampling locations are not defined by a uniform grid but rather by a sampling map that allows shifting sampling locations from one region of the image plane to another.

Figure 4.1. Visualization of sampling map for Figure 1.4.

By generalizing the ray pattern this way, the dataset is no longer sampled uniformly, but rather with higher sampling rate in certain regions while lower rate in the others. As a result, the output image of FPC is a coherent non-uniform sampling (CoNUS) of the dataset.

The CoNUS image in Figure 1.4, left, samples the five faces at a higher rate. The underlying sampling map is shown in Figure 4.1. The ray pattern generalization is optimized to be represented as a sampling map. The sampling map has the topology of a 32×32 regular rectangular mesh which is distorted accordingly to reflect the sampling rate preference.

FPC CoNUS images preserve the advantages of conventional images. A CoNUS image can be computed quickly with the help of GPUs. Data access remains constant time, with the small additional cost of the sampling map indirection. A CoNUS image has good pixel to pixel coherence and conventional image compression algorithms apply. Finally, a CoNUS image remains a single-layer 2D array of samples which defines connectivity implicitly.

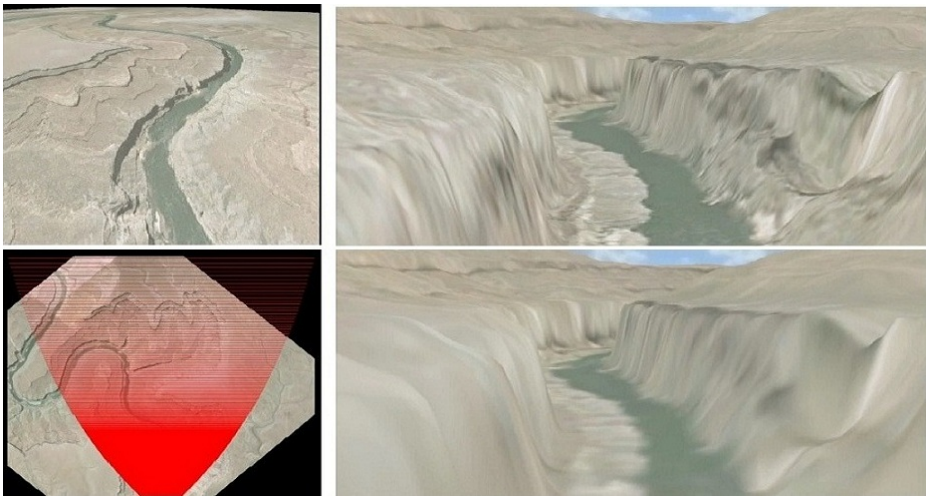

Figure 4.2. CoNUS height field and its sampling pattern (left), output frame rendered from CoNUS (right top), and from conventional height field (right bottom).

4.1 Prior work

We first review prior efforts aimed at removing the uniform sampling rate constraint of conventional images, and then review prior work in the application contexts where we examine the benefits of FPC rendered CoNUS images.

Non-uniform sampling rate Hierarchical spatial partitioning schemes such as kd-trees do improve representation efficiency by stopping subdivision in regions where data is sampled accurately. One could define the FPC sampling map with such a partitioning scheme. Compared to our distorted grid approach (Figure 4.1), the hierarchical approach has the advantage of supporting a wider range of sampling rates, but it has the important disadvantages of sampling rate discontinuity, of lack of contiguity, and of greater construction (i.e. rendering) and usage (i.e. lookup) complexity.

Images with a non-uniform sampling rate were first obtained as a side effect of techniques for removing the field of view limitation of conventional images. For spherical panoramas the sampling rate variation was an unwanted side effect and they were replaced by cube maps with a more uniform sampling rate. Recently, single image panoramas have received renewed attention, as the programmability of

graphics hardware enables sampling patterns that avoid the earlier undersampling problems [122].

Researchers have also been addressing the single viewpoint limitation of conventional images with innovations at the camera model level such as the general linear camera [24] and the occlusion camera [4]. The FPC CoNUS is complementary to these approaches, providing sampling rate flexibility to panoramic and non-pinhole cameras. Irregular sampling patterns have also occurred in the contexts of image-based rendering by 3D warping [20] and of shadow antialiasing [67]. In both cases depth images are reprojected to novel views where the forward mapped samples are irregular.

The granularity with which sampling is controlled in the FPC CoNUS approach is insufficient to sample the shadow map precisely at the locations where the output image does, as needed to completely eliminate shadow aliasing. However, shadow aliasing could be reduced by using a CoNUS shadow map with a higher sampling rate in regions that are magnified in the output image.

The most general pinhole camera defines each ray independently with its own image plane point [106]. Such a camera model has the theoretical importance of maximum generality, allowing for any sampling pattern given n rays, but it has no practical use. First, the rays are not organized in a 2D array and therefore the resulting image is an unsorted list of color samples which cannot be easily displayed. Second, rendering such an image is expensive as it would require tracing each ray independently.

The practical implementation of the general pinhole camera that the researchers use [106] restricts the sampling rate variation to a rectangular region R of the image plane. A smaller rectangle r, concentric with R, provides a higher resolution sampling of the scene at that region of the image. The higher resolution region r is sampled with a planar pinhole camera and hence it is distortion free (i.e. 3D scene lines map to 2D image lines). The region $R - r$ is used to transition from the low sampling rate outside R to the high sampling rate inside r. The sampling locations in $R - r$ are chosen with a quadratic or cubic function to achieve C^0 or C^1 continuity. Several regions of higher resolution are supported as long as they are disjoint.

The FPC amounts to a different specialization of the abstract general pinhole camera model. The FPC CoNUS approach has two fundamental advantages over the previous specialization. First, the FPC provides far greater flexibility in defining

the sampling locations. Second, as explained in the next sections, the FPC sampling map provides fine grain control of the sampling rate while keeping the amortized cost of the fundamental image point distortion and undistortion operations constant. In contrast, a general planar pinhole camera with multiple rectangular regions of high resolution requires checking each of the regions for the distortion / undistortion of a point, which does not scale with the number of regions.

In texture mapping non-uniform sampling has been pursued through compression, atlasing, enhancement with explicitly modeled high frequency features (e.g. edges), and distortion. We only discuss the last two approaches as they are closest to our work. Textures enhanced with edges modeling shadow silhouettes [123] or abrupt changes in color [124] are more robust to magnification. The approach is compatible with CoNUS textures. When edges are derived from vector graphics primitives the edges have to undergo the sampling map distortion (Figure 4.1), and long edges have to be split. For edges derived from the texture, the CoNUS texture can be used directly. Space-optimized textures [125] distort textures with a similar mechanism to our sampling map. Our FPC work extends nonuniform sampling to more types of data and applications.

Remote visualization. As the size of acquired and computed datasets continues to increase, so will the importance of remote visualization which is called upon to provide access to remote datasets for clients with no high-end storage or visualization capabilities. One approach is to reduce the dataset on the server to a size that can be transmitted to and visualized by the client. Many techniques can be used to reduce the dataset size on the server, including data compression (e.g. [88]), feature extraction (e.g. [82]), and level of detail (e.g. [80]). A second approach is to compute the visualization at the server and send images to the client (e.g. [73]). The client only needs a simple terminal that can display images, but network bandwidth limits visualization resolution and frame rate.

A hybrid approach is to transfer from the server to the client images that have more data than what is needed for the client's current frame. Such an enhanced image should be sufficient for a quality reconstruction of a sequence of frames at the client, without any additional data from the server. Images enhanced with per pixel depth [102] and with additional samples at the center [106] have been used to allow translating and zooming in at the client. Remote visualization based on transferring CoNUS images falls in this third, hybrid category. A CoNUS image that samples

known regions of interest in greater detail anticipates the users intention to zoom in on those regions. A CoNUS height field that samples the ground plane orthogonally yet at a higher rate close to the user supports six degrees of freedom navigation at the client in the neighborhood of the current view.

Rendering acceleration using depth images. Depth images are powerful geometry approximations used for acceleration in many contexts including rendering of complex geometric surface detail, of specular reflections, of refractions, and of ambient occlusion. We limit the discussion of prior work to the first two contexts which are used here to illustrate the benefits of our method.

Relief texture mapping is a technique for adding geometric detail to surfaces. The technique produces correct silhouettes and correct interactions between relief and other relief and non-relief geometry (e.g. intersections, casting and receiving shadows) [126]. The relief texture is a depth image attached to a base box. Rendering the box triggers intersecting the eye ray with the depth image at every pixel. The intersection computation is performed by projecting the ray onto the depth image and following the ray projection until the first intersection is found.

Specular reflections are challenging for the feed-forward 3D graphics pipeline because one cannot easily compute the image plane projection of reflected vertices. We group specular reflection rendering techniques into four categories: ray tracing [127], approximations of the projection of reflected vertices (e.g. [128]), image-based rendering (e.g. [129]), and approximations of the reflected scene. We only discuss the fourth category since it is the category where the CoNUS specular reflection rendering method falls. The most drastic approximation is undertaken by environment mapping [130], where the reflected scene is assumed to be infinitely far away from the reflector. Environment mapped reflections are incorrect for objects close to the reflector. Approximating these objects with billboards or depth images [129] improves reflection accuracy. Using CoNUS depth images as relief textures or to approximate reflected objects brings sampling flexibility without a considerable increase of the cost of ray / depth image intersection.

Focus-plus-context visualization. The visualization of complex scenes can benefit from highlighting the scene region that is more important in the context of the application. The pipeline of such focus-plus-context visualization has multiple stages, including finding the regions of interest, finding the best view for a region of interest, and highlighting the region of interest by assigning it a salient color, by assigning

it more pixels, and by managing occlusions through cutaway, transparency, or non-pinhole camera techniques. For example, finding the best viewpoint for a region of interest can be done automatically by analyzing the region feature distribution in an information-theoretic framework [131]. We refer the reader to an excellent survey [132] of the state-of-the-art methods for the various stages of the focus-plus-context pipeline, and we limit the discussion to the problem of highlighting the region of interest by allocating more pixels to the region of interest, which is where the FPC makes it contribution to focus-plus-context visualization.

An important challenge stems from the fact that displays have a uniform pixel resolution (with the exception of special focus-plus-context screens [133]). Consequently, the focus-plus-context image cannot be displayed directly and it has to be mapped to displays with uniform resolution by introducing distortions between the focus and context regions. Focus-plus-context visualization is typically applied to 2D data (e.g. to hierarchies [134], graphs [135], and maps [136]). Applying the technique to 3D data can be done either by distorting the dataset and then visualizing it with a conventional camera [137], or by distorting the camera model [106, 138]. FPC focus-plus-context visualization falls in the second category. Like the general pinhole camera, the *volume lens* [138] defines one or a few regions of interest with higher resolution. The ray perturbation employed does not provide closed form projection and the method is restricted to volume rendering and ray tracing.

4.2 The flexible pinhole camera

4.2.1 Camera model

The goal is to define a camera whose rays pass through a point and that renders an image with a variable sampling rate, i.e. a CoNUS image. The camera has to be flexible, to allow defining the desired sampling rate for sub-regions of the CoNUS image, and it has to be fast, in order to render the CoNUS image quickly from a variety of types of input data. We implement the sampling rate variation with a sampling map that defines a distortion of a regular 2D mesh. The distorted mesh has the same topology as a regular 2D mesh, but with quadrilateral cells that are larger where a higher sampling rate is desired (Figure 4.1). The sampling map is encoded as a 2D array of 2D points. Each point defines a node of the distorted mesh.

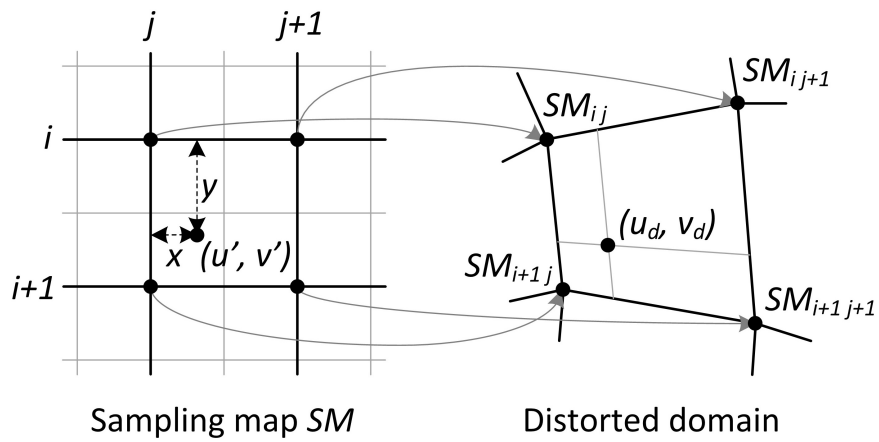

Figure 4.3. Piecewise bilinear image distortion using a sampling map.

Given an (undistorted) image point (u, v), the corresponding (distorted) CoNUS image point (u_d, v_d) is found by looking up the sampling map using bilinear interpolation as shown in Algorithm 6 and Figure 4.3. The input point is first converted to sampling map coordinates (u', v') (line 1). Then the distorted point is computed by bilinear interpolation of the four distorted mesh points stored in the sampling map at the 2×2 neighborhood containing (u', v') (line 2).

Camera model definition. We define the flexible pinhole camera model FPC with a conventional planar pinhole camera PPC and a sampling map SM that distorts the PPC image as described in Algorithm 6.

Projection. A flexible pinhole camera $FPC(PPC, SM)$ projects a 3D point P to its CoNUS image plane by first projecting P with PPC to obtain the undistorted coordinates (u, v) and then by distorting (u, v) to (u_d, v_d) (Algorithm 6).

Algorithm 6 FPC::$Distort(u, v)$ // FPC distortion

Input: undistorted image resolution (w, h), undistorted location (u, v), and sampling map SM of resolution (w_0, h_0)

Output: distorted location (u_d, v_d)

1: $(u', v') = (uw_0/w, vh_0/h)$

2: $(u_d, v_d) = SM.\text{BilinearLookup}(u', v')$

Algorithm 7 FPC::$Rays()$ // Computation of FPC rays

Input: FPC of resolution $w \times h$, defined by PPC and by SM of resolution $w_0 \times h_0$
Output: FPC rays

1: Initialize 2D mesh QM of $w_0 \times h_0$ resolution
2: **for all** (i, j) where $0 \leq i < w_0$, $0 \leq j < h_0$ **do**
3: Vertex coordinates $QM.v_{i,j} = SM_{i,j}$;
4: Texture coordinates $QM.(s,t)_{i,j} = (i/w_0, j/h_0)$;
5: **for all** triangles q in QM **do**
6: **for all** pixels p covered by q **do**
7: $(u, v) = (ws_p, ht_p)$
8: $ray_p = PPC.GetRay(u, v)$

Camera rays. The $FPC(PPC, SM)$ ray through (u_d, v_d) is the PPC ray through (u, v). Consequently, in order to compute the camera ray, one needs to invert the distortion, which poses two challenges. First, one has to find the quadrilateral cell of the distorted mesh that contains (u_d, v_d). A naive approach would examine all quads. A better approach would use a hierarchical subdivision of the CoNUS image (e.g. using a kd-tree or a BSP-tree) to quickly find the quad that contains (u_d, v_d), but constructing the subdivision remains laborious. Second one needs to solve the quadratic equations of the inverse bilinear interpolation that computes x and y from $(u_d, v_d), SM_{ij}, SM_{i+1,j}, SM_{i,j+1}$, and $SM_{i+1,j+1}$.

We bypass these challenges by leveraging two observations. First, CoNUS applications do not need to compute an individual ray of the FPC, but rather all rays iteratively. Second, bilinear interpolation inversion can be avoided by splitting the distorted mesh quads into two triangles and by replacing the quad bilinear interpolation with two triangle barycentric interpolations. This modification does not reduce the sampling rate flexibility of the FPC. We find all rays of the modified FPC efficiently as shown in Algorithm 7.

The rays of the FPC are found by rasterizing the distorted mesh QM defined by the sampling map (line 3). QM has the topology of a 2D regular mesh, but its vertices are displaced according to the desired sampling rate variation (see Figure 4.1). Each distorted mesh vertex carries its undistorted coordinates as texture coordinates (line 4). The ray at the current pixel p is found by first finding the undistorted coordinates (u, v) of p from its texture coordinates (s_p, t_p) (line 7), and then by computing the regular planar pinhole camera ray at the undistorted coordinates (line 8).

The rays are found at the cost of rasterizing the $2 \times w_0 \times h_0$ triangles of the distorted mesh, which is small since the resolution of the sampling map is much smaller than the resolution of the CoNUS image. We compute the rays on the GPU with a trivial fragment shader that executes lines 7 and 8. Algorithm 7 provides the rays of the FPC camera, one at a time, at a small amortized cost. The algorithm is not used as is, but it is rather specialized as needed to render a CoNUS image from a regular image or from volume data, as described below.

4.2.2 Rendering CoNUS images with the FPC

The FPC allows rendering CoNUS images efficiently from a variety of types of input data.

Algorithm 8 FPC::$Render(T)$ // render from geometry

Input: FPC FPC and triangle mesh T
Output: CoNUS image I

```
1: for all vertices v of T do
2:     v' = FPC.Project(v)
3: for all projected triangles t' of T do
4:     Rasterize t'
```

Geometry data. A CoNUS image is rendered from a 3D triangle mesh with the steps shown in Algorithm 8. The 3D triangle mesh T is projected by projecting its vertices with FPC (i.e. PPC projection followed by distortion with Algorithm 6, Section 4.2.1). Then the projected triangles are rasterized conventionally. The projected triangles have to be small enough such that conventional rasterization provides a good approximation of the nonlinear projection induced by the sampling map. Most datasets have small triangles and conventional rasterization is acceptable without further subdivision. When subdivision is needed, an offline approach is preferred in order to avoid the performance bottleneck of issuing a large number of primitives in the geometry shader.

Image and height field data. A CoNUS image is rendered from a conventional input image by modifying line 8 of Algorithm 7 as shown in Algorithm 9. Once the undistorted location (u, v) is known, the input image is looked up to set the current CoNUS image pixel (u_d, v_d). A CoNUS image has fewer pixels than the original image. The original image provides the maximum resolution over the entire field of

view, which is preserved in some regions of the CoNUS image, whereas the other regions of the CoNUS image are at lower resolution. A CoNUS height field sampled orthogonally to the base plane is constructed similarly with the exception that the pixel is setup by looking up the depth in the original height field instead of (or in addition to) looking up the color.

Algorithm 9 FPC::$Render(I)$ // render from image

Input: FPC FPC and image I
Output: CoNUS image I'
// identical to Algorithm 7 except for line 8
...
9: $I'(u_d, v_d) = I(u, v)$ // difference with Algorithm 7
...

Volume data. A CoNUS image is rendered from volume data by tracing the FPC rays through the volume. The rays are determined with Algorithm 7.

4.2.3 Resampling of regular image from CoNUS image

We have described rendering a CoNUS image from a regular image. However, some applications, such as remote visualization, use the CoNUS image as an intermediate representation from which they have to resample a conventional image to be presented to the user. A regular image I_1 is resampled from a CoNUS image I_0 with the steps shown in Algorithm 10. The rays that sample I_1 are defined by a planar pinhole camera PPC_1. Given an I_1 pixel (u_1, v_1), the corresponding CoNUS image pixel (u_d, v_d) is computed in two steps.

Algorithm 10 FPC::$CoNUS2Regular(I_0)$ // Resampling

Input: CoNUS image I_0, $FPC(PPC_0, SM)$, PPC_1
Output: Conventional image I_1 for PPC_1
1: **for all** pixels (u_1, v_1) in I_1 **do**
2: $P = PPC_1.Unproject(u_1, v_1)$
3: $(u_0, v_0) = PPC_0.Project(P)$
4: $(u_d, v_d) = FPC.Distort(u_0, v_0)$ // see Algorithm 6
5: $I(u_1, v_1) = I'(u_d, v_d)$

Figure 4.4. Mass-spring system used to define sampling maps interactively. The user defines regions of higher resolution using a circular brush (yellow).

First, one computes the corresponding point (u_0, v_0) on the image plane of PPC_0, as shown by lines 2 and 3. This correspondence is computed by generating the 3D point P corresponding to (u_1, v_1) by unprojection with PPC_1 and then by projecting P with PPC_0. The unprojection followed by projection can be combined into a single matrix multiplication followed by perspective divides.

Second, the corresponding (u_d, v_d) is computed by distortion leveraging Algorithm 6.

4.2.4 Sampling map construction

We construct sampling maps in one of three ways. One way is through the use of an interactive physics-based 2D mass-spring system. The image is covered with regularly distributed particles connected with springs to form a quadrilateral mesh. All particles have the same mass and all springs have the same resting length (set to 10% of the initial particle distance in our implementation). The user perturbs the system interactively by adding repulsive forces between particles with a circular brush (Figure 4.4).

The force magnitude decreases from the center towards the periphery of the brush exponentially. The equilibrium state is computed by tracking the position of each particle over time until all particle velocity vectors have negligible magnitude. For each time step, the forces acting on each particle are computed first using Hooke's equation for harmonic oscillators $F_i = -kx_i$, where F_i is the force applied to the particle by spring i connected to it, k is the spring constant, and x_i is the particle displacement along the spring direction. Then the particle velocity $\mathbf{v}$ and displacement $\mathbf{x}$ are updated with equations $\mathbf{v} = \mathbf{v} + \Delta t\mathbf{F}/m$ and $\mathbf{x} = \mathbf{x} + \Delta t\mathbf{v}$, where F is the resultant force acting on the particle. A mesh of 256×256 particles is updated at 30 fps and a stable state is reached in less than 2s. The sampling map is defined by the final position of the particles, and it can have a lower resolution than the particle mesh.

Sampling maps can also be generated through a linear combination of the distortion vectors of existing sampling maps, as shown in the equation below. SM_{ij}, SM^0_{ij}, and SM^k_{ij} are element (i, j) of the new sampling map, of the undistorted sampling map, and of the input sampling map k. s_k and t_k are the scale factor and the translation vector of map k.

$$SM_{ij} = SM^0_{ij} + \sum_k (s_k(SM^k_{ij} - SM^0_{ij}) + t_k) \tag{4.1}$$

A third approach is to do away with the discrete representation and define the distortion analytically as shown in Figure 4.2, bottom left, and as described in Section 4.3.

4.3 Remote visualization

The resolution of digital cameras continues to increase faster than network bandwidth. It is also the case that workstation displays now have a lower resolution than the simplest digital cameras attached to cellular phones (e.g. Apple's 4MP 30" LCD and 8MP iPhone 5S camera). Consequently, even if the image is transferred at full resolution, it is most likely going to be downsized for viewing. Often not all pixels in a digital image have the same relevance for the application. For example faces in a portrait photograph are more important than the furnishings in the room. Moreover faces are found automatically by digital cameras for focusing purposes. In the context of an online geographic atlas, pixels sampling famous locations or locations marked by other users as interesting have higher relevance. In the context of remote scientific

visualization, some image regions might be known to be of higher interest to scientists, such as regions showing receptors targeted in drug molecule design.

In such contexts, the CoNUS capability of the FPC could help reduce bandwidth requirements and improve interactivity as follows. The server FPC renders a CoNUS image that samples the regions of interest at a higher rate. Then the CoNUS image is transferred to the client, where it is resampled to a conventional image (Algorithm 10). The application tours the CoNUS image, showing the regions of interest in detail.

We have also investigated the use of the FPC CoNUS approach in the context of remote terrain visualization. Given a height field H at the server and a current view PPC at the client, the goal is to resample H to a CoNUS height field that has all and only the samples needed to provide a quality visualization of the height field from views in the neighborhood of PPC.

Algorithm 11 $HeightFieldCoNUS(H, PPC_0)$

Input: Height field H, client refrence view PPC_0
Output: CoNUS height field CH

1: **for all** samples (u_d, v_d) in CH **do**
2: $\quad (u, v) = PPC_0.Ray(u_d, v_d) \cap H.g$
3: $\quad CH(u_d, v_d) = H(u, v)$

First, a reference view PPC_0 is constructed by enlarging the field of view of PPC, to support view rotations, and by increasing the resolution, to support zooming in and forward translation. Then a CoNUS height field is constructed with a sampling rate that matches the requirements of PPC_0. CH should have more samples close to the viewpoint and fewer at a distance, as illustrated in Figure 4.2, bottom left. We construct the CoNUS height field CH with an analytical distortion function as described in Algorithm 11.

The CoNUS height field sample (u_d, v_d) is looked up in the original (undistorted) height field H at location (u, v) which is computed by intersecting the ray at pixel (u_d, v_d) in the client view PPC_0 with the ground plane $H.g$ of H. This construction applies the perspective foreshortening of PPC_0 while maintaining the orthogonal sampling of H, which avoids disocclusion error problems that would occur if one actually rendered the geometry of H from PPC_0. CH is sent to the client where it is transformed in a 3D triangle mesh that is rendered for each frame. A CH sample is converted to a 3D triangle mesh vertex by computing the ground plane point P

corresponding to (u_d, v_d) (line 2 in Algorithm 11) and by offseting P by $CH(u_d, v_d)$ above the ground plane.

Quality The CoNUS image shown in Figure 1.4 allows rendering all five faces in great detail. The CoNUS height field produces frames that are comparable to frames rendered from the original high-resolution height field (Figure 4.2).

Performance For Figure 1.4, once the FPC model is known, rendering the CoNUS image takes negligible time; the FPC sampling map was designed interactively using the spring-mass system. For the example in Figure 4.2, we use a CoNUS height field of $1,024 \times 1,024$ resolution, which is rendered and used at over 400 and 100 frames per second, respectively.

Limitations The FPC CoNUS approach increases the sampling rate of the regions of interest at the expense of the rest of the image. When high frequencies are present outside the regions of interest, the undersampling can become noticeable (Figure 4.7). The approach does not address occlusions. Whereas occlusions do not occur for images or orthogonally sampled height fields, the FPC CoNUS approach will have to be integrated with an occlusion alleviation scheme such as a non-pinhole camera to support six degrees of freedom remote visualization of general 3D data.

4.4 Depth image rendering acceleration

A depth image is a powerful method for approximating geometry: the depth image is computed quickly with the help of graphics hardware, and a depth image can be quickly intersected with a ray. Because of these important advantages depth images have been used to accelerate the rendering of complex effects such as specular reflections, refractions, ambient occlusion, and relief texture mapping. Eliminating the uniform sampling rate constraint of conventional depth images using the FPC CoNUS approach could benefit all these techniques provided that the efficiency of depth image construction and of ray intersection is preserved. CoNUS depth images can be rendered efficiently from height field or geometry data using the FPC as discussed in Section 4.2.

A conventional depth image DI is intersected with a ray by projecting the ray to the image plane of DI and by tracing the projection with one pixel steps until an intersection is found [126]. In the case of a CoNUS depth image, the projection of the ray is no longer a line segment, but rather a curve segment. The ray cannot longer be

Figure 4.5. CoNUS depth image emphasizing all 4 engraved tablets (left top), scene setup (left bottom), and reflection details rendered with CoNUS (middle) and conv. (right) depth image.

projected solely by projecting its endpoints. Instead, the ray has to be subdivided into segments, each projected with the FPC as described in Section 4.2. The fundamental advantage of depth images of 1D intersection with a ray is preserved, at the cost of a slightly more complicated projection of the ray. We integrated CoNUS depth images into relief texture mapping and into specular reflection rendering, where the CoNUS depth image is intersected with eye rays and with reflected rays, respectively.

Quality. The sampling flexibility afforded by CoNUS depth images allowed improving the clarity of the engraved tablets (Figure 1.4) and of their reflection (Figure 4.5).

Performance. For both conventional and CoNUS depth images, the performance bottleneck for relief texture mapping and specular reflection rendering is the depth image / ray intersection computation. Intersecting a ray with a CoNUS depth image brings the additional cost of distorting a 2D point at every step along the ray. However, CoNUS distortion is fast, and we measured an average frame rate penalty of only 5%. For applications where the CoNUS depth image is intersected with a large number of rays, it might be advantageous to undistort the CoNUS depth image at

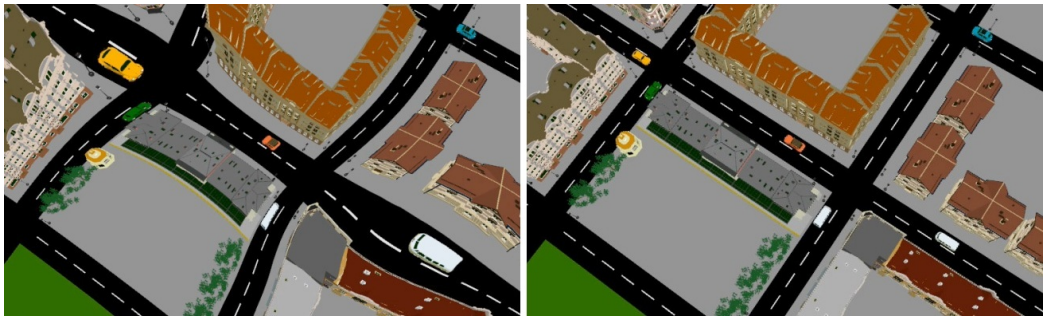

Figure 4.6. CoNUS focus-plus-context visualization emphasizing the yellow and white cars (top), and conventional image (bottom).

the client to a higher resolution conventional depth image using Algorithm 10, which results in straight ray projections and avoids the cost of per step distortion.

Limitations. CoNUS depth images inherit the occlusion limitations of conventional depth images. The sampling tradeoff can lead to visual artifacts outside regions of interest.

4.5 Focus-plus-context visualization

The FPC CoNUS approach is well suited for focus-plus-context visualization because it offers good control over the sampling rate, which allows precisely designing one or multiple focus regions, and because CoNUS images can be rendered quickly, which supports dynamic scenes and the interactive change of focus region parameters. The CoNUS image is shown directly to the user thus no decoding is needed. The CoNUS image can be rendered efficiently from a variety of data as described in Section 4.2. The only remaining challenge is sampling map construction.

Unlike for the previous applications of CoNUS images, in focus plus-context-visualization the sampling map has to be constructed online, once for every output frame, which precludes the use of the mass-spring approach. We construct sampling maps by composing canonical circular sampling maps, one for every focus region. We demonstrate the approach in the context of volume rendering (Figure 1.4), where the user manipulates focus region and view parameters interactively to examine a volume dataset, and in the context of a city scene modeled with triangle meshes (Figure 4.6),

where focus regions track moving cars. The focus region location is computed by projecting the center of the tracked car in the output view.

Quality. The CoNUS approach enables high quality focus-plus-context visualization for a variety of data types. The focus regions have strong magnification and low distortion. Focus region parameters can change and focus regions can merge and then separate again without abrupt changes in the output visualization. Focus plus context visualization is particularly robust to undersampling outside the focus region–users are likely to focus on the region that they themselves selected as important, and focus regions can be shifted interactively to visualize any region in more detail.

Performance. In our experiments FPC volume rendering was on average 7% slower than conventional volume rendering. The cost of volume rendering by ray casting is dominated by the traversal of the volume, thus computing the perturbed rays for the CoNUS approach has no impact on performance. We attribute the slight performance decrease to a larger output image footprint for the distorted volume, and to more rays being focused on the center of the dataset where volume traversal distances are longer. The vertex distortion performed when rendering CoNUS images from triangle meshes had no measurable performance impact.

Limitations. Since the CoNUS approach does not alleviate occlusions, tracked objects of interest can become hidden and the user has to change the view to reveal the object. As future work we will examine changing the view automatically to keep the tracked object visible.

4.6 Conclusions and future work

We have presented a general method for removing the uniform sampling rate constraint of conventional images. CoNUS images can be rendered efficiently from image, height field, geometric, and volume data. Like a conventional image, a CoNUS image has a single layer and good pixel to pixel coherence, thus conventional image compression algorithms can be readily applied. The underlying sampling map can be constructed from known regions of interest in a variety of ways including using a mass spring system, by composing multiple input sampling maps, and analytically.

The sampling map is a powerful tool for assigning more pixels to some regions of the image plane. For example, for the image in Figure 1, the maximum sampling rate increase is 8.13×, respectively, which was measured by finding the largest quadrilat-

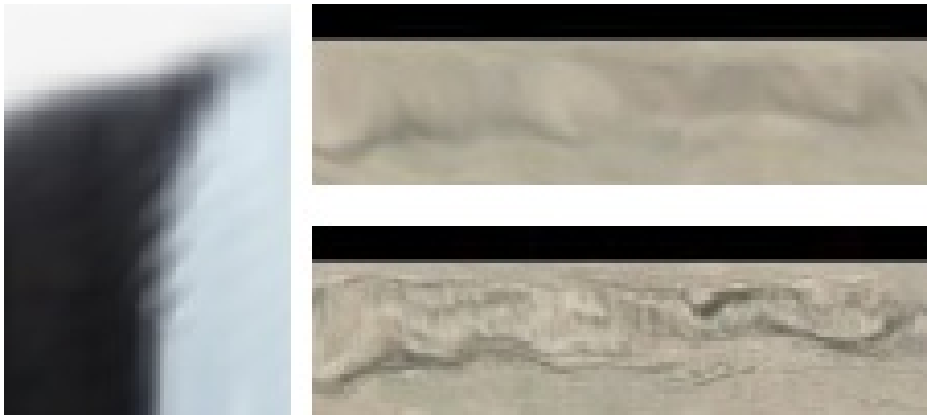

Figure 4.7. Sampling artifact outside the regions of interest in a frame reconstructed from the CoNUS image in Figure 1 (left), and undersampling of distant mountain by CoNUS height field (right, top) compared to original height field (right, bottom).

eral cell of the sampling mesh, and by dividing its area to the area of an undistorted cell. The sampling map does not create new pixels–the sampling rate is increased by decreasing the sampling rate in other regions deemed of lesser importance.

For a sampling map of resolution $w_0 \times h_0$, for regions of interest occupying k cells, and for a minimum sampling rate of the context regions of $c\times$, the upper bound for the sampling rate increase is $z = w_0 h_0(1-c)/k+c$. For example, if $w_0 \times h_0 = 1,024, k = 64$ and $c = 1/2$, then $z = 8.5\times$. If the application tolerates downsampling the context to $1/8$, z increases to $14.125\times$. If there is a single region of interest that fits in one cell, i.e. $k = 1$, then, even for a negligible downsampling of the context regions by $c = 0.95\times$, the sampling rate of the region of interest can reach $z = 52.15\times$.

Possible directions for future work include exploring other uses of CoNUS images (e.g. geometric simplification, acceleration of additional rendering effects), investigating the benefit/cost tradeoff of higher order interpolation of the sampling map to achieve C^1 sampling rate continuity, and developing automatic sampling map constructors. This chapter describes how to sample at a non-uniform rate.

We are particularly interested in tightly coupling the FPC CoNUS approach with automatic techniques for determining what to sample in more detail, such as automatic geometric complexity analysis, object recognition, eye tracking, and saliency

maps. We foresee that FPC-rendered CoNUS images will have wide applicability as they are compatible with virtually all contexts where images are used.

5 THE CURVED RAY CAMERA

Conventional images are rendered with the planar pinhole camera and only capture dataset samples to which there is a direct line of sight. Multiperspective rendering is an approach for extending the visibility computation capability of images while maintain the coherent, single-layer structure of conventional images. A multiperspective image integrates samples from multiple viewpoints seamlessly into an image stored as a regular 2D array of pixels that can be displayed conventionally. This opens up the door to using the multi-viewpoint visibility solution directly in an application by showing the multiperspective image to the user, and not just to use the visibility solution as a data subset from which to render conventional output images.

One powerful method for rendering multiperspective images is to generalize the geometry of a camera ray, a method adopted by our image generalization paradigm. The rays of an initial planar pinhole camera are bent to go around occluders and to reach geometric primitives that are hidden from the reference viewpoint of the initial camera. A ray is not a straight line anymore, but rather the set of 3D points that project at a given image plane location. Since the initial camera has millions of rays and since each rays geometry can be modeled with many degrees of freedom, the camera model design space is sufficiently broad to provide good visibility sampling for the most complex of datasets. Unlike for conventional images, the camera model depends on the reference viewpoint, on the dataset, and on the application.

The design of the camera model usually enforces two constraints. One constraint is to design the set of rays such that the resulting camera model provides a fast projection operation. Given a 3D point, the multiperspective image projection of the point should be computable in a small number of steps. Fast projection enables rendering the multiperspective image efficiently in feed-forward fashion. The second constraint is to avoid that the rays of the camera intersect. More precisely, we want to avoid that the same region of the dataset be sampled by multiple non-adjacent ray bundles, where a ray bundle is a set of rays originating from a contiguous region of the image. Two ray bundles are adjacent if their image regions are adjacent. Sampling a data subset multiple times leads to image redundancy.

Prior work on camera model design for multiperspective rendering has produced the family of *occlusion cameras*. An occlusion camera captures not only samples visible from the reference viewpoint, but also samples that are not visible from the reference viewpoint but are visible from nearby viewpoints. The occlusion camera generalizes the viewpoint to a view region centered at the reference viewpoint. Another prior work camera model for multiperspective rendering is the graph camera. A graph camera is literally a graph of planar pinhole cameras. The graph camera is constructed from an initial planar pinhole camera through a series of bend, split, and merge operations. The resulting camera model has the ability to reach deep into the dataset to reveal distant data subsets.

A graph camera image integrates multiple viewpoints into a continuous multiperspective image. The image is a seamless collage of conventional planar pinhole camera sub-images. Each sub-image is distortion free, with dataset lines projecting to sub-image lines. However, an important limitation of the graph camera image is that data subsets that cross from one sub-image to the next appear distorted. The distortion is due to the fact that the graph camera rays transition from one viewpoint to the next abruptly as they reach the planar boundary between the sub-frusta of the two viewpoints. In Figure 1.6 left, the yellow car crosses the boundary from one sub-frustum to the next. Half of the object is imaged from one viewpoint, and half is imaged with the next viewpoint. As a result, the yellow car appears to be distorted and broken into two hinged parts.

We alleviate this distortion artifact of graph camera images by transitioning from one viewpoint to the next gradually, over a transition region. The transition region turns the rays of one viewpoint into the rays of the other viewpoint progressively. The rays are curved at the transition region. The large ray curvature all but eliminates the distortion of objects located in the transition region (Figure 1.6, right). The rays are modeled with conic curves, which are the simplest curves that provide C^1 continuity with the ray segments at both ends, and that provide fast projection. We call the resulting graph camera variant a *curved ray camera* (CRC), since the rays are not piecewise linear anymore, but rather a sequence of segments interconnected by arcs. Given a 3D point inside the transition region, the curved ray that passes through the point, and thereby the graph camera image projection of the point, is found with a closed-form expression. The CRC rays do not intersect within a planar pinhole camera or a transition subfrustum, and subfrusta are constructed to not intersect.

We have developed three CRC constructors. The first constructor relies on the user to define viewpoints and transition regions interactively. The second constructor tracks a targeted data subset of interest that it maintains disoccluded. When the reference viewpoint or the target move, occluders can hide the target in a conventional image. The target tracking CRC constructor bends the rays of the camera to find a path to the target. The resulting CRC image shows the target. The third constructor takes an input path and conforms the CRC rays to the path. The resulting CRC image provides a preview of the dataset beyond the turns of the input path. The CRC supports any type geometry data, including triangle meshes, particles, or height fields, as well as volume data.

5.1 Prior work

The multiperspective rendering prior work most relevant to the curved ray camera is the general linear camera [24], the occlusion cameras [4–6], and the graph camera [7].

5.1.1 The general linear camera

The general linear camera(GLC) is a camera that interpolates linearly in between three input construction rays [24]. When the construction rays are concurrent, the GLC is a pinhole. When the construction rays are not concurrent, the resulting camera is a non-pinhole whose rays gradually change from one construction ray to the other. The camera model is defined using two planes. The construction rays define the three vertices of a triangle on each plane. The rays of the GLC are obtained through linear (barycentric) interpolation over the two triangles. A triple of barycentric coordinates defines two corresponding points in each triangle, which define a GLC ray. The GLC has been used to model the rays reflected off a curved surface. If the reflective surface is modeled with a triangle mesh, each triangle vertex has a reflected ray, and the rays reflected by the triangle are approximated well with a GLC.

When the two planes parameterizing the rays of the GLC are parallel, GLC projection is linear. However, parallel planes do not enforce continuity between two adjacent GLCs. In the reflection example, at the shared edge of two adjacent reflector surface triangles, the reflected rays generated by one triangle are not the same as those generated by the other triangle. Subsequent work on devising camera models that

interpolate in between given rays has produced the k-ray family of cameras, where k is the number of construction rays [139]. The limitation of discontinuity between adjacent GLCs was eliminated with a 3-ray camera that interpolates in between three rays of unit length, and not in between rays of length defined through intersection with parallel planes.

The GLC and k-ray cameras are suitable when the set of rays whose image one wants to compute is available, as it is for example in the case of specular reflections. These cameras however do not provide constructors that define the rays in order to solve visibility. Moreover, a single GLC or k-ray camera doesnt have the ray modeling flexibility needed to achieve disocclusion in complex datasets.

5.1.2 The occlusion cameras

The occlusion cameras are a family of cameras that shrink the occlusion shadows of occluders. Given a reference conventional planar pinhole camera, the occlusion camera detects occluders in its frustum and then routes rays to capture some of the samples that are hidden but that are close to the silhouette of the occluder. Such samples are likely to be needed from viewpoints close to the reference viewpoint. All occlusion cameras provide closed-form projection, and the occlusion camera rays do not intersect which makes the occlusion camera images non-redundant. The occlusion cameras differ on how the non-concurrent rays are constructed.

The single-pole occlusion camera [4] is a non-pinhole whose rays are obtained from the reference planar pinhole camera by bending the rays in, towards a pole in the image. The pole is the image projection of the centroid of the occluder. The ray distortion is equivalent to displacing the dataset samples captured by the rays away from the pole. The displacement depends on the depth of the sample. The deeper the sample, the more it is displaced. This way, hidden samples near the silhouette of the occluder will move more than the occluding samples and will become disoccluded in the occlusion camera image. Consider a sphere in front of a background. A single-pole occlusion camera of the sphere will use the projection of the sphere center as a pole, and the resulting image will show more of the background around the sphere than visible in a conventional image from the reference viewpoint. Moreover, the occlusion camera image will also show more of the sphere by effectively receding the silhouette line on the sphere. The single-pole occlusion camera is limited to a single occluder.

The depth discontinuity occlusion camera [5] allows designing the rays with greater flexibility than the single-pole occlusion camera, in order to support more complex occlusion patterns. Construction starts from a z-buffer rendered conventionally from the reference viewpoint. Depth discontinuities are detected in the z-buffer, and then the depth discontinuities are used to define a distortion map that moves samples perpendicularly to the depth discontinuity, and away from the near surface. The distortion map allows defining distortions with far greater control compared to the single pole that can only define a single radial distortion for the entire image. The depth discontinuity occlusion camera extends the reference viewpoint to a view region. However, depth discontinuity camera construction does not allow the application to specify a view region for which the occlusion camera image should contain sufficient samples.

The epipolar occlusion camera [6] generates a multiperspective image that samples what is visible from a view segment. Unlike the previous occlusion cameras, epipolar occlusion camera construction takes a set of viewpoints as input and builds a camera accordingly. Epipolar occlusion camera construction leverages that when the viewpoint translates on a segment, occlusion/disocclusion events are confined on epipolar segments. This reduces the dimensionality of the construction from 3D to 2D: construction considers one epipolar plane at the time. The resulting epipolar occlusion camera image is coherent along rows, but the rows of the image can be of different length. A row where there are multiple depth discontinuities is typically longer than a row with fewer discontinuities, as the more numerous depth discontinuities requires inserting more samples. For complex datasets, row length changes frequently and most rows are misaligned with respect to their neighbors above and below, which makes the epipolar occlusion image harder to interpret directly by the user. Therefore, epipolar occlusion images are used as intermediate representations from which conventional output image frames are reconstructed.

The inserted samples become visible as the viewpoint translates on the segment and alleviate disocclusion errors. The epipolar occlusion camera has the strength of capturing all samples visible in the first order depth discontinuities. This means that the camera is built with rays that sample into the first order depth discontinuities visible from the reference viewpoint. However, the construction is not recursive, that is there are no rays that sample into the secondary depth discontinuities revealed by the rays added for the first order depth discontinuities. This means that the epipolar

occlusion camera image provides an aggressive and not exact visibility solution. The guarantee that all first-order depth discontinuities are handled translates to a quality aggressive solution.

5.1.3 The graph camera

The GLC, the k-ray camera, and the occlusion camera achieve disocclusion locally, that is they extend the reference viewpoint to a view region. Some applications require imaging samples that are far away and are substantially occluded. Such samples are not visible from nearby viewpoints. What is needed is a camera model that integrates seamlessly a set of disparate viewpoints. Such a camera is the graph camera [7]. Bundles of the reference planar pinhole camera rays are bent, split, and merged to reach the samples of interest, no matter how far or how heavily occluded from the reference viewpoints. A graph camera can be built to sample all the corridors of the maze and to image the entire maze into a single coherent and non-redundant multiperspective image. As discussed in the introduction, the graph camera has the shortcoming of transitioning abruptly from one viewpoint to the other. The curved ray camera is designed to transition gradually from one viewpoint to another to avoid the distortion artifacts caused by the graph cameras abrupt transitions.

5.2 The curved ray camera model

The curved ray camera is constructed starting from a reference planar pinhole camera PPC_0 that undergoes a series of successive bending operations. Like for the graph camera, the bending operation is defined by a new viewpoint C_1 and by a plane t that indicates where the transition to the new viewpoint should occur. Unlike for the graph camera however, the transition does not occur abruptly, at t but rather at a transition region that contains t. The rays of the CRC are line segments before the transition region and after the transition region. At the transition region, the rays are conic arcs. The conic arcs are constructed to be tangent to the two segment rays they connect. The conic arc construction is also constrained by the desire to obtain a closed-form projection for the resulting curved rays.

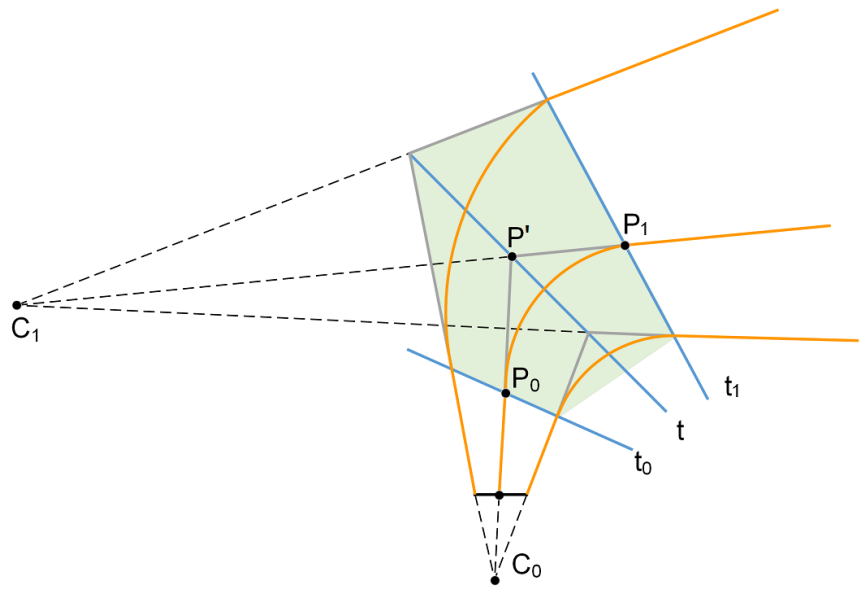

Figure 5.1. The curved ray camera model.

5.2.1 CRC rays

Consider a simple CRC defined by two viewpoints C_0 and C_1, by a transition plane t, and by two transition region planes t_0 and t_1. Given an image plane point P, the ray of the CRC through P is segment C_0P_0, where P_0 is the intersection between C_0P and t_0. Then the ray is a conic arc defined by points P_0, P', and P_1, where P' is the intersection between C_0P and t, and P_1 is the intersection between C_1P and t_1 (Figure 5.1). Finally the ray is a line segment again beyond P_1, namely the line segment through C_1. For general CRC's with multiple viewpoints interconnected with transition regions, the CRC ray is constructed similarly, as a sequence of line segments interconnected by conic arcs.

One option is to choose the conic arc to be a Bézier arc with control points P_0, P', and P_1. This is sufficient to achieve the tangency at P_0 and P_1 requirement, but it does not provide closed-form projection. Nonetheless, Bézier arcs are acceptable when the CRC image is rendered by ray tracing, as is the case, for example, in volume rendering (Figure 5.2). In order to provide closed-form projection, the arcs are not Bézier arcs but rather general conic arcs, as described in the paper [3].

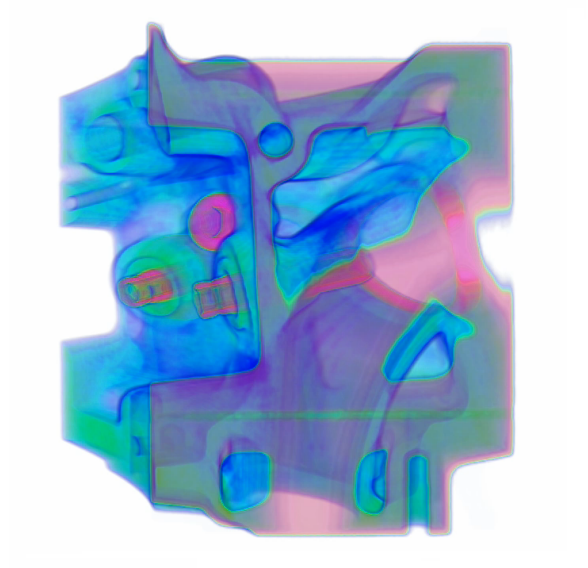

Figure 5.2. Volume rendering with the CRC

5.2.2 Projection

Given a 3D point P, feed-forward rendering requires quickly computing whether P projects on the CRC image, and, if so, where. The first task is to determine whether P is inside the frustum of the CRC. P is in the frustum of the CRC if it is inside one of the subfrusta. The subfrusta of the CRC are PPC subfrusta interconnected by transition region subfrusta. Both these types of subfrusta are convex hexahedra, and one can check whether P is inside a subfrustum with six dot products. When the number of subfrusta is large, one can arrange the planes of the faces in a hierarchical scheme such as a BSP tree.

The second task when projecting a point P, is to compute the CRC image coordinates of P. If P is not inside the CRC frustum, there is no projection. If P is inside the CRC frustum, P is either inside a PPC or a transition subfrustum. If P is inside a PPC subfrustum, projection is straightforward. Like in the case of the graph camera, the output image coordinates of the projection of P are computed by multiplying P with a matrix that concatenates all transformation and projection matrices from the subfrustum containing P to the initial subfrustum. If P is inside a transition subfrustum, the projection involves one line/Bézier arc intersection (i.e. a univariate

quadratic equation), one line/line intersection, and two line/plane intersections, as described in the paper [3].

5.2.3 Constructors

We have developed an interactive, a target tracking, and a path following CRC constructor.

Conventional cameras have only a few degree of freedom. During an interactive visualization session the user typically translates and rotates the camera to obtain an image that reveals the data subset of interest. The camera frustum is rigid with the exception of occasional changes of the field of view. The CRC has many degrees of freedom and one way to use the CRC for interactive visualization of a 3D dataset is to allow the user to add and position viewpoints interactively.

In certain applications, the goal is to keep a target disoccluded as either the target moves, or the visualization camera moves. We achieved this goal with a target tracking CRC constructor. Construction starts from a conventional PPC. If the target is occluded in the PPC image, the constructor will search for a way to bend the rays to reveal the target. One variant of the constructor attempts to bend the rays left/right. A second variant bends rays top/down. The construction for the next frame starts from the camera constructed for the current frame. The camera can only change a controlled amount from one frame to the next to avoid abrupt changes in the camera model which would translate in temporal visualization discontinuities. Let us assume that for a while the target tracking constructor succeeded at dissocluding the target by bending the rays right. At some point it can happen that due to the complex occlusion patterns in the dataset there is no more disocclusion solution to be found by bending the rays right, and a solution was found by bending the rays left. The camera switches over the next few frames from a right bend to a left bend to provide a gradual change of the image. If at some point the target becomes visible again in a conventional camera, the bend is removed gradually and the visualization converges to a conventional planar pinhole camera visualization.

Our third constructor takes a path as input and builds a CRC whose rays conform to the path. The path is segmented uniformly and each pair of consecutive segments is modeled with two PPC subfrusta and a transition region. The CRC image linearizes the path. A typical input path is occlusion free, e.g. the riverbed at the bottom of a

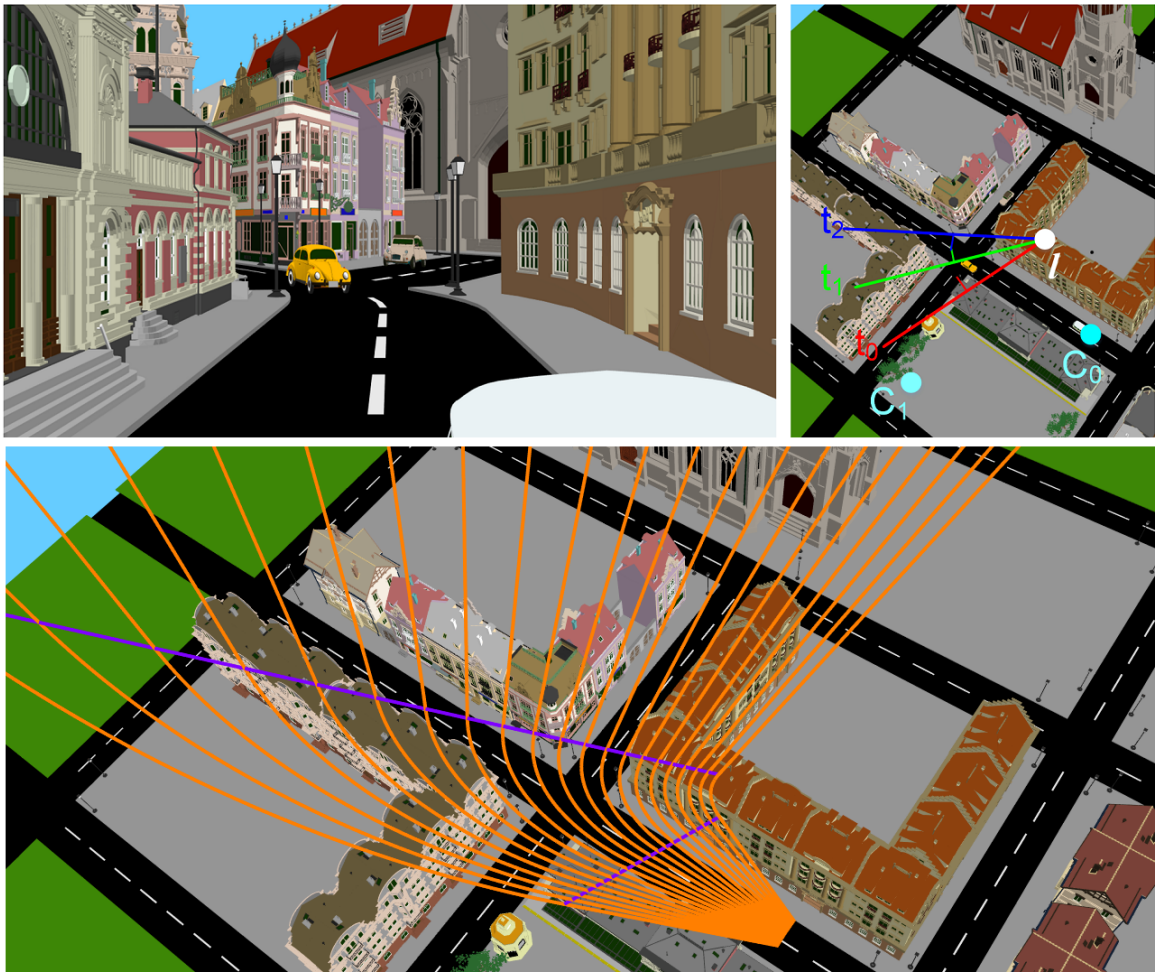

Figure 5.3. A CRC image and its interactive constructor.

canyon, which results in a CRC image that sees ahead, beyond the turns of the path. Figure 5.5 shows the images generated by this constructor (left column) comparing conventional PPC (right column).

5.3 Results and discussion

We demonstrated CRC visualization on the following datasets: Blocks (Figure 5.4, bottom), DNA (Figure 5.4, top), Canyon (Figure 5.5) and City (Figure 5.3), as well as the Engine volume dataset. The CRC achieves disocclusion in a variety of contexts without the distortions of the graph camera.

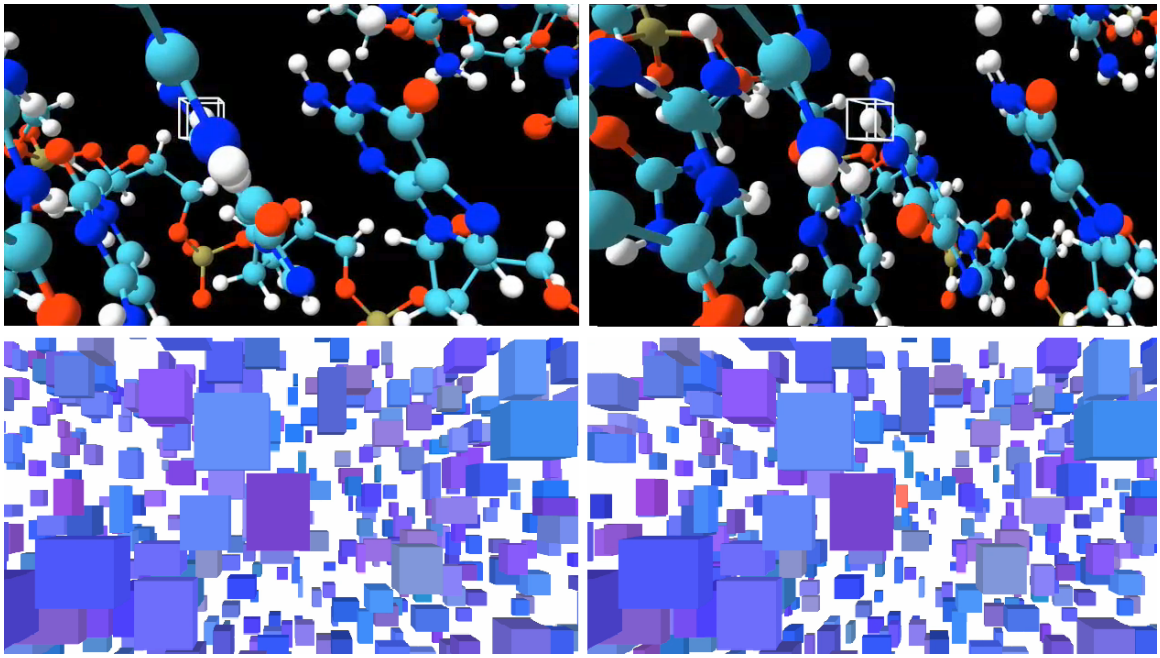

Figure 5.4. Constructing CRC towards target tracking

The CRC has good rendering performance of surface geometry datasets due to its fast projection operation. The Blocks, DNA, Canyon, and City datasets are rendered at 137, 23, 52, and 2.8 frames per second. The performance is only marginally lower compared to graph camera rendering, which yields frame rates of 136, 32, 56, and 2.8. Volume rendering performance on the Engine dataset is 4.1 fps for the CRC compared to 6.7fps for the graph camera.

5.4 Conclusion

In this chapter, we have introduced the curved ray camera model which addresses the single viewpoint limitation of regular image by generalizing the ray geometry to C^1 continuous arcs. This camera model is the first curved ray camera with fast projection operation to the best of our knowledge. This camera model offers high flexibility with many adjustable parameters and thus a good disocclusion power in scenes that there are possible solution to circumvent the occluders. The fast projection modification enables interactive rendering of complex 3D scenes with projection

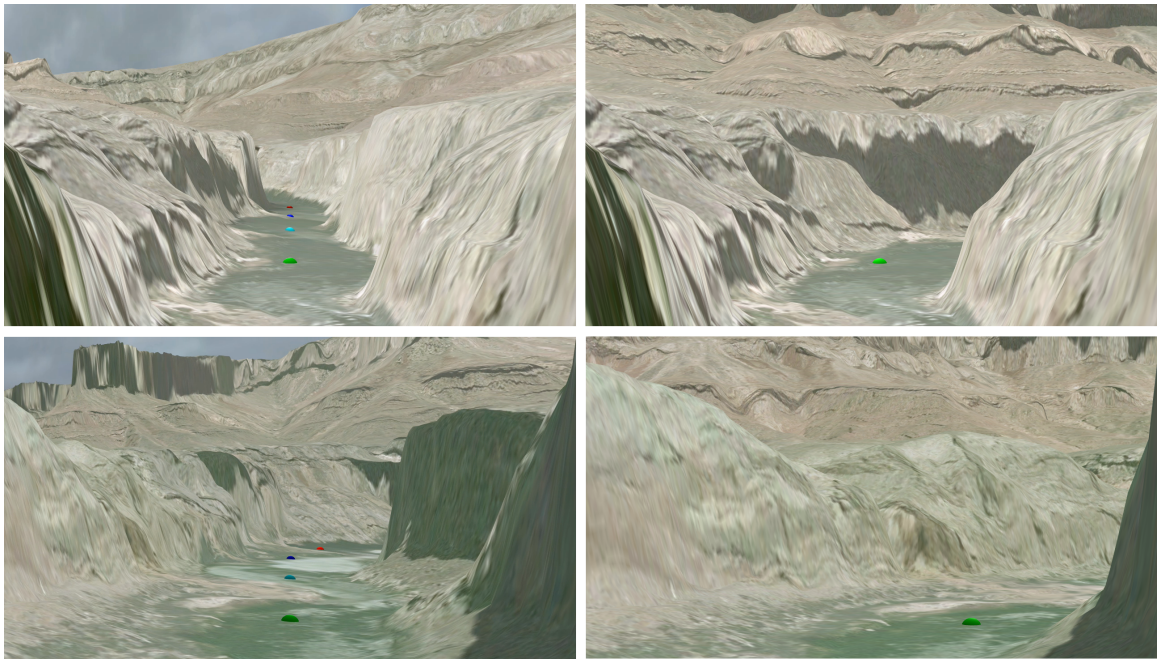

Figure 5.5. Advancing viewpoint along the path with fixed depth.

followed by rasterization technique comparable to the conventional PPC. The closed form definition of the CRC rays also enable straightforward implementation for volume rendering and other ray-tracing techniques.

In short, the curved ray camera model produces images that integrates multiple viewpoints continuously and present the visibility in a human comprehensible format.

6 CONCLUSION

In this dissertation we introduce and validate the image generalization paradigm as a general sample based approach for solving visibility problems in different domains. Images have great potential for solving visibility. However, conventional images fall short of this potential due to the uniform sampling rate limitation, the sample-based limitation, and the single viewpoint/time point limitation. To alleviate these limitations, the image generalization visibility paradigm has the following three elements:

- Sampling pattern generalization
- Visibility sample generalization
- Ray geometry generalization

Ray pattern generalization abandons the uniform sampling rate of conventional images and allows for sampling rate variation based on the local complexity of the dataset. Visibility sample generalization abandons the scalar (i.e. zero-dimensional) visibility sample stored in a conventional z-buffer in favor of more complex visibility samples that can store multiple visible geometric primitives or the trajectory of a sample in a dynamic dataset. Ray geometry generalization abandons the straight line geometry of camera rays and redefines a ray as the set of 3D points that project at a given image location. The generalized rays can avoid occluders to gather the visible samples or geometric primitives needed by the application. We have used the image generalization paradigm to develop four visibility computation approaches.

The *framebuffer generalization* approach uses the sampling pattern generalization and the visibility sampling generalization elements to derive algorithms for from viewpoint, from view segment, from view rectangle, and over time interval visibility.

We have developed an aggressive from viewpoint algorithm that adds sampling locations to enforce that all geometric primitive fragments are sampled. This way, the algorithm guarantees finding all geometric primitives with a completely visible fragment. This includes all completely visible geometric primitives, no matter how small their footprint. To the best of our knowledge, this visibility algorithm is the first aggressive algorithm that provides a quality guarantee.

We have developed an exact from viewpoint algorithm that is efficient and robust. The algorithm combines an aggressive visibility computation based on adding sampling locations to the framebuffer with the use of a single 2D visibility sample for the entire image. The aggressive stage finds visible geometric primitives; the visibility subdivision stored by the 2D sample finds geometric primitives hidden by the visible ones and suggests additional sampling locations where to probe visibility to decide the visibility status of the primitives that are not hidden by the current visible set; the aggressive stage runs again on the suggested sampling locations, finds more visible primitives, which are added to the visibility subdivision, which rules more primitives as hidden; the algorithm converges quickly (e.g. five iterations or less) to the exact visibility solution.

We have developed a quality-guaranteed aggressive from view segment or over time interval algorithm. The visibility sample is generalized to 1D and stores a list of visible geometric primitives as the viewpoint translates on the view segment, or as the geometric primitives move as time changes in a dynamic dataset. The primitives that are visible at a sampling location are computed by solving visibility event equations, and are *not* computed by sampling viewpoint translation or time. The resulting visible set is exact under view translation or under time change. This means that if the view only translates and does not rotate, the visible set contains all geometric primitives visible at the output image pixels. Similarly, as time advances for a dynamic dataset and the view stays fixed, the set contains all visible geometric primitives.

We have developed a quality-guaranteed aggressive from rectangle visibility algorithm. The visibility sample is generalized to a 2D visibility subdivision in the visibility parameter domain of the two viewpoint translations. The 2D visibility sample is computed at each sampling location using the exact from-point visibility algorithm, except that for from-point visibility the two visibility parameters are the image plane coordinates whereas now they are the two viewpoint translations. The resulting visible set is exact under translation.

The *animated depth image* approach uses the visibility sample generalization element of our image generalization visibility paradigm to enhance a visibility sample with the trajectory of the sample as the geometric primitive to which the sample belongs moves over time in a dynamic dataset. Whereas a conventional image acquires a snapshot of a dynamic dataset, an animated depth image stores compactly sufficient data to allow for the reconstruction of output frames at any point in a time interval.

One application of animated depth images is in the context of remote visualization, where, once transferred from the server to the client, the animated depth image is sufficient to reconstruct thousands of frames without the latency of additional data transfers from the server.

The *flexible pinhole camera* approach uses the sampling pattern generalization element of our image generalization visibility paradigm to allow for local variations of the sampling rate of an image, according to an input importance or complexity map. The input complexity map is used to build a distortion map that directs more rays towards the regions of the dataset with higher complexity. The resulting flexible pinhole camera image has a single layer and good coherence, therefore it compresses well with conventional image compression techniques. In a remote visualization scenario, the user can zoom in on the regions of higher complexity or interest without the blurriness associated when zooming into a conventional image. The *curved ray camera* approach uses the ray geometry generalization element of our visibility paradigm to create multiperspective images that enhance a conventional image with samples of distant and heavily occluded regions of the dataset. Unlike the graph camera prior work multiperspective rendering approach, a curved ray camera transitions from one viewpoint to the next gradually, avoiding multiperspective image distortions. A curved ray camera ray is a sequence of line segments interconnected by conic arcs. The conic arcs are designed to provide a closed-form projection operation. The curved ray camera renders complex geometry datasets at interactive rates on the GPU in feed-forward fashion.

The visibility algorithms developed and their evaluation prove the statement that:

The accuracy and efficiency of sample-based 3D visibility computation using images is improved by generalizing the sampling pattern, the visibility sample, and the geometry of the rays of the camera model used to render the image.

REFERENCES

REFERENCES

[1] Jian Cui, Zhiqiang Ma, and Voicu Popescu. Animated depth images for interactive remote visualization of time-varying data sets. *Visualization and Computer Graphics, IEEE Transactions on*, 20(11):1474–1489, 2014.

[2] Voicu Popescu, Bedrich Benes, Paul Rosen, Jian Cui, and Lili Wang. A flexible pinhole camera model for coherent nonuniform sampling. *Computer Graphics and Applications, IEEE*, 34(4):30–41, 2014.

[3] Jian Cui, Paul Rosen, Voicu Popescu, and Christoph Hoffmann. A curved ray camera for handling occlusions through continuous multiperspective visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1235–1242, 2010.

[4] Chunhui Mei, Voicu Popescu, and Elisha Sacks. The occlusion camera. *Computer Graphics Forum*, 24(3):335–342, 2005.

[5] Qi Mo, Voicu Popescu, and Chris Wyman. The soft shadow occlusion camera. In *Computer Graphics and Applications, 2007. PG'07. Fifteenth Pacific Conference on*, pages 189–198. IEEE, 2007.

[6] Paul Rosen and Voicu Popescu. The epipolar occlusion camera. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, pages 115–122. ACM, 2008.

[7] Voicu Popescu, Paul Rosen, and Nicoletta Adamo-Villani. The graph camera. *ACM Transactions on Graphics*, 28(5):158, 2009.

[8] Daniel Cohen-Or, Yiorgos Chrysanthou, Claudio Silva, and Frédo Durand. A survey of visibility for walkthrough applications. *Visualization and Computer Graphics, IEEE Transactions on*, 9(3):412–431, 2003.

[9] Frédo Durand. *3D Visibility: Analytical Study and Applications*. PhD dissertation, Université Joseph Fourier, 2010.

[10] Shaun Nirenstein and Edwin Blake. Hardware accelerated visibility preprocessing using adaptive sampling. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques*, pages 207–216. Springer, 2004.

[11] Peter Wonka, Michael Wimmer, Kaichi Zhou, Stefan Maierhofer, Gerd Hesina, and Alexander Reshetov. Guided visibility sampling. *ACM Transactions on Graphics*, 25(3):494–502, 2006.

[12] Robert Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Proceedings of the Eleventh Annual Conference on Computer Graphics and Interactive Techniques*, pages 137–145. ACM, 1984.

[13] Michael Wand, Matthias Fischer, Ingmar Peter, Friedhelm Meyer auf der Heide, and Wolfgang Straßer. The randomized Z-buffer algorithm: Interactive rendering of highly complex scenes. In *Proceedings of the Twenty-eighth Annual Conference on Computer Graphics and Interactive Techniques*, pages 361–370. ACM, 2001.

[14] Jiří Bittner, Oliver Mattausch, Peter Wonka, Vlastimil Havran, and Michael Wimmer. Adaptive global visibility sampling. *ACM Transactions on Graphics*, 28(3):94, 2009.

[15] László Szirmay-Kalos and Werner Purgathofer. Global ray-bundle tracing with hardware acceleration. In *Proceedings of the Eurographics Workshop on Rendering Techniques '98*, pages 247–258. Springer, 1998.

[16] Andrew Glassner. Spacetime ray tracing for animation. *Computer Graphics and Applications, IEEE*, 8(2):60–70, 1988.

[17] Herve Maurel, Yves Duthen, and Rene Caubet. A 4D ray tracing. *Computer Graphics Forum*, 12(3):285–294, 1993.

[18] Hank Weghorst, Gary Hooper, and Donald Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, 1984.

[19] Michael Cohen and Donald Greenberg. The hemi-cube: A radiosity solution for complex environments. In *Tutorial: Computer Graphics; Image Synthesis*, pages 254–263. Computer Science Press, Inc., 1988.

[20] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Proceedings of the Twenty-second Annual Conference on Computer Graphics and Interactive Techniques*, pages 39–46. ACM, 1995.

[21] Wolfgang Heidrich, Stefan Brabec, and Hans-Peter Seidel. Soft shadow maps for linear lights. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 269–280. Springer, 2000.

[22] Nelson Max and Keiichi Ohsaki. Rendering trees from precomputed Z-buffer views. In *Proceedings of the Eurographics Workshop on Rendering Techniques '95*, pages 74–81. Springer, 1995.

[23] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *Proceedings of the Twenty-fifth Annual Conference on Computer Graphics and Interactive Techniques*, pages 231–242. ACM, 1998.

[24] Jingyi Yu and Leonard McMillan. General linear cameras. In *Computer Vision-ECCV 2004*, pages 14–27. Springer, 2004.

[25] Carl Gribel, Rasmus Barringer, and Tomas Akenine-Möller. High-quality spatio-temporal rendering using semi-analytical visibility. *ACM Transactions on Graphics*, 30(4):54, 2011.

[26] Thouis Jones and Ronald Perry. Antialiasing with line samples. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 197–205. Springer, Springer, 2000.

[27] Loren Carpenter. The A-buffer, an antialiased hidden surface method. In *Proceedings of the Eleventh Annual Conference on Computer Graphics and Interactive Techniques*, pages 103–108. ACM, 1984.

[28] Edwin Catmull. A hidden-surface algorithm with anti-aliasing. In *Proceedings of the Fifth Annual Conference on Computer Graphics and Interactive Techniques*, pages 6–11. ACM, 1978.

[29] Kevin Weiler and Peter Atherton. Hidden surface removal using polygon area sorting. In *Proceedings of the Fourth Annual Conference on Computer Graphics and Interactive Techniques*, pages 214–222. ACM, 1977.

[30] Michael Goodrich. A polygonal approach to hidden-line and hidden-surface elimination. *CVGIP: Graphical Models and Image Processing*, 54(1):1–12, 1992.

[31] Matthew Katz, Mark Overmars, and Micha Sharir. Efficient hidden surface removal for objects with small union size. *Computational Geometry*, 2(4):223–234, 1992.

[32] Micha Sharir and Mark Overmars. A simple output-sensitive algorithm for hidden surface removal. *ACM Transactions on Graphics*, 11(1):1–11, 1992.

[33] Thomas Auzinger, Michael Wimmer, and Stefan Jescke. Analytic visibility on the GPU. *Computer Graphics Forum*, 32(2pt4):409–418, 2013.

[34] Paul Heckbert and Pat Hanrahan. Beam tracing polygonal objects. In *Proceedings of the Eleventh Annual Conference on Computer Graphics and Interactive Techniques*, pages 119–127. ACM, 1984.

[35] John Amanatides. Ray tracing with cones. In *Proceedings of the Eleventh Annual Conference on Computer Graphics and Interactive Techniques*, pages 129–135. ACM, 1984.

[36] Oana Apostu, Frédéric Mora, Djamchid Ghazanfarpour, and Lilian Aveneau. Analytic ambient occlusion using exact from-polygon visibility. *Computers & Graphics*, 36(6):727–739, 2012.

[37] Anish Chandak, Christian Lauterbach, Micah Taylor, Zhimin Ren, and Dinesh Manocha. Ad-frustum: Adaptive frustum tracing for interactive sound propagation. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1707–1722, 2008.

[38] Ryan Overbeck, Ravi Ramamoorthi, and William Mark. A real-time beam tracer with application to exact soft shadows. In *Proceedings of the Eighteenth Eurographics Conference on Rendering Techniques*, pages 85–98. Springer, 2007.

[39] Harry Plantinga, W Brent Seales, and Charles Dyer. Real-time hidden-line elimination for a rotating polyhedral scene using the aspect representation. In *Proceedings of Graphics Interface '90*. Graphics Interface, 1990.

[40] Frédo Durand, George Drettakis, and Claude Puech. The 3D visibility complex. *ACM Transactions on Graphics*, 21(2):176–206, 2002.

[41] Denis Haumont, Otso Mäkinen, and Shaun Nirenstein. A low dimensional framework for exact polygon-to-polygon occlusion queries. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*, pages 211–222. Springer, 2005.

[42] Frédéric Mora and Lilian Aveneau. Fast and exact direct illumination. In *Computer Graphics International 2005*, pages 191–197. IEEE, 2005.

[43] Sylvain Charneau, Lilian Aveneau, and Laurent Fuchs. Exact, robust and efficient full visibility computation in plücker space. *The Visual Computer*, 23(9-11):773–782, 2007.

[44] Jiří Bittner. *Hierarchical Techniques for Visibility Computations*. PhD dissertation, Faculty of Electrical Engineering, Czech Technical University in Prague, 2002.

[45] Shaun Nirenstein, Edwin Blake, and James Gain. Exact from-region visibility culling. In *Proceedings of the Thirteenth Eurographics Workshop on Rendering*, pages 191–202. Springer, 2002.

[46] Leila De Floriani and Paola Magillo. Horizon computation on a hierarchical triangulated terrain model. *The Visual Computer*, 11(3):134–149, 1995.

[47] Peter Wonka and Dieter Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum*, 18(3):51–60, 1999.

[48] Jiří Bittner, Peter Wonka, and Michael Wimmer. Visibility preprocessing for urban scenes using line space subdivision. In *Computer Graphics and Applications, 2001. Proceedings. Ninth Pacific Conference on*, pages 276–284. IEEE, 2001.

[49] Laura Downs, Tomas Möller, and Carlo Séquin. Occlusion horizons for driving through urban scenery. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 121–124. ACM, 2001.

[50] Brandon Lloyd and Parris Egbert. Horizon occlusion culling for real-time rendering of hierarchical terrains. In *Proceedings of the Conference on Visualization '02*, pages 403–410. IEEE, 2002.

[51] Tommer Leyvand, Olga Sorkine, and Daniel Cohen-Or. Ray space factorization for from-region visibility. *ACM Transactions on Graphics*, 22(3):595–604, 2003.

[52] Jiří Bittner, Peter Wonka, and Michael Wimmer. Fast exact from-region visibility in urban scenes. In *Proceedings of the Eurographics Workshop on Rendering Technique 2005*, pages 223–230, 2005.

[53] Seth Teller and Carlo Séquin. Visibility preprocessing for interactive walkthroughs. In *Proceedings of the Eighteenth Annual Conference on Computer Graphics and Interactive Techniques*, pages 61–70. ACM, 1991.

[54] Seth Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. Technical report, University of California at Berkeley, 1992.

[55] Frédo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. In *Proceedings of the Twenty-seventh Annual Conference on Computer Graphics and Interactive Techniques*, pages 239–248. ACM, 2000.

[56] Xavier Décoret, Gilles Debunne, and François Sillion. Erosion based visibility preprocessing. In *Proceedings of the Fourteenth Eurographics Workshop on Rendering*, pages 281–288. Springer, 2003.

[57] James Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[58] Ned Greene, Michael Kass, and Gavin Miller. Hierarchical Z-buffer visibility. In *Proceedings of the Twentieth Annual Conference on Computer Graphics and Interactive Techniques*, pages 231–238. ACM, 1993.

[59] Hansong Zhang, Dinesh Manocha, Tom Hudson, and Kenneth Hoff III. Visibility culling using hierarchical occlusion maps. In *Proceedings of the Twenty-fourth Annual Conference on Computer Graphics and Interactive Techniques*, pages 77–88. ACM, 1997.

[60] Tom Hudson, Dinesh Manocha, Jonathan Cohen, Ming Lin, Kenneth Hoff, and Hansong Zhang. Accelerated occlusion culling using shadow frusta. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 1–10. ACM, 1997.

[61] Jiří Bittner, Vlastimil Havran, and Pavel Slavik. Hierarchical visibility culling with occlusion trees. In *Computer Graphics International, 1998. Proceedings*, pages 207–219. IEEE, 1998.

[62] Heinrich Hey, Robert F Tobler, and Werner Purgathofer. Real-time occlusion culling with a lazy occlusion grid. In *Proceedings of the Twelfth Eurographics Conference on Rendering*, pages 217–221. Springer, 2001.

[63] James Klosowski and Claudio Silva. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *Visualization and Computer Graphics, IEEE Transactions on*, 7(4):365–379, 2001.

[64] Jiří Bittner, Michael Wimmer, Harald Piringer, and Werner Purgathofer. Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum*, 23(3):615–624, 2004.

[65] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 83–90. ACM, 1997.

[66] Timo Aila and Samuli Laine. Alias-free shadow maps. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques*, pages 161–166. Springer, 2004.

[67] Gregory Johnson, Juhyun Lee, Christopher Burns, and William Mark. The irregular Z-buffer: Hardware acceleration for irregular data structures. *ACM Transactions on Graphics*, 24(4):1462–1482, 2005.

[68] Voicu Popescu, John Eyles, Anselmo Lastra, Joshua Steinhurst, Nick England, and Lars Nyland. The WarpEngine: An architecture for the post-polygonal age. In *Proceedings of the Twenty-seventh Annual Conference on Computer Graphics and Interactive Techniques*, pages 433–442. ACM, 2000.

[69] George Furnas. Generalized fisheye views. In *Proceedings of the ACM SIGCHI'86 Conference on Human Factors in Computing Systems*, pages 16–23. ACM, 1986.

[70] Elisha Sacks and Victor Milenkovic. Robust cascading of operations on polyhedra. *Computer-Aided Design*, 46:216–220, 2014.

[71] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. MPFR: A multiple precision binary floating point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2), 2007.

[72] Gerhard Klimeck, Michael McLennan, Sean Brophy, George Adams III, and Mark Lundstrom. nanoHUB.org: Advancing education and research in nanotechnology. *Computing in Science & Engineering*, 10(5):17–23, 2008.

[73] Simon Stegmaier, Marcelo Magallón, and Thomas Ertl. A generic solution for hardware-accelerated remote visualization. In *VISSYM '02 Proceedings of the Dymposium on Data Visualisation*. VISSYM, 2002.

[74] Fabrizio Lamberti and Andrea Sanna. A streaming-based solution for remote visualization of 3D graphics on mobile devices. *Visualization and Computer Graphics, IEEE Transactions on*, 13(2):247–260, 2007.

[75] Silicon Graphics Inc. *OpenGL Vizserver 3.0–Application-Transparent Remote Interactive Visualization and Collaboration*, 2003.

[76] Pieter Simoens, Filip De Turck, Bart Dhoedt, and Piet Demeester. Remote display solutions for mobile cloud computing. *Computer*, 44(8):46–53, 2011.

[77] Gianluca Paravati, Cesare Celozzi, Andrea Sanna, and Fabrizio Lamberti. A feedback-based control technique for interactive live streaming systems to mobile devices. *Consumer Electronics, IEEE Transactions on*, 56(1):190–197, 2010.

[78] Richard Werner Nelem Pazzi, Azzedine Boukerche, and Tingxue Huang. Implementation, measurement, and analysis of an image-based virtual environment streaming protocol for wireless mobile devices. *Instrumentation and Measurement, IEEE Transactions on*, 57(9):1894–1907, 2008.

[79] Andrei Hutanu, Gabrielle Allen, and Tevfik Kosar. High-performance remote data access for remote visualization. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 121–128. IEEE, 2010.

[80] Steven Callahan, Joao Luiz Dihl Comba, Peter Shirley, and Cláudio Silva. Interactive rendering of large unstructured grids using dynamic level-of-detail. In *Visualization, 2005. VIS 05. IEEE*, pages 199–206. IEEE, 2005.

[81] David Luebke. *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2003.

[82] Yarden Livnat, Steven Parker, and Christopher Johnson. Fast isosurface extraction methods for large image data sets. In *Handbook of Medical Imaging*, pages 731–745. Academic Press, Inc., 2000.

[83] Attila Gyulassy and Vijay Natarajan. Topology-based simplification for feature extraction from 3D scalar fields. In *Visualization, 2005. VIS 05. IEEE*, pages 535–542. IEEE, 2005.

[84] Steven Callahan, Louis Bavoil, Valerio Pascucci, and Claudio Silva. Progressive volume rendering of large unstructured grids. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1307–1314, 2006.

[85] Hugues Hoppe. Progressive meshes. In *Proceedings of the Twenty-third Annual Conference on Computer Graphics and Interactive Techniques*, pages 99–108. ACM, 1996.

[86] Sinésio Pesco, Peter Lindstrom, Valerio Pascucci, and Cláudio Silva. Implicit occluders. In *Volume Visualization and Graphics, 2004 IEEE Symposium on*, pages 47–54. IEEE, 2004.

[87] Jinzhu Gao and Han-Wei Shen. Parallel view-dependent isosurface extraction using multi-pass occlusion culling. In *Parallel and Large-Data Visualization and Graphics, 2001. Proceedings. IEEE 2001 Symposium on*, pages 67–152. IEEE, 2001.

[88] Lars Lippert, Markus Gross, and Christian Kurmann. Compression domain volume rendering for distributed environments. *Computer Graphics Forum*, 16(s3):C95–C107, 1997.

[89] Martin Isenburg, Peter Lindstrom, and Jack Snoeyink. Streaming compression of triangle meshes. In *ACM SIGGRAPH 2005 Sketches*, page 136. ACM, 2005.

[90] Paul Rosen and Voicu Popescu. An evaluation of 3-D scene exploration using a multiperspective image framework. *The Visual Computer*, 27(6-8):623–632, 2011.

[91] Kwan-Liu Ma and David Camp. High performance visualization of time-varying volume data over a wide-area network. In *Supercomputing, ACM/IEEE 2000 Conference*, pages 29–29. IEEE, 2000.

[92] Anna Tikhonova, Hongfeng Yu, Carlos Correa, Jacqueline Chen, and Kwan-Liu Ma. A preview and exploratory technique for large-scale scientific simulations. In *Proceedings of the Eleventh Eurographics Conference on Parallel Graphics and Visualization*, pages 111–120. Springer, 2011.

[93] Simon Stegmaier, Joachim Diepstraten, Manfred Weiler, and Thomas Ertl. Widening the remote visualization bottleneck. In *Image and Signal Processing and Analysis, 2003. ISPA 2003. Proceedings of the Third International Symposium on*, volume 1, pages 174–179. IEEE, 2003.

[94] Ned Greene. Environment mapping and other applications of world projections. *Computer Graphics and Applications, IEEE*, 6(11):21–29, 1986.

[95] Shigang Li. Full-view spherical image camera. In *Pattern Recognition, 2006. ICPR 2006. Eighteenth International Conference on*, volume 4, pages 386–390. IEEE, 2006.

[96] Stephen DiVerdi, Jason Wither, and Tobias Höllerer. All around the map: Online spherical panorama construction. *Computers & Graphics*, 33(1):73–84, 2009.

[97] Richard Szeliski and Heung-Yeung Shum. Creating full view panoramic image mosaics and environment maps. In *Proceedings of the Twenty-fourth Annual Conference on Computer Graphics and Interactive Techniques*, pages 251–258. ACM, 1997.

[98] Shree Nayar and Amruta Karmarkar. 360× 360 mosaics. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 388–395. IEEE, 2000.

[99] Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stéphane Lafon, Richard Lyon, Abhijit Ogale, Luc Vincent, and Josh Weaver. Google street view: Capturing the world at street level. *Computer*, 43(6):32–38, 2010.

[100] Augusto Roman, Gaurav Garg, and Marc Levoy. Interactive design of multi-perspective images for visualizing urban landscapes. In *Proceedings of the Conference on Visualization '04*, pages 537–544. IEEE, 2004.

[101] Aseem Agarwala, Maneesh Agrawala, Michael Cohen, David Salesin, and Richard Szeliski. Photographing long scenes with multi-viewpoint panoramas. *ACM Transactions on Graphics*, 25(3):853–861, 2006.

[102] Eric Luke and Charles Hansen. Semotus visum: A flexible remote visualization framework. In *Proceedings of the Conference on Visualization '02*, pages 61–68. IEEE, 2002.

[103] Wes Bethel, Brian Tierney, Jason Lee, Dan Gunter, and Stephen Lau. Using high-speed WANs and network data caches to enable remote and distributed visualization. In *Supercomputing, ACM/IEEE 2000 Conference*, page 28. IEEE, 2000.

[104] Daniel Wood, Adam Finkelstein, John Hughes, Craig Thayer, and David Salesin. Multiperspective panoramas for cel animation. In *Proceedings of the Twenty-fourth Annual Conference on Computer Graphics and Interactive Techniques*, pages 243–250. ACM, 1997.

[105] Paul Rademacher and Gary Bishop. Multiple-center-of-projection images. In *Proceedings of the Twenty-fifth Annual Conference on Computer Graphics and Interactive Techniques*, pages 199–206. ACM, 1998.

[106] Voicu Popescu, Paul Rosen, Laura Arns, Xavier Tricoche, Chris Wyman, and Christoph Hoffmann. The general pinhole camera: Effective and efficient nonuniform sampling for visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 16(5):777–790, 2010.

[107] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the Twenty-seventh Annual Conference on Computer Graphics and Interactive Techniques*, pages 335–342. ACM, 2000.

[108] Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of the Twenty-seventh Annual Conference on Computer Graphics and Interactive Techniques*, pages 343–352. ACM, 2000.

[109] Renato Pajarola, Miguel Sainz, and Yu Meng. DMesh: Fast depth-image meshing and warping. *International Journal of Image and Graphics*, 4(4):653–681, 2004.

[110] Anna Tikhonova, Carlos Correa, and Kwan-Liu Ma. Visualization by proxy: A novel framework for deferred interaction with volume data. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1551–1559, 2010.

[111] Anna Tikhonova, Carlos Correa, and Kwan-Liu Ma. An exploratory technique for coherent visualization of time-varying volume data. *Computer Graphics Forum*, 29(3):783–792, 2010.

[112] William Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3D warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 7–16. ACM, 1997.

[113] Chun-Fa Chang, Gary Bishop, and Anselmo Lastra. LDI tree: A hierarchical representation for image-based rendering. In *Proceedings of the Twenty-sixth Annual Conference on Computer Graphics and Interactive Techniques*, pages 291–298. ACM, 1999.

[114] Niklas Elmqvist and Philippas Tsigas. A taxonomy of 3D occlusion management for visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 14(5):1095–1109, 2008.

[115] Leonard McMillan Jr. *An Image-based Approach to Three-dimensional Computer Graphics*. PhD dissertation, University of North Carolina at Chapel Hill, 1997.

[116] Paul Rosen and Voicu Popescu. Simplification of node position data for interactive visualization of dynamic data sets. *Visualization and Computer Graphics, IEEE Transactions on*, 18(9):1537–1548, 2012.

[117] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972.

[118] David Douglas and Thomas Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[119] Voicu Popescu and Paul Rosen. Forward rasterization. *ACM Transactions on Graphics*, 25(2):375–411, 2006.

[120] Asma Al-Saidi, David Walker, and Omer Rana. On-demand transmission model for remote visualization using image-based rendering. *Concurrency and Computation: Practice and Experience*, 24(18):2328–2345, 2012.

[121] Hariharan Lalgudi, Michael Marcellin, Ali Bilgin, Han Oh, and Mariappan Nadar. View compensated compression of volume rendered images for remote visualization. *Image Processing, IEEE Transactions on*, 18(7):1501–1511, 2009.

[122] Jean-Dominique Gascuel, Nicolas Holzschuch, Gabriel Fournier, and Bernard Péroche. Fast non-linear projections using graphics hardware. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, pages 107–114. ACM, 2008.

[123] Pradeep Sen, Mike Cammarano, and Pat Hanrahan. Shadow silhouette maps. *ACM Transactions on Graphics*, 22(3):521–526, 2003.

[124] Ganesh Ramanarayanan, Kavita Bala, and Bruce Walter. *Feature-based Textures*. Technical report, Cornell University, 2004.

[125] Laurent Balmelli, Gabriel Taubin, and Fausto Bernardini. Space-optimized texture maps. *Computer Graphics Forum*, 21(3):411–420, 2002.

[126] Fabio Policarpo and Manuel Oliveira. Relief mapping of non-height-field surface details. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, pages 55–62. ACM, 2006.

[127] Turner Whitted. An improved illumination model for shaded display. In *Proceedings of the Sixth Annual Conference on Computer Graphics and Interactive Techniques*, page 14. ACM, 1979.

[128] Eyal Ofek and Ari Rappoport. Interactive reflections on curved objects. In *Proceedings of the Twenty-fifth Annual Conference on Computer Graphics and Interactive Techniques*, pages 333–342. ACM, 1998.

[129] László Szirmay-Kalos, Barnabás Aszódi, István Lazányi, and Mátyás Premecz. Approximate ray-tracing on the GPU with distance impostors. *Computer Graphics Forum*, 24(3):695–704, 2005.

[130] James Blinn and Martin Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, 1976.

[131] Ivan Viola, Miquel Feixas, Mateu Sbert, and Meister Groller. Importance-driven focus of attention. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):933–940, 2006.

[132] Stefan Bruckner and Eduard Groller. Exploded views for volume data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1077–1084, 2006.

[133] Patrick Baudisch, Nathaniel Good, and Paul Stewart. Focus plus context screens: Combining display technology with visualization techniques. In *Proceedings of the Fourteenth Annual ACM Symposium on User Interface Software and Technology*, pages 31–40. ACM, 2001.

[134] Jonh Lamping and Ramana Rao. The hyperbolic browser: A focus + context technique for visualizing large hierarchies. *Journal of Visual Languages & Computing*, 7(1):33–55, 1996.

[135] Nelson Wong, Sheelagh Carpendale, and Saul Greenberg. Edgelens: An interactive method for managing edge congestion in graphs. In *Information Visualization, 2003. INFOVIS 2003. IEEE Symposium on*, pages 51–58. IEEE, 2003.

[136] Emmanuel Pietriga and Caroline Appert. Sigma lenses: Focus-context transitions combining space, time and translucence. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1343–1352. ACM, 2008.

[137] Marianne Sheelagh Therese Carpendale, David Cowperthwaite, and David Fracchia. Distortion viewing techniques for 3-dimensional data. In *Information Visualization '96, Proceedings IEEE Symposium on*, pages 46–53. IEEE, 1996.

[138] Lujin Wang, Ye Zhao, Klaus Mueller, and Arie Kaufman. The magic volume lens: An interactive focus + context technique for volume rendering. In *Visualization, 2005. VIS 05. IEEE*, pages 367–374. IEEE, 2005.

[139] Voicu Popescu, Jordan Dauble, Chunhui Mei, and Elisha Sacks. An efficient error-bounded general camera model. In *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, pages 121–128. IEEE, 2006.

VITA

VITA

Jian Cui was born in Harbin, China. In 2009, he received a Bachelor of Science degree from the Computer Science and Technology Department of the Harbin Institute of Technology, China. In the same year, he started his PhD studies in the Computer Science Department of Purdue University. He received a Master of Science degree from Purdue in 2012. During his graduate studies at Purdue University, he acquired a comprehensive understanding of computer architecture, programming languages, operating systems, and especially computer graphics.

His research focuses on the fundamentals of computer graphics, including camera model design, image generalization and visibility computation, addressing the limitation of existing computer graphics pipelines in various specific scenarios. His research results in these areas have been presented in this dissertation.

His research interests cross over to computer vision, scientific visualization, and statistical machine learning. He also has been a member of a cyberlearning project that aims to apply computer graphics to educational psychology research, especially in the context of instructor avatar gesture studies.

In his leisure time, he enjoys reading, traveling, photography, as well as implementing fun toy programs, mostly relevant to image processing.