A POINT-BASED REMOTE VISUALIZATION PIPELINE FOR LARGE-SCALE VIRTUAL REALITY

BY

JINGHUA GE
B.S., Beijing Information Technology Institute, China, 1997
M.S., Tsinghua University, Beijing, China, 2000

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2007

Chicago, Illinois

ACKNOWLEDGMENTS

I would like to thank my thesis committee -- Professor Andrew Johnson, Professor Dan Sandin, Professor Jason Leigh, Professor Tom Moher, and Professor Dan Schonfeld -- for their unwavering support and assistance. They provided guidance in all areas that helped me accomplish my research goals and enjoy myself in the process. I would also like to acknowledge my coworkers in my research group -- Tom Peterka, Robert Kooima, and Javier Girado – for their in-depth discussions which are very important to the conduct of the thesis research.

A number of individuals in EVL were extremely helpful to me during my research, and I would like to thank them as well – Lance Long, Patrick Hallihan, Alan Verlo, Laura Wolf, Eric He, Charles Zhang, Venkat Vishwanath – for their support, encourage and advice.

JG

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER PAGE

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

LIST OF FIGURES (Continued)

FIGURE PAGE

LIST OF ABBREVIATIONS

VR	Beck Depression Inventory
PBR	Brief Pain Questionnaire
LOD	Level of Detail
SISD	Single Instruction, Single Data
SIMD	Single Instruction, Multiple Data
MISD	Multiple Instructions, Single Data
MIMD	Multiple Instructions, Multiple Data

SUMMARY

State-of-the-art Virtual Reality technologies such as Varrier™ bring better interaction and comprehension into visualization experience. But VR applications are still limited in the area of large-scale scientific visualization mostly because of the intensive graphics computation for VR viewing.

The goal of this thesis is to design and implement a distributed visualization framework which combines VR technologies and remote computing resources through a high speed network, so that large-scale scientific datasets can be visualized in real-time on local VR devices.

The framework is designed to be a scalable distributed system with pipelined data retrieval, computation, and visualization for various datasets. Scalability makes the system adaptive to the available computing and visualizing resource configurations. Each subsystem of the distributed system can perform either cluster-based parallel computing or single workstation-based sequential computing. The pipeline configuration can be optimized based on a balanced granularity as the ratio of computation to communication. The pipeline is an MIMD design which explores computing and networking parallelism along with the data flow.

Special implementation features of the pipeline are presented in this thesis based on the requirements of interactive VR exploration. First of all, point samples are introduced as an intermediate format of data which flow through the pipeline. The conceptual simplicity and rendering performance of points make them a good choice as modeling and display primitives for efficient VR-end geometry caching and view reconstruction. Sampling, packing, and rendering algorithms are discussed in this thesis to transform the original dataset into point samples, cache the point geometry, and reconstruct seamless 2D viewing from the point geometry. Different implementations of these algorithms with different levels of computational complexity are studied and customized to match the various visualization requirements for specific VR applications. The straightforward functional decomposition of point-based graphics enables flexible and balanced workload distribution through the computation pipeline. Secondly, different subsystem coupling schemes are discussed and can be selected to fit for different VR application requirements. Looser coupling of the VR client from the computation server means less waiting time inside the view construction cycle, but with possible viewing artifacts due to delayed view updating.

SUMMARY (Continued)

As case studies to prove the feasibility of the proposed visualization strategy, datasets with different characteristics, such as triangle meshes and volumes, are used as customized visualization instances of the proposed framework. Several pipeline configurations, such as single server to single client, server cluster to single client, and server cluster to client cluster, are tested for different applications. Also, different point based algorithms and subsystem coupling schemes are selected in each case study and their functionalities can be merged together seamlessly for a specific application. All experiments show that VR interaction can be improved for various visualization tasks by utilizing the visualization framework presented in this thesis.

CHAPTER 1 INTRODUCTION

1.1 Motivation

Modern data acquisition techniques have produced huge datasets in high-precision for real world objects and environments. These datasets may also be distributed among multiple data servers. The growth of the size and distribution of scientific data sets has been not only pushing the limits of computing resources and networking bandwidth, but also taxing the ability of scientists to understand them. Effective visualization systems must therefore be both *efficient* on large data retrieval and processing and *comprehensible* for the user. Visualization research has focused on developing techniques that address both of these criteria.

Classically, a virtual reality (VR) application features a complex simulation using input and output devices to provide users with a sense of immersion in a synthetic world. One main expectation of VR is to maintain good quality stereo visualization and interaction with low latency and high refresh rates. VR brings immersive comprehension into scientific visualization for its users. Today's VR device can not only be a single workstation, but also a scalable cluster-driven tiled display. Real-time large-scale dataset exploration in an immersive tiled-display VR environment is very promising for the future of scientific visualization.

There exist several large challenges in developing such a real-time VR visualization system for large-scale datasets. First of all, the size and distributed storage of large-scale datasets make it undesirable to download the original data completely into a local machine. Secondly, graphics processing power is limited for commodity computing hardware without super computation power. Local visualization of large scale datasets with high frame rates is a challenge, especially for VR, because VR is generally graphics-intensive with stereo drawing. Furthermore, for some passive auto-stereo VR systems [Sandin05], the image interleaving overhead has proved to be a heavy load even with the newest GPU based solutions [Kooima07].

Fortunately, new technologies such as parallel computing over a computer cluster have been brought in to solve large-scale problems. Moreover, the advent of high-speed networks, such as CAVEwave [Cavewave], is providing the potential for new approaches to real time organization, distribution, analysis, and visualization of large-scale scientific datasets. For example, it's now possible to retrieve data in real-time from data sources distributed all around the world through a dedicated high speed high bandwidth network. Also, remote visualization techniques have been presented as a prototype distributed real-time visualization pipeline, where a local visualization environment is connected to a scalable parallel computer via a high-speed network. The data are either computed in real-time or pre-computed on the parallel computer, and then are transferred to the local visualization environment where fast view reconstruction is accomplished.

In this thesis, a point-based real-time remote visualization pipeline for virtual reality is proposed to enable high speed exploration of large-scale datasets for VR devices from single workstations to cluster-driven tiled displays. A scalable parallel computer cluster, called the computation server, stands as a bridge to connect both the possibly distributed data servers and the local VR devices through high-speed network. Based on the VR client's viewing demands, the computation server retrieves data from the data servers and samples the visible part of the original data into 3D point samples. 3D point samples are chosen to be the intermediate data form flowing from the computation server to the VR visualization client because of their consistency with the output of conventional graphics rasterization, and their conceptual simplicity and efficient rendering performance as a display primitive. Scalable remote visualization in client-server architecture brings together distributed data storage, high-performance computing and state-of-art VR techniques for better data exploration and analysis in various research areas.

1.2 Terminology

The important and commonly used terms in this thesis are listed below. These terms are mainly associated with Virtual Reality, scientific visualization, and parallel computing. Most of these will be discussed in more detail throughout the thesis.

Virtual Reality, Large-scale Virtual Reality

A Virtual Reality (VR) application features a complex simulation using input and output devices to provide users with a sense of immersion in a synthetic world with stereo viewing. Smooth VR interaction generally needs fast view construction with frame rates of at least 15fps. A large-scale Virtual Reality problem indicates that the application navigates through a large-scale dataset.

Autostereoscopy

Autostereoscopy is a method of displaying three-dimensional images that can be viewed without the use of special headgear or glasses on the part of the user. These methods produce depth perception in the viewer even though the image is produced by a flat device.

Image Interleaving

Image interleaving is the stereo image compositing process at the end of passive autostereoscopic visualization. For example, in a static parallax barrier display like the Varrier^{TM}, the left eye and right eye images are rendered in strips by a virtual linescreen occlusion, and finally interleaved together into the same frame buffer and directed into the correct eyes by the physical barrier attached to the LCD display.

Surface dataset and volumetric dataset

Surface datasets represent exterior characteristics of an object. Volumetric datasets represent a 3D sampling of the interior structure of the objects, including amorphous and semi-transparent features, over a uniform/non-uniform 3D grid.

Point Sample, sample rate, and sample resolution

A point sample represents a type of modeling and display primitive of graphics datasets. For surface datasets, a point sample can be a surfel [Pfister00], which consists of spatial coordinate ***p***, normal orientation ***n***, color ***c***, and information about its spatial extent in object-space. For volumetric datasets, a point sample can represent an ellipsoid or sphere which has 3D spatial expansion. The term ***sample rate*** indicates the object-space point sample distribution; while the term ***sample resolution*** is used to indicate the sample density according to decimation.

Pipeline, Parallel pipeline, and Distributed pipeline

A pipeline consists of a sequence of stages through which computation and data flow. New data is input at the start of the pipeline while other data is being processed throughout the pipeline. Important issues are the interconnections and data paths between the stages of different pipelines. In a parallel pipeline each stage of the pipeline itself performs parallel computing too. A distributed pipeline

refers that different stages of the pipeline are executed on geographically distributed computing systems.

Remote Visualization

Remote visualization is a distributed pipeline whose purpose is to retrieve, analyze and visualize 3D scientific datasets. It usually involves distributed data storage, remote computation resources, and local visualization facilities. The different computing stages in the pipeline are executed by distributed resources connected by a high speed network.

Data Partition

Data partitioning is the basis of a parallel computing algorithm by unit repetition. The data associated with a problem is decomposed and each parallel task then works on a portion of the data. There are different ways to partition data over the processing units for a parallel visualization task, such as object-space data partition and image-space data partition.

Workload Distribution

Workload partitioning is the basis of pipeline computing. The computation functionalities are decomposed and distributed over the pipeline. Workload distribution lends itself well to problems that can be split into different tasks.

Load Balancing

Load balancing refers to the practice of distributing work among tasks so that ***all*** tasks are kept busy ***all*** of the time. It can be considered a minimization of task idle time. Load balancing is important to parallel programs for performance reasons.

Communication

Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network, however the actual event of data exchange is commonly referred to as communication regardless of the method employed.

Synchronization

The coordination of parallel tasks in real time, very often associated with communication. Often implemented by establishing a synchronization point within an application where a task may not proceed further until other task(s) reaches the same or logically equivalent point. Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's overall execution time to increase.

Granularity

In parallel computing, granularity is a qualitative measure of the ratio of computation to communication. A ***coarse*** granularity indicates that relatively large amounts of computational work are done between communication events. Similarly, a ***fine*** granularity indicates that relatively small amounts of computational work are done between communication events.

Scalability

Scalability refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors.

1.3 Problem Statement

This thesis proposes to implement a real-time remote visualization pipeline to enable high speed exploration of large-scale datasets for VR devices from single workstations to cluster driven tiled displays. A remote visualization system is a distributed pipeline where the rendering process involves both remote and local resources. In this thesis, the remote resoures are called ***server*** and the local resources are called ***client***. For a typical remote visualization system, the implementation features include its work load distribution scheme, the server-client coupling mode, and the data caching scheme.

The **work load distribution scheme** between the server and the client usually falls into one of the two categories:

- *Remote rendering – local display*. In this mode, rendering is performed by a remote server based on the client's demand, and the resulting stream of pixels is sent over the network to the client for display. This image-based solution makes client visualization independent of scene complexity, but requires powerful server computation and high network bandwidth, especially when the client asks for large display resolution.

- *Remote geometry delivering – local rendering*. In this mode, a remote server will send partial geometry data to the client on demand, and then the client can render the scene locally. This solution relies on server's data retrieval and client's rendering ability. Low-end clients are usually not capable of rendering very complex scenes in high speed.

Because VR is graphics-intensive, it's important to have a balanced workload distribution scheme between the server and client for better overall performance. For example, a passive auto-stereo VR rendering frame includes stereo rendering of the scene for both eyes and a final image interleaving. In the remote rendering – local display mode, the server needs to do stereo rendering for each frame and the client waits for the stereo

images before executing the image interleaving. The server's workload can easily become too heavy in this mode. On the other hand, in the remote geometry delivering – local rendering mode, the server does the geometry delivering for each frame, and the client needs to do both stereo rendering and image interleaving. The client's workload can easily become too heavy in this mode. A more flexible workload distribution scheme is needed for VR in order to achieve better balance between the server and client's processing.

The work flow cycle for each frame in the remote computation pipeline includes view requesting from the client to server, server computation, data transfer from the server to the client, and client view construction. The relation among multiple work flow cycles defines the degree of **server-client coupling**, described as following:

- *Synchronous coupling*. In this mode, the work flow is sequential. The server and the client both wait for the completion of the current data flow cycle before continuing to the next frame. Waiting happens at both server and client and aggravates the inter-frame delay.

- *Loose coupling*. In this mode, adjacent work flow cycles can be interleaved with each other by introducing a circular buffering algorithm at the client side. Data coming from different server computation frames can be stored in different

buffers, thus avoiding unnecessary inter-waiting between the server and client. The looseness of the coupling depends on the number of buffers decides. Loose coupling need more data storage space at the client side, but also improves the view updating frame rate.

For an interactive VR, the view reconstruction frame rate is very critical in order to reflect the free movement of a head tracked user. When the loose coupling of server/client can't reach the frame rate requirement for a specific VR project, it's desirable to introduce a different kind of server-client coupling scheme to achieve a higher view construction frame rate, even at the expense of possibly decreasing the viewing quality.

In a remote visualization task, data is transferred every frame from the server to the client. Usually no **data caching** scheme is applied at the client side, especially when the data delivered to the client is view-dependent, such as the 2D pixel stream, as it's not reusable after the display of the current frame. Still, data caching can be a beneficial option for more global analysis or future reuse. In a frame-rate critical visualization environment like VR, it may be desirable to have a sophisticated data caching scheme at both the server and the client side to enable more efficient and flexible view construction.

As a summary, besides making use of the existing parallel graphics rendering and remote visualization techniques, new algorithms need to be developed to solve the aforementioned problems to implement a scalable remote visualization pipeline for the graphics-expansive yet frame-rate critical VR environment.

1.3.1 Approaches

The traditional graphics rendering algorithm takes the original dataset as input and rasterize it into 2D pixel stream(s) for a specific viewing. The key idea of a load-balanced remote visualization is to split the traditional graphics rendering algorithm into a sequence of processing stages with decomposed functionalities and distribute the factorized functionalities over the pipeline. The rendering functionality decomposition enables flexible workload distribution over the pipeline which can be adaptive to the system configurations and computing performance.

As to the data flow along the pipeline, usually an intermediate data format other than the original dataset setup or the 2D pixel stream is introduced as the output of the server processing and the input of the client processing. The introduction of an intermediate data format can improve the performance of a remote visualization system by:

- Intermediate data generation may reduce the amount of data needed to be transferred from the server to the client.

- Intermediate data format may benefit the client end graphics processing performance compared to the graphics setup by rendering the original dataset.

- Intermediate data format may improve the data reusability compared to the view dependent 2D pixel streams.

A **generalized workload distribution scheme**, referred as remote computation – local view construction, is proposed in this thesis. In this mode, the data flow from the server to the client is not limited to the original geometry or 2D pixel stream. By introducing an intermediate primitive as the output of server computation and the basis of client visualization, the workload can be distributed in a more balanced way throughout the pipeline. In the current implementation, 3D point samples are chosen to be the intermediate primitive. The server computation stage of the pipeline is a 3D point-based sampling process of the original dataset, and the client view construction stage of the pipeline is a point-based splatting process.

To meet the requirement of high-speed VR interaction with the free movement of a head tracking, an **asynchronous coupling mode** between the server and client is

introduced. In this mode, client view construction is isolated from the normal server computation – client visualization work flow cycle. The client continuously reconstructs interim views for arbitrary viewing conditions using available data cached in its local memory. The tradeoff is the possible visual artifacts, such as dis-occlusion, fuzziness, holes and gaps, due to inadequate data coverage and resolution of the currently cached geometry. The visual artifacts are expected to be diminished after a new view update within a short amount of waiting period. The asynchronous server-client coupling mode fits in a scenario where a VR explorer asks for fast navigation of a large-scale dataset and is willing to wait for 1-2 seconds for a complete view updating when s/he wants to examine a particular area of interest.

A sophisticated **dynamic data caching scheme** is introduced to facilitate efficient interim view construction in an asynchronous server-client coupling mode. In the implementation of this thesis, the data caching scheme represents a dynamic point-based incremental data packing with spatial partition hierarchy structure. Based on the data packing, redundancy could be eliminated for each server-end view-updating frame so that the amount of data needed to be transferred from the server to the client is decreased. More algorithms are designed to keep a compact and clean geometry packing along the runtime.

1.3.2 Contributions

A scalable remote visualization pipeline is presented in this thesis for large-scale scientific visualization in a graphics-intensive and frame-rate critical VR environment. Compared to traditional VR visualization systems, the main contributions of this thesis fall into the following categories:

- *Remote visualization strategy*. This thesis presents a remote visualization strategy for VR, which combines distributed data storage, remote computation resources, and state-of-the-art VR techniques.

- *Feature implementations for VR*. Feature implementations for the remote visualization pipeline, such as the generalized remote computation – local view construction workload distribution scheme, asynchronous client-server coupling, and dynamic data packing algorithm, are designed to meet the requirements of the VR interaction.

- *Flexible and extendable framework*. Various datasets and related graphics algorithms can fit into the framework. Available feature functionalities can be switched on or off depending on dataset characteristics and visualization requirements. Missing functionality can be added and out-of-date algorithms can

be improved for better performance. Also, system configuration optimization can be calculated for different visualization tasks.

1.4 Document Organization

After this section, chapter 2 explains the conceptual framework of the proposed system as a scalable remote visualization pipeline, followed by a brief summarization of the functionalities of each component subsystem. Chapter 3 elaborates on the point-based graphics processing algorithms for the proposed remote visualization framework. Point-based processing, such as point sampling, packing and splatting techniques used in this thesis's implementation will be explained in detail, for both surface datasets and volumetric datasets. In Chapter 4, the scalable configuration of the distributed pipeline is discussed, for both intra-subsystems computing and inter-subsystem communication. Chapter 5 discusses different client-server coupling schemes, the supporting algorithms behind them, and the resulting system characteristics. Case studies and experimental results are reported in Chapter 6. Chapter 7 concludes the thesis and talks more about future work.

CHAPTER 2 RELATED WORK

2.1 VR Technologies and Applications

The term 'Virtual Reality' (VR) was initially coined by Jaron Lanier, founder of VPL Research (1989). More recent related terms are 'Virtual Worlds' and 'Virtual Environments' (1990s). In immersive VR, the user becomes fully immersed in an artificial, three-dimensional world that is completely generated by a computer. A variety of input devices like head trackers, data gloves, joysticks, and hand-held wands allow the user to navigate through a virtual environment and to interact with virtual objects. Directional sound, tactile and force feedback devices, voice recognition and other technologies are being employed to enrich the immersive experience and to create more "sensualized" interfaces. The unique characteristics of immersive virtual reality can be summarized as follows:

- *Head-referenced viewing*. Head-referenced viewing provides a natural interface for navigation in three-dimensional space and allows for look-around, walk-around, and fly-through capabilities in virtual environments.

- *Stereoscopic viewing*. Stereoscopic viewing enhances the perception of depth and the sense of space.

- *Realistic interaction*. Realistic interactions with virtual objects via data glove and similar devices allow for manipulation, operation, and control of virtual worlds.

VR technologies, such as the CAVE [Cruz-Neira92] and the Varrier™ [Sandin05], use active or passive stereo technologies to build up stereoscopic or autosterescopic virtual environments. Applications of virtual environments include games, exploration of virtual sites, simulation and training of situational awareness, etc. The automobile industry uses VR installations for design reviews as well as product design. Architectural walk-throughs are common as well.

Unlike traditional VR environments where the world is completely synthetic, augmented reality [Wellner93] tries to enrich the real world with computer generated data. Collaborative environments [Papka97] are the future of virtual reality. These are networked worlds where many users can meet, communicate and work on shared data. One example is the blue-c project [Gross03] for real time collaboration, which combines simultaneous acquisition of multiple live video streams with advanced 3D projection technology in a CAVE™-like environment, creating the impression of total immersion. From multiple video streams, a 3D video representation of the user is computed in real time and streamed to other participants through a networked virtual environment.

Distributed algorithms have also been developed for large-scale VR problems. ([Hudson96] [Huang96])

2.2 Point-based Representation

Recently, many researchers have developed point-based rendering systems where a point representation is generated from the original model and then rendered using splatting techniques to gain high quality and rendering speed.

2.2.1 Model/display primitives: triangles, images and points

Triangles are the most common primitive used in modeling and rendering. Its rendering is fully accelerated in popular graphics hardware. But rendering highly complex models can result in triangles whose projected area is less than a few pixels. Using standard scan-conversion methods for the rendering of these tiny triangles becomes inefficient because of the necessary overhead for the triangle setup.

Images as a modeling/rendering primitive ([McMillan95][Levoy96][Gortler96]) have been used to render complex real world objects with rendering cost proportional to the number of pixels in the image rather than to scene complexity. [Chen95] presented an approach which uses 360-degree cylindrical panoramic images to compose a virtual environment. The image-based approach has been used in the commercial product QuickTime VR, a virtual reality extension to Apple Computer's QuickTime digital

multimedia framework. But still those techniques come with drawbacks such as large memory requirements, noticeable artifacts from many viewing directions, the inability to handle dynamic lighting, restricted position of the viewpoint, and others.

Like image based rendering, point sample rendering makes use of today's large memories to represent complex objects in a manner that avoids the large render-time computational costs of polygons. Unlike image based representations, which are view dependent and will therefore sample the same surface element multiple times, point sample representations contain very little redundancy, allowing memory to be used efficiently. Point based rendering is now used widely in very complex scene rendering where each polygon may only occupy less than one pixel [Rusikiewicz00], and in volume rendering. [Zwicher01b]

Point rendering is in fact quite an old concept. [Csuri79] suggested the idea of using points as primitives to render 3D surfaces more than two decades ago. [Levoy85] used points to render differentiable surfaces. Points have also been used to model fuzzy objects such as clouds, fire, and plants ([Reeves83], [Blinn92], [Smith84]). Benefits of point-based geometry representation include:

- *Conceptual simplicity*. Since no connectivity information exists, only a set of points has to be stored and processed. Hierarchical encoding schemes for point-

based geometry provide compact storage and efficient progressive transmission of these datasets. Recently, several mesh processing algorithms have been reformulated for point-based surface representations, e.g. spectral processing [pauly01], geometry simplification [pauly02], surface editing [pauly03] and multi-resolution shape modeling [zwicker02].

- *Rendering performance*. Points can be rendered extremely quickly; there is no need for polygon clipping, scan conversion, texture mapping, or bump mapping.

- *Rendering quality*. For Point Based Rendering (PBR) the lighting computations are performed on a per point basis, corresponding to high quality Phong shading in the surface case. For anti-aliased rendering, sophisticated splatting techniques assign a Gaussian filter kernel to the splats, resulting in an elliptically weighted average (EWA) filtering of the image similar to anisotropic texture filtering [Heckbert89].

- *Hardware acceleration*. The increasing availability and programmability of graphics hardware has lead to the development of very efficient hardware-accelerated rendering methods, thus providing high visual quality as well as efficient rendering.

PBR also has some problems which are inherent to point representation.

- *Discrete topology*. Points as a primitive can't provide a linear surface representation. Special splatting techniques are needed to fill the holes caused by under-sampling, occlusion and close range zooming.

- *Detailed representation*. For large, flat surfaces with detailed texturing, point rendering becomes less efficient than polygon rendering when the gain of incremental rasterization of polygon rendering outweighs the extra setup required. Specifically, using large textured polygons provides better image quality at lower rendering cost than using a large number of textured points.

[Chen01] presented a hybrid approach, coded POP, in which both points and polygons are used to represent scenes. Points or triangles are chosen during the rendering to guarantee the highest image quality while delivering the maximum rendering speedup. Switching between points and triangles is determined on-the-fly based on their screen projection size.

2.2.2 Point sample acquisition

Point sampling can be done by directly transforming the triangle mesh ([Rusinkiewicz00], [chen01]), orthogonal or perspective projected imaging

([Grossman98], [Pfister00]), ray casting [Hart82] or 3D laser scanning. For different datasets, the term ***point sample*** can either be an abstract 3D point without a sense of spatial extent, a ***surfel*** [Pfister00] which represents a tangent-plane aligned 2-D surface primitive with certain size and shape, or a 3D volume primitive with a certain 3D spatial extent.

The most basic attributes of a ***point sample*** usually consist of a spatial coordinate ***p***, normal orientation ***n*** and color ***c***. Furthermore, it is assumed that each point also contains information about its spatial extent in object-space. For example, the spatial extent of a ***surfel*** usually specifies a circular or elliptical tangent disk centered at ***p*** and perpendicular to ***n***. Elliptical disk *e* consists of major and minor axis directions ***e***1 and ***e***2 and their lengths. Figure 2.1 shows a surfel representation of a curved 3D surface. Other attributes optionally include a normal-cone semi-angle θ, or any additional information for further shading.

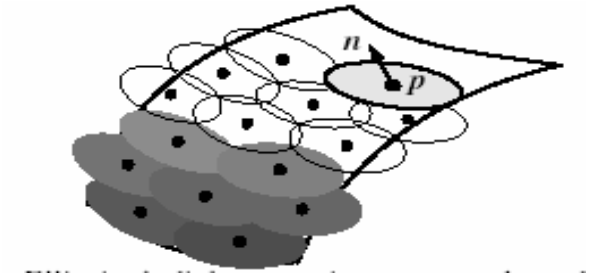


Figure2.1: Elliptical surfels covering a smooth and curved 3D surface.

For correct visibility the point samples must cover the sampled object nicely without holes and thus overlap each other in object-space. An **adequate point sampling** means that the discrete point samples satisfy necessary sampling criteria such as the Nyquist condition, and fully define the object geometry and topology.

Grossman and Wally [Grossman98] present a method for determining the side length of the sampling triangle grids that guarantees adequate sampling of the surface dataset and thus controls sampling density. In QSplat [Rusinkiewicz00] a point sample is a sphere which is created from each vertex of a triangular mesh representing the model. The size of the sphere at a vertex is equal to the maximum size of the bounding spheres of all triangles that touch that vertex. This is a conservative method – it may result in spheres that are too large, but is guaranteed not to leave any holes. Similarly in POP [chen01], each point sample is a bounding tangent disk which is created from each triangle from the triangle mesh. In [Pfister00] surfels are acquired by rasterizing the object on a regular three dimensional rectangular grid. The grid spacing is identical to the pixel spacing of the frame buffer thus controlling the sampling density. Other point based rendering systems [Dachsbacher03] assume a **uniform point sampling** of the surface. [Alexa01] uses local Least Squares approximations to adjust the point sampling for display. Their point set surfaces have been extended to a progressive representation in [Fleishman03].

2.2.3 Point organization and multi-resolution representation

2.2.3.1 Image space organization

Point samples which are organized in image space use image-based 3D warping techniques to assign or interpolate pixels to new positions to form a new view.

Layered depth image introduced by Shade in [Shade98] organize point samples in a view-dependent manner in image space. A fast incremental warping algorithm as well as a method for calculating the splat size is used to render LDI at a speed of multiple frames per second on a PC within a limited viewing range. In point sample rendering [Grossman98], the point samples from each projection (image) are grouped into blocks of 8*8 samples and a greedy algorithm is used to collect a suitable subset of all blocks needed to represent the whole object while avoiding redundancy. Surfels [Pfister00] arrange point samples in three orthogonal LDIs by blocks. The rendering process can then be accelerated by using incremental calculations.

A point model organized in image space is usually re-sampled at a lower resolution (bigger pixel spacing) to get multi-resolution samples [Grossman98, Pfister00].

2.2.3.2 Object space organization

Most point models in object space organization use some sort of a hierarchical space-partitioning data structure as multiresolution representation. The most often proposed structures are octrees in which the region-octree with regular subdivision has been favored in [Ren02, Botsch02, Botsch03]. In [Rusinkiewicz00] a midpoint-split kD-tree and in [Pajarola03a, Pajarola04a] an adaptive point-octree are used. The latter two offer data-adaptive hierarchies with fewer nodes than regular subdivision approaches.

Basically, a LOD point-hierarchy stores aggregate information in each node, such as centroid position, normal and bounding volume information about all points in its subtree. An extremely memory efficient point LOD-hierarchy is given in [Rusinkiewicz00]. Aggressive quantization techniques and look-up tables are used to reduce the cost to represent a point $\boldsymbol{p}$ and bounding sphere radius r in only 13 bits, and the normal $\boldsymbol{n}$ in 14 bits. The color $\boldsymbol{c}$ is quantized to 5-6-5 bit and the normal-cone semi-angle θ to 2 bits. The tree structure uses 3 bits in each node to encode the number of children. The LOD-hierarchy is laid out in breadth-first order in an array with each group of siblings sharing one pointer (index) to their list of consecutive child nodes. In [Botsch02] an octree is proposed that implicitly encodes the point coordinates $\boldsymbol{p}$ as the center of a cell in the recursive octree subdivision. A byte-code of the subdivision provides the tree branching information at each node. The normal $\boldsymbol{n}$ and color $\boldsymbol{c}$ are quantized to less than 2bytes and 1byte respectively. No bounding sphere size is used as

it is implicit in the hierarchy and a normal-cone semi-angle is optionally maintained in non-leaf nodes only. During hierarchy traversal, due to the lack of explicit parent child links, back-tracking at a node is only supported by actively skipping its entire subtree without performing any operations. Such compact encodings of the LOD hierarchy and point attributes lead to storage costs of only a few bytes per point which in turn allows the representation of several 100 million points within the 4GB virtual memory addressing limit of 32bit systems. This is a significant benefit over methods with more complex node formats.

2.2.4 Point-based Rendering: Surface Splatting

Point splatting techniques are used for a hole-free view reconstruction when the contribution of one sample point is spread to multiple pixels in the frame buffer. Each point is associated with a 3-D rendering primitive that is projected onto the image plane. Splats provide a good compromise between the quality and complexity of the geometry representation. The **choices of primitive** to render a point sample include:

Points: Several approaches (i.e. [Rusinkiewicz00, Dachsbacher03, Botsch02]) have proposed to use simple OpenGL point primitives, which have the advantage of a low cost per primitive (3D position, color and normal if lighting is required). The primitive is drawn on screen as a fixed sized square, or rounded point with GL_POINT_SMOOTH enabled, thus using a box reconstruction kernel.

Moreover, with the use of vertex and fragment programs and recent extensions, the size of points can be calculated on a per-primitive basis to be the actual screen-projected size of a point sample, improving the visual quality by avoiding conservatively large points and holes between rendered points.

Sprites: Another choice for point primitives consists of using POINT SPRITES as promoted in [BK03] which can be considered textured points. This primitive combines the simplicity of points for geometry submission to the graphics card with the flexibility of texture mapping with blending kernels to support smooth interpolation of discrete points and hence visually higher-quality renderings. With POINT SPRITES a single coordinate is specified per point and the graphics card rasterization unit generates a quadrilateral with texture coordinates. As presented in [Botsch03], with some work these sprites can be modified to represent surface-normal oriented disks, rendered with proper per-pixel depth values using graphics card programmability. Moreover, smooth blending can be achieved by computing a per-pixel α-value in the fragment program.

Triangles: The third hardware supported primitive type is triangles and polygons. In [Ren02] and [Pajarola04a] polygonal faces are used with the α-texture which provides a disk or elliptical shape as desired (using α-tests). In fact, the α-texture can describe any desired blending kernel mapped onto the elliptical point splat

primitive. The system presented in [Rusinkiewicz00] also allows the use of oriented solid polygonal disks which tend to run significantly slower as they are made of many vertices. The use of more complex primitives than simple points has the advantage that α-texture mapping and blending kernels can be used to obtain smoothly blended points and more realistic rendered surfaces.

Depending on the type of point-primitives chosen for display, different rendering strategies are necessary. The key factor for this is whether blending kernels are used on the points. If blending is performed then it is necessary to ensure that only the front-facing points closest to the viewpoint are combined. If there exist front-facing points farther away, occluded from the viewpoint, it must be assured that these are not blended with the closest visible points. This can be achieved by carefully selecting just the closest overlapping points. Commonly a two-pass ϵ-z-buffer rendering approach [Ren02, Pajarola03b, Botsch03, Pajarola04a] works efficiently: the first pass initializes the z-buffer to generate a depth mask without rendering to the color buffer, and the second pass only performs z-buffer tests for each pixel fragment against some ϵ offset of the z value from the first pass. Hence when rendering opaque point primitives with no blending, only a single pass over the data is performed, but when polygons or sprites are used with smooth blending a two-pass approach is required. Although the first pass is less expensive than the second one, it still requires the geometry to be processed twice by the graphics hardware.

Sophisticated **high-quality splatting algorithms** resolve aliasing issues. [Zwicker01] introduces surface splatting through image-based EWA filtering, resulting in high quality anti-aliased rendering, comparable to anisotropic texture filtering [Heckbert89]. While this software based approach is only able to process 250k splats per second, it provides the highest visual quality. An approximation using look-up tables has been presented by [Botsch02] which reduces the computational complexity of EWA splatting. Their method is able to process up to 14M points or 4M high quality filtered splats per second by using a quantization of splat shapes. [Ren02] reformulates the image based EWA filtering of [Zwicher01] to object-space filtering in order to map the surface splatting approach to graphics hardware, also using a two-pass rendering method. They render each splat as a textured rectangle in object-space. This concept causes the number of processed points to be multiplied by four, slowing down the rendering to about 2M-3M splats per second.

Many point based rendering techniques use modern **programmable graphics hardware** to rasterize a circular or elliptical reconstruction kernel for a point primitive. For example, a vertex shader can be designed to determine the OpenGL point size for each point primitive passed to the GPU [Dachsbacher03] and reports splatting performance at 50M splats per second. The points are rendered as unfiltered view-plane aligned small squares. [Botsch03] uses programmable graphics hardware (vertex shader and fragment shader) to render up to 28M mid-quality surface splats per second on the

latest graphics hardware. They also use a two-pass splatting technique with Gaussian filtering to render up to 10M high-quality surface splats per second. In [Zwicker04], a perspective-accurate elliptical weighted average (EWA) splatting method is presented to rasterize elliptical splat kernels. The vertex shader can be used to compute the bounding box of the ellipse and the fragment shader to define the pixel color within the elliptical coverage This method is able to render up to 3M high-quality surface splats per second on the latest graphics hardware using a 3-pass algorithm. Phong splatting [Botsch04] enables Phong shading in point rendering by associating a linearly varying normal field with each splat instead of keeping the normal constant. They thereby achieve the same visual quality as Phong shaded polygons. Table 2.1 is a summary of different point based rendering method.

Table 2.1: Point based surface splatting techniques

Paper Reference	**Reconstruction kernal**	**Speed (points/second)**	**rendering requirement**
[Dachsbacher03]	Solid square	50M	GL_POINTS + Vertex program
[Botsch03]	Solid circle/oriented ellipse	28M	POINT SPRITES with solid color texture + Vertex and fragment program
[Rusinkiewicz00]	Solid circle/oriented ellipse	4-5M	Polygon with solid color texture
[Zwicker01]	Screen space EWA (Gaussian filtered oriented ellipse)	250K	software
[Botsch02]		4M	Lookup table for quantization of splat shapes
[Botsch03]		10M	POINT SPRITES with Gaussian filtered texture + Vertex and fragment program
[Ren02]	Object-space EWA	2-3M	α-textured polygon
[Zwicker04]	Screen space perspective-accurate EWA	3M	Vertex and fragment program

2.2.5 Point-based Rendering: Volume Splatting

Generally, volume rendering algorithms can be divided into two categories. One is the image-order method, as in ray casting [Kajiya84]. The other is the object-order method. Splatting is an object-order high-quality volume re-sampling and compositing technique, where each voxel's contribution is accumulated in an image buffer using a projected reconstruction kernel called a footprint. It can incorporate a variety of reconstruction kernels without extra computational overhead, as well as reduce computation and storage costs using a sparse volume representation that holds only non-transparent voxels. [Westover89] first presented the splatting technique for volume rendering. [Westover90] solved the inaccurate visibility problem using an axis-aligned sheet buffer. [Mueller98] proposed to align the sheet buffers parallel to the image plane instead of parallel to an axis of the volume data. This technique is similar to *slice-based* volume rendering [VanGelder96] [Cabral94] and does not suffer from popping artifacts. [Mueller96] combined splatting with ray casting techniques to accelerate rendering with perspective projection. [Laur91] describes a hierarchical splatting algorithm enabling progressive refinement during rendering. Furthermore, [Lippert95] introduced a splatting algorithm that directly uses a wavelet representation of the volume data. To render curvilinear grids, [Mao96] use a stochastic Poisson resampling to generate a set of new points whose kernels are spheres or ellipsoids. They compute the elliptical footprints very similar to [Westover90]. [Swan97] used a distance-dependent stretch of the footprints to make them act as low-pass filters. [Zwicker01b] developed EWA volume splatting along

similar lines to [Heckbert89], who introduced EWA filtering to avoid aliasing of surface textures. The software-based volume splatting technique can achieve high image quality but is only able to process 250k splats per second.

2.3 Parallel Computing

During the past 10 years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures suggest that parallelism is the future of computing. Parallel computing is used widely today to solve large problems in real-time by providing concurrency. Based on how parallel computing resources are geographically connected, a parallel system could be a supercomputer with multiple processors, a computing cluster, or a distributed system which connects either single computers or parallel computers. The primary advantages for using parallel computing include:

- *More computing resources*. Use available compute resources on a high speed network when local compute resources are scarce.

- *Cost savings*. Use multiple "cheap" computing resources instead of one expansive supercomputer.

- *Overcoming memory constraints*. For large problems, use the memories of multiple computers where a single computer has finite memory resources.

2.3.1 Architecture Classification

There are different ways to classify parallel computers. One of the more widely used classifications is called Flynn's Taxonomy [Flynn72]. Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of ***Instruction*** and ***Data***. Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple***. The below paragraphs explains the 4 possible classifications according to Flynn.

- *Single Instruction, Single Data (SISD).* This architecture represents a serial computer which exploits single data stream against single instruction stream during any one clock cycle.

- *Single Instruction, Multiple Data (SIMD)*. This architecture represents unit repetition which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelised.

- *Multiple Instructions, Single Data (MISD).* Single data stream is fed into multiple processing units. Each processing unit operates on the data independently via independent instruction streams.

- *Multiple Instructions, Multiple Data (MIMD).* Multiple autonomous processors simultaneously execute different instructions on different data. Computing pipelines are generally recognised to be MIMD architectures.

2.3.2 Parallel Rendering Algorithms

Parallel rendering algorithms have been widely used in large-scale visualization problems. Generally, the key design concepts in a parallel program include data partition, communication, synchronization, and load balancing among parallel tasks. The combination of different implementations of these design concepts features different parallel algorithms.

For example, for a cluster-parallel graphics rendering task, parallel rendering strategies fall within three main categories, depending on which stage of the rendering pipeline sorting for visible-surface determination takes place [Molnar94]. These categories are sort-first, sort-middle, and sort-last. Sort-first approaches divide the 2D screen into disjoint tiles, and assign each region to a different processor, which is responsible for all the rendering in its tile. Sort-middle approaches assign an arbitrary

subset of primitives to each geometry processor, and a portion of the screen to each rasterizer. A geometry processor transforms and lights its primitives, and then sends them to the appropriate rasterizers. Sort-last approaches assign an arbitrary subset of the primitives to each renderer. A renderer computes pixel values for its subset, no matter where they fall in the screen, and then transfer theses pixels (color and depth values) to the compositing processors.

Most systems chose to use a sort-first approach, because sort-first processors implement the entire pipeline for a portion of the screen, which is exactly the case for which PC graphics cards are optimized. A sort-middle approach requires tight integration between the geometry processing and rasterization stages, which is not available in PC graphics cards. A sort-last approach requires high pixel bandwidth, which is also not available in graphics PC cards.

Chromium [Humphreys02] is a system for manipulating streams of graphics API commands on clusters of workstations. Chromium's stream filters can be arranged to create sort-first and sort-last parallel graphics architectures that, in many cases, support the same applications while using only commodity graphics accelerators.

CHAPTER 3 CONCEPTUAL FRAMEWORK

3.1 Chapter Organization

In section 3.2, the design concepts of the scalable remote visualization pipeline are expatiated and a framework diagram is provided to show the pipeline design and data flow. The generalized functionality assignments for each subsystem are elaborated on in the section 3.3. Section 3.4 summarizes this chapter by emphasizing the contribution of the framework design to solve large-scale VR problems.

3.2 Concept of the Framework Design

The purpose of this thesis is to design a real-time remote visualization framework to solve large scale VR problems by taking advantage of remote computation resources. The structure of the framework is a distributed pipeline with a MIMD computing architecture. The data retrieving, generating, and consuming functionalities are executed by distributed computation modules, and data flow through these modules during the pipelined processing.

The distributed remote visualization pipeline presented in this thesis involves four modules: the command master, the data server, the computation server, and the visualization client. The command master usually resides together with the visualization

client and its purpose is to send meta-data and workflow control commands to the other three modules. The other three modules are computation subsystems; each of them applies different functionalities to the data flow over the pipeline. Figure 3.1 shows the framework diagram and the data flow through the pipeline.

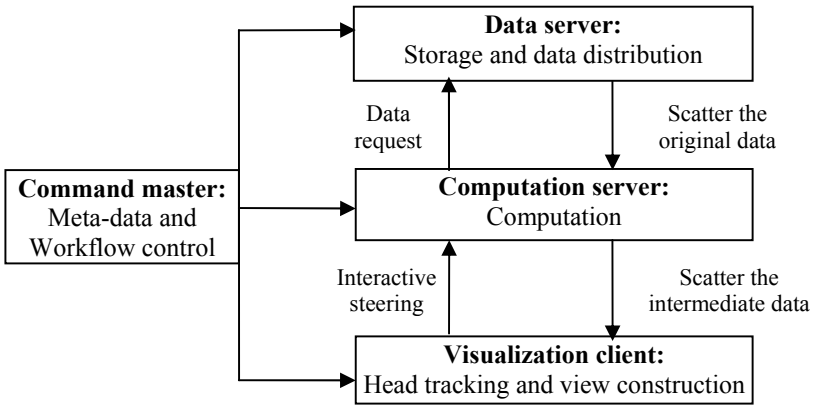


Figure 3.1: The framework diagram of the distributed pipeline

Each computation module is defined by the input data stream, the output data stream, and the data processing algorithm. Communications among the modules are network transfer algorithms multiplexed with computation. The design concepts are:

- Every computation module is a functionally independent data processing system with input and output data stream. Different data processing functionalities can be applied for different dataset and visualization characteristics, without affecting the data flow model of the pipeline.

- Each computation module is a scalable system. Different data partition schemes can be applied as part of the scalable computing. The scalability of data communication among the modules is adaptive to the scalability of the modules.

- The design of the whole pipeline should achieve good load-balancing for real-time large-scale VR visualization task. This means efficient workload distribution and data exchanging mode between the remote server computation and local VR view construction.

- The design of the pipeline should maintain proper granularity, adaptive to the module computation performance and network condition.

Later in this thesis, each computation module is referred to as a subsystem within the pipelined computing architecture.

3.3 Subsystem Functionalities

At the implementation level, each subsystem in the remote visualization framework represents a SIMD parallel rendering program in the distributed MIMD parallel pipeline. As an independent data processing unit with input and output data

stream, every computing subsystem has specific functionality assignments to serve in a generalized VR visualization pipeline for various types of scientific datasets.

3.3.1 Data server

The data server stores the original dataset and distributes data to another party upon authorized data request. Necessary pre-processing can be applied to the original dataset according to specific data requests before the actual data transfer. For example, the data requesting party will send a transfer function to data server when a volumetric dataset is involved. The transfer function will be applied to the original volume data by the data server computation and only non-transparent voxels will be actually transferred over the network.

A data scattering algorithm is used by the data server to transfer data to the data requesting party. The scalability of the data scattering algorithm is adaptive to the scalability of both the data server and data requester. The design of the data scattering algorithm depends on both the existing data partition at the data server and the desired data partition of the data requesting party.

3.3.2 Computation server

The computation server takes the data received from the data server as input and transforms them into an intermediate data form according to the VR client's viewing

demand. As in this thesis, the data transformation is usually implemented by a point-based sampling or re-sampling process. Other related data processing functionalities can be applied before data flow into the next stage in the pipeline.

The intermediate data generated by the computation server are scattered to the client for view construction. A data scattering algorithm is used for efficient data transferring. The scalability of the data scattering algorithm is adaptive to the scalability of both the computation server and visualization client. The design of the data scattering algorithm depends on both the resulting data partition at the computation server and the required data partition at the visualization client.

3.3.3 Visualization client

The visualization client sends steering requests to the remote computation server and receives intermediate data from the computation server. The fact that the client actually requests the intermediate data generated by the server not only reduces the data communication compared to directly requesting data from the data server, but also improves the view construction performance than directly rendering the original dataset locally. Taking the intermediate data as input, 2D stereo viewings are reconstructed for the current viewing condition. As in this thesis, point-based splatting techniques are used for efficient and high-quality 2D view construction.

If a data packing stage is included in the client-end subsystem implementation, view construction can be performed asynchronously with receiving data from the computation server. This is where the asynchronous client-server coupling stands. Asynchronous client-server coupling trades data flow integrity along the visualization pipeline for high frame rate interim view updating.

3.4 Summary

The distributed pipeline presented in this thesis serves as a remote visualization framework for large-scale VR problems. The pipeline design is suitable for different data types and is adaptive to various computation resources and network conditions. Scalable system configuration in combination with adaptive computation algorithms makes a flexible and efficient framework to serve the requirements of VR.

CHAPTER 4 POINT-BASED GRAPHICS FOR VR

4.1 Chapter Organization

In this thesis, point samples are introduced as the intermediate primitive through the server computation and client visualization pipeline. By using 3D point samples as an intermediate data format other than a triangle set or 2D pixel stream, the workload can be distributed in a more flexible and balanced way throughout the remote visualization pipeline. Different point sample representations, point-based graphics algorithms and other related functionalities are described in this thesis to facilitate various dataset visualizations for VR.

For surface datasets which represent the exterior characteristics of an object, a point sampling function is provided first to sample the original surface into small surface areas called ***surfels***. A point modeling function is also available to pack surfels into a compact point-based geometry representation. Surface splatting techniques are discussed in detail to splat surfels onto the screen for a final seamless view construction. Surface splatting is a re-sampling process from the surfel sampling space to the output screen space.

Volumetric datasets represent a 3D sampling of the interior structure of objects, including amorphous and semi-transparent features, over a uniform/non-uniform 3D grid. A volume is a compact point-based embodiment of an object with each voxel as a point sample. Volume splatting is a re-sampling process from the voxel sampling space to the output screen space.

Section 4.2 and its subsections discuss the point-based sampling process for surface datasets. Section 4.3 and its subsections discuss the surfel packing process and its related function implementations. Section 4.4 elaborates on the surface splatting algorithms. Section 4.5 explains the volume splatting algorithm in the shear-warping context, and its parallel extension for a distributed system. Section 4.6 explains that the point-based graphics processing functions can be performed in a distributed computing architecture thus taking advantage of remote storage and computing resources. Section 4.7 summarizes this chapter by pointing out that sophisticated point based graphics algorithms are studied and implemented to support the point-based VR visualization pipeline presented in this thesis.

4.2 Point-based Sampling for Surface Datasets

Point-based sampling of a surface dataset transforms the original continuous 3D surface into a discrete point sample (surfel) representation.

4.2.1 Point Sampling of the Mesh Dataset by Rasterization

The most straightforward point sampling algorithm for a mesh dataset is performed through the traditional rendering pipeline by graphics hardware-based triangle setup and rasterization. A perspective sampling is produced by projecting the visible part of the dataset onto discrete 2D grids of color map and depth map as in perspective viewing. The color map and depth map are collectively henceforth termed "depth-image". The Depth Image-Based Representations (DIBR) as a new family of 3D geometry representation [Ignatenko03] has been adopted into MPEG-4 Part16: Animation Framework eXtension (AFX).

In this thesis, the term ***sample rate*** indicates the object-space point sample distribution. Perspective sampling causes non-uniform surfel distribution. Therefore, the output point sample rate of the same surface geometry differs under different viewing transformation. Figure 4.1 show that each pixel in screen space from different viewing transformations implies different surfel distribution in object space.

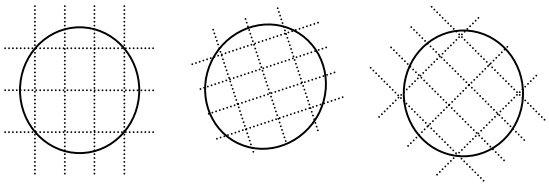


Figure 4.1: Different sample rate of the same surface geometry

Beside the color map and depth map, a normal map can also be retrieved from the sampling process, thus a normal attribute can be assigned to each point sample for more sophisticated shading in later view reconstruction. Saito and Takahashi propose a general framework for image-space rendering algorithms called G-buffer [Saito90]. This buffer forms an enriched image space, also referred to as 2.5D image space, whereby each pixel of the image space holds arbitrary additional information, such as normal, depth, or texture coordinates. In order to fill the G-buffer with the required information, it is necessary to set the G-buffer as the current render target. Similar GPU-based G-buffer implementations can be inferred from deferred shading algorithms [Policarpo05].

For parallel point-sampling of a mesh dataset, the original transformation frustum is first divided into smaller portals and then one sampling process can rasterize the mesh inside one of the small portals in parallel with all other sampling processes. The parallel sampling algorithm by frustum-dividing is scalable.

4.2.2 Point Sampling of the Mathematic dataset by Ray-tracing

Some mathematical models can be visualized in the form of a 3D shape, such as the quaternion Julia sets [Nortan82]. The point sampling of the quaternion Julia set is computed by ray-tracing the intersection points on the Julia surface onto a 2D discrete grid. Each point sample has position and incident color attributes. The Julia coloring is determined by the distance from the sampled surface point to the center of the Julia set. A

surface normal determination algorithm is also used to assign a surface normal to a point sample on a non-differentiable surface.

The ray-tracing based point sampling of a mathematical model is slow. Experimental results show that on a Linux machine with dual Intel Xeon 1.8 GHz processors, generating a ray-traced Julia set image with the resolution of 1024*640 usually takes about 1 minute. Fortunately, the ray-tracing based Julia set computation is amenable to parallel processing because each point position on the Julia set surface hit by the ray for each pixel can be independently computed. The parallel ray-tracing algorithm is scalable.

4.3 Point Sample Packing

The point sampling process usually carries on through multiple or continuous sampling frames. Point sample packing is a dynamic modeling process which incrementally builds up a compact point model along the sampling frames. Real time point sample packing is important for a real-time visualization pipeline. Along with the packing process, related functions include redundancy elimination and obsolete data deletion.

4.3.1 Redundancy Elimination

To maintain point geometry compactness, it may be desirable to delete repetitive sampling among multiple sampling frames during point packing.

For surface sampling, an existing surfel becomes redundant when the 3D geometry it represents is sampled better by another surfel. The quality of a surfel sample on a continuous surface indicates how well a single-colored oriented small ellipse matches the geometry and illumination of the part of surface coverage it represents. There can be different ways to compute a quantified value indicating a surfel's sampling quality. For example, for an elliptical surfel with long axis radius r_0, short axis radius r_1, color c and normal n. A simple formula to compute a quantified value as surfel quality is:

$$q = (r_1 / r_0) * (1 / r_0) \quad (4.1)$$

Here q is in direct proportion to the roundness of the ellipse, and in inverse proportion to the size of the ellipse. It means that a good surfel sample is expected to be in high resolution with a disk shape, so that the color and normal attributes of the surfel can be a more reasonable approximation to represent a small continuous surface area.

For a new sampling frame, accurate redundant surfel elimination involves expensive sampling quality computation and possible surfel deletion inside both the

existing packed model and the current sampling. It's difficult to implement real-time point packing for VR visualization with expensive surfel sampling quality computation every frame. Besides, data deletion of an existing point model can break the integrity of the current packing and involve a complicated data identification scheme. For simplicity, an approximate redundancy elimination algorithm can be applied instead by avoiding the sample quality computation and only deleting redundant points in current sampling. The algorithm is introduced below:

Suppose a current surface sampling includes a color map *C* and a depth map *Z*. A normal map is not required since this algorithm will not involve the surfel computation. First, build reference color and depth maps by rendering the available point samples under the current viewing condition without splatting. Compare the current depth map *Z* and reference depth map *refZ* using the following pseudo code:

```
for every pixel i in Z and refZ
{
      if ( Z(i) < refZ(i) – ε ) keep pixel i in current maps
      else discard pixel i in current maps
}

Here ε is a small threshold to remedy the possible difference of
depth maps sampled from triangle meshes and previously extracted
3D points.
```

This algorithm is very simple because it only needs one map comparison operation per pixel. Also it ensures data integrity after the data is actually packed, because it only deletes redundancy in the current sampling map. The tradeoff is its

inability to delete previous low resolution data. Instead, some of the high-resolution data in current sampling is sparsely deleted. This will impair data integrity along the packing and cause problems in the final view construction stage. Discretion should be taken when applying this algorithm during the packing process.

Figure 4.2 shows an example of the new color map, the reference color map and the resulting non-redundant color map after redundancy elimination. From the maps we can see that all the pixels in the current sampling map are deleted if they have already appeared in the reference map with same depth value.

(a) (b) (c)

Figure 4.2: Example color maps during the redundancy elimination.

(a) Reference map; (b) Current map; (c) Resulting non-redundant map

4.3.2 Point Sample Packing with Spatial Partition Hierarchy

Because of the conceptual simplicity of using points as the modeling primitives, the extracted point samples can be packed together without topological connection

enforcement to form a compact and photorealistic representation of the original 3D geometry. For surface datasets, usually non-uniformly distributed point samples are extracted from the sampling process frames with overlapping geometry coverage. For example, a point sample will be exacted from each pixel in the sampled maps (color map + depth map + normal map) and multiple sample rates are naturally introduced for perspective sampling projections. A surfel representation of a point sample usually consists of spatial coordinate ***p***, normal orientation ***n***, color ***c***, and information about its spatial extent in object-space. There could also be abstract point samples which don't assume a spatial extension. Furthermore, the simplest point sample representation may only include a spatial coordinate ***p*** and color ***c***.

A point-based geometry is composed of point patches and is reconstructed incrementally and dynamically during run time. A ***point patch*** is a collection of point samples extracted from each sampling frame. Compared to existing point patches, a new point patch has either new geometry coverage or different sample rate. The dynamic (real-time) patch definition differs from pre-processed geometry segmentation because it is completely view-dependent and will be different for different sampling sequence. If map decimation is applied as a pre-processing step before the actual point packing, a set of multi-resolution point patches will be produced from one sampling frame which represents the same data with different levels of detail. In this thesis, the term ***resolution*** is used to indicate the sample density because of decimation. The point packing is

naturally a **multi-sample rate point packing** if point patches are extracted from different perspective sampling projections. A **multi-resolution point packing** means that each point patch has multiple LOD levels from decimation.

For local storage, the point model is partitioned into ***point clusters*** and organized into a level-limited octree, with each leaf node representing a point cluster inside its bounding box. The point patches extracted from every sampling frame are partitioned into mosaics to fit into the octree leaf nodes bounding boxes, and every mosaic has sequential memory storage for fast data access. For a point cluster, its component point patch mosaics from different sampling frames are linked together. The octree-based space partition hierarchy of the point geometry enables fast frustum culling and efficient data access. For one point cluster, each of its component point patch mosaics has an attribute called ***pixel range*** which is a quantified indication of its data resolution. The term ***pixel range*** refers to the projected pixel coverage of a point cluster's bounding box under a certain viewing condition. A normal cone [Shirman 93] attribute can also be added for each point patch mosaic as an extended normal vector for all of its component point samples. Back face culling functionality can be enabled by the normal cone attribute. Other attributes of a point patch include the point sample number and the center position as the average center position of its component point samples. The octree leaf node keeps the record of memory storage of its point cluster. Figure 4.3 shows the data structure of an octree leaf node.

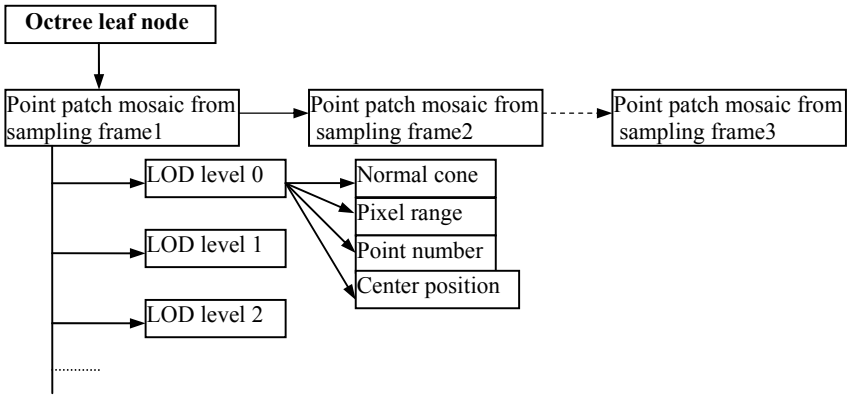


Figure 4.3: Data structure of an octree leaf node in point packing

Figure 4.4 shows an example of the component point patch and point cluster representation of a point-based geometry, both signified by different colors. In the example, the simplified redundancy elimination algorithm discussed in section 4.3.1 is applied in the creation of each point patch.

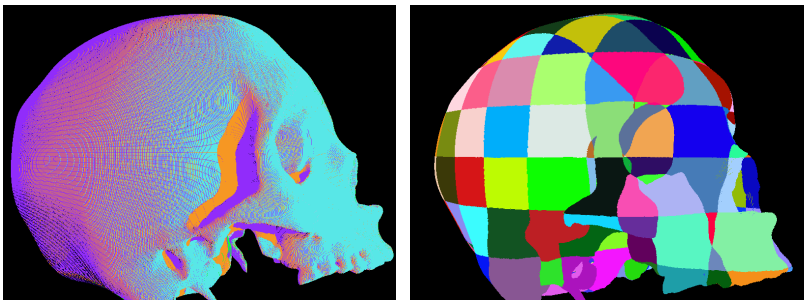


Figure 4.4: Point patch and point cluster representation of a point packing.
Both images are signified by different colors. Left: Point patches; Right: Point clusters

4.3.3 Obsolete Data Deletion

Usually an obsolete data deletion mechanism is based on a point cluster's most recent access time, so that a most recent point model can always be maintained and fit into the client's main memory. To facilitate the obsolete data identification, the ***access time*** attribute of a point cluster's octree leaf node will be updated every time it's selected for splatting. Below is the pseudo code of the obsolete data deletion algorithm:

```
for every second, search each leaf node of the octree
{
      Assume current time is t and the node's latest access time
is nt
      if (t - nt) > ϵ
      {
            Calculate the current node's ID number;
            Delete this node's point cluster storage;
      }
}
```

The threshold ϵ indicates the no-access period beyond which the points will be considered to be obsolete. Obsolete data deletion is especially important when dealing with visualization with data animation or deformation, similar to the particle system implementations. [Reeves83]

Other data deletion mechanism can also be applied to maintain the effectiveness of the point model. For example, the sample rate of the packed data can also be taken into consideration as part of data deletion, so that very detailed sampling can be deleted to maintain compact and complete geometry coverage.

4.4 Surface Splatting

During view reconstruction, all the point samples are sent to the graphics card. The graphic hardware rasterizes the output re-sampling kernel for each point primitive, resulting in a final seamless image.

The point rendering functionalities for surface splatting provided in this thesis incorporate the following features:

- Rendering primitives: resizable GL_POINTS for fast yet low-quality splatting; oriented elliptical POINT SPRITES with solid color texture for medium-quality splatting; or oriented elliptical POINT SPRITES with Gaussian filtered texture for high-quality splatting.

- One-pass rendering algorithm for opaque resizable points, and a two-pass rendering for blended primitives.

- Vertex and fragment program for per-splat GPU programming.

Other surface splatting techniques, such as polygon-based object space EWA splatting and screen space perspective-accurate EWA splatting, are not included because of their low speed for VR.

4.4.1 Point Sample Splatting: Input to Output Screen Space Resampling

The combination of point-sampling and point-splatting can be expressed as a re-sampling process from screen space to screen space, which is a concatenation of a 2D input screen space to 2D surfel space back-projective mapping, followed by 2D surfel space to 2D output screen space projective mapping. Here the term ***surfel space*** refers to the 2D local surfel parameterization in 3D object space. Given a circular input screen space sampling kernel, the contour of the output screen space re-sampling kernel is a general ellipse. Figure 4.5 shows the re-sampling kernel contour transformation from input screen-space to output screen-space. When rasterizing an oriented elliptical POINT SPRITE with either solid color or Gaussian filtered texture in a perspective splatting, it's crucial to know the point sample's 3D position and its projected contour in the final 2D view space.

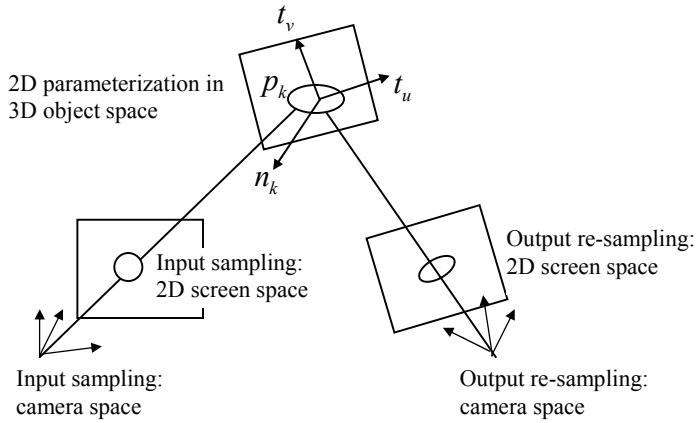


Figure 4.5: The screen-space to screen-space re-sampling transformation.

According to [Zwicker04], a contour-preserving 2D-to-2D projective mapping between two centric conics can be expressed by first applying a 3D affine mapping to the original conic in its homogeneous coordinates, and then transforming the projectively mapped homogeneous conic into its central form. A tight axis aligned bounding box can be calculated from the conic equation.

Firstly, let's consider the transformation for the input sampling process. A central conic in the input sampling's 2D screen space can be expressed in matrix form by:

$$[x\ y]\begin{bmatrix} A & B \\ B & C \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = F, \quad or \quad XQX^T = F, \tag{4.2}$$

Q is the conic matrix. Rewrite the conic in homogeneous coordinates, also in matrix form:

$$[x\ y\ 1]\begin{bmatrix} A & B & 0 \\ B & C & 0 \\ 0 & 0 & -F \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0, \quad or \quad XQ_hX^T = 0, \tag{4.3}$$

The projective mapping from a 2D screen space conic to a 2D surfel space conic can be expressed in homogeneous coordinates as following:

$$Q_h^{'} = M_i Q_h M_i^{T}, \quad where \quad M_i = \begin{bmatrix} t_{iu} \\ t_{iv} \\ p_{ik} \end{bmatrix}$$

$$t_{iu} = R_i \cdot t_u, \quad t_{iv} = R_i \cdot t_v, \quad p_{ik} = R_i \cdot p_k + T_i, \tag{4.4}$$

Here t_u and t_v are the two basis vectors for the 2D surfel space, and p_k is the origin of the surfel. Given the rotation matrix R_i and the translation matrix T_i from object space to input sampling camera space, t_{iu} , t_{iv} and p_{ik} are the corresponding 3D vectors in input sampling camera space.

The second transformation is another projective mapping between 2D surfel space to 2D output screen space. Again, the conic transformation in homogeneous coordinates can be expressed as following:

$$Q_h^{''} = M_o^{-1} Q_h^{'} M_o^{-1^{T}}, \quad where \quad M_o = \begin{bmatrix} t_{ou} \\ t_{ov} \\ p_{ok} \end{bmatrix}$$

$$t_{ou} = R_o \cdot t_u, \quad t_{ov} = R_o \cdot t_v, \quad p_{ok} = R_o \cdot p_k + T_o, \tag{4.5}$$

Similarly, given the rotation matrix R_o and the translation matrix T_o from object space to the output re-sampling's camera space, t_{ou} , t_{ov} and p_{ok} are the corresponding 3D vectors in the output re-sampling's camera space.

So, by combining the two above projective mappings, we get the homogeneous matrix for the output re-sampling conic:

$$Q_h^{''} = M_o^{-1} M_i Q_h M_i^T M_o^{-1^T} = \begin{bmatrix} a & b & d \\ b & c & e \\ d & e & -f \end{bmatrix}, \tag{4.6}$$

Finally, the homogeneous conic is transformed into a central conic:

$$(x - x_t) Q^{''} (x - x_t)^T = f - dx_t - ey_t$$

$$\text{Where } Q^{''} = \begin{bmatrix} a & b \\ b & c \end{bmatrix},\ x_t = \frac{be - cd}{ac - b^2}, y_t = \frac{bd - ae}{ac - b^2}, \tag{4.7}$$

This is the computation of re-sampling 2D contour conic from input to output screen-space. Usually the input screen-space sampling kernel is a disk, the object-space surfel kernel is a general ellipse and the output screen-space kernel is another ellipse. The exact computation is quite expensive, since it depends on two concatenated 2D-2D

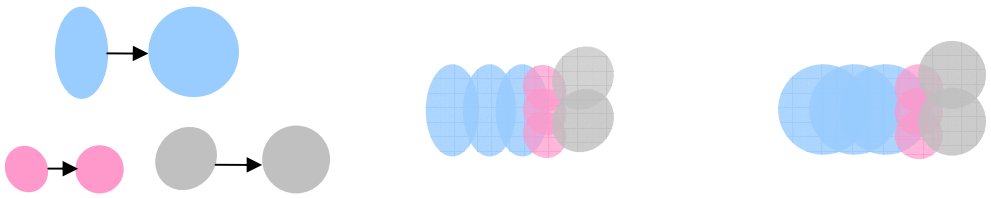
projections, the surfel position, and the surfel normal. As for a point sampling by graphics hardware based rasterization, a normal map has to be computed for per-sample normal information. In ray-tracing based point sampling, a surface normal determination algorithm is also used to assign a surface normal to a point sample on a non-differentiable surface.

4.4.2 Splatting Algorithms with Different Kernel Selection

Applying different splat kernel for point-based view reconstruction directly affects the final image quality.

For perspective projection, using point sprites with an elliptical Gaussian filtered texture as the output screen-space splat kernel has a high image reconstruction quality. The per-splat size and shape computation can be done by GPU programming. For each splat, a 2x2 contour mapping matrix is computed in a vertex program and multiplied to the texture coordinates of the proxy polygon of the sprite, so each sprite has the correct size and shape after projection. Two pass rendering is needed for correct blending. Even though it provides high-quality image in the final view reconstruction, the contour mapping computation is quite expensive, and only achieves about 10M splats/second rendering speed. This is not fast enough for a VR visualization requirement.

To trade quality for speed, approximations are made as for the splat kernel selection. Using GL_POINTS with a circular splat kernel for point primitives can avoid expensive computation of the projective elliptical contour mapping. Figure 4.6 shows the approximation of an elliptical splat kernel as a circular splat kernel and how it causes blurriness in the final view reconstruction. As an example, a correct view construction using elliptical splatting kernel is shown in (b) and the same view construction with approximated circular splatting kernel is shown in (c). Compared to (b), (c) looks fuzzier because the circular splats are fatter than needed and the overlapping area is enlarged.



(a) (b) (c)

Figure 4.6: Approximation of splat kernel in example view construction.

(a) The elliptical splat kernels and their circular approximation

(b) An example view construction using elliptical splatting kernels

(c) Same view construction as (b) with circular splatting kernels

Furthermore, instead of using a semi-transparent Gaussian filtering for the splat kernel, opaque kernels could be used to avoid the multi-pass process for correct blending. The final view-reconstruction is expected to be significantly faster.

4.4.3 Splat a Point Model onto Screen

In this thesis, a point model is incrementally built as linked point patches with a level-limited octree-based spatial partition hierarchy. To splat such a point model onto screen under current viewing condition, the first step is to perform culling. The octree structure of the point geometry is traversed recursively for an efficient view culling. Only those point clusters which passed the culling stage will be further processed for the final view construction. For the following subsections, GL_POINTS with solid disk kernel is assumed in the splatting algorithms.

4.4.3.1 LOD Control for a Multi-resolution Point Packing

LOD control can be applied in multi-resolution packing. For a selected point cluster, each of its point patches has multiple resolution levels and a particular resolution level is chosen to be splatted during view construction. LOD control provides anti-aliasing and higher rendering speed.

The level of detail selection of a point patch is done by pixel range comparison. As introduced in section 4.2.2, the term ***pixel range*** refers to the projected pixel coverage of a point cluster's bounding box under a certain viewing condition. For each chosen point cluster, the ***desired pixel range*** is computed based on its bounding box position and the current viewpoint and viewing matrices. The point patch resolution level which has the closest pixel range to the desired pixel range is chosen. After that, a proper splat size

is calculated to splat the chosen point samples onto screen. Equation (4.8) shows how to calculate the splat-size:

$$\textbf{Splat size} = \frac{\textit{scale factor} * \textbf{point patch pixel range}}{\textbf{desired pixel range}} \qquad (4.8)$$

Here the *scale factor* is used to ensure that a 2D view reconstruction by point splatting is as seamless as possible.

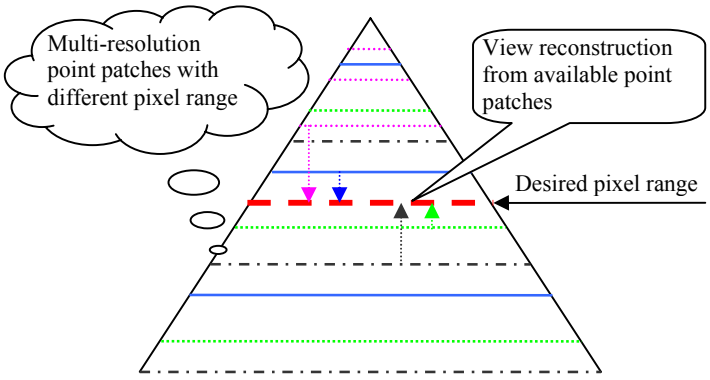


Figure 4.7: View reconstruction from multi-resolution point patches for one point cluster. Each colored line in the pyramid indicates a point patch with a certain resolution. Line length indicates the pixel range value. Point patches are signified by different colors.

Figure 4.7 illustrates the view reconstruction algorithm for a point cluster which contains four point patches. Each point patch has three resolution levels, which are signified by their pixel ranges. The whole structure is illustrated by a pyramid filled with

colored lines, where each line indicates a point patch resolution level and all lines with the same color are from the same point patch. The length of each line indicates the pixel range value. The bold dashed line simulates the current view-reconstruction with a desired pixel range. The colored arrow indicates that, from each point patch, the resolution level which has the closest pixel range to the desired pixel range value is chosen and then all the point samples from the chosen level of detail are either minified or magnified to form a new view reconstruction.

Furthermore, for an interactive VR, the viewpoint's moving velocity can also be taken into consideration in the LOD control algorithm. In this case, the desired pixel range in equation (4.8) will be divided by the moving velocity first for the splat size calculation. The viewpoint's moving velocity is computed at every view construction frame as the head tracking position change divided by the time difference.

4.3.3.2 Splat a Redundancy Eliminated Point Packing

If redundancy elimination is applied during the packing process, then the point patches should have non-overlapping geometry coverage. For a selected point cluster, all of its point patches should be splatted for the final view reconstruction.

In Section 3.2.2.1, the drawback of the simplified redundancy elimination algorithm was discussed. For the linked point patches, it is possible that some points

with lower sample rates in earlier patches are not deleted while some points with higher sample rates in more recent patches are deleted instead. Using the splat size calculated by equation (4.8), sometimes the new view reconstruction will be blurred even if there are high sample rate data available. This is because low sample rate points may have depth values smaller than or equal to high sample rate points, so the high sample rate data will actually be blocked by the depth test stage of graphics processing. To remedy this problem, the splat size determination algorithm is revised as follows:

```
Let sp_size = splat size of the point patch with closest pixel
range to the desired pixel range value;

for each chosen point patch
{
    compute its splat size cu_size;
    if(cu_size > ratio * sp_size)
    {
      cu_size = sp_size + ε * cu_size;
    }
}
```

Here *ratio* > 1, and 0< ε <1. This algorithm means that when a low sample rate point patch may block out the higher sample rate data, its splat size should be set to a smaller value.

4.4.3.3 Splat a Point Packing with Geometry Redundancy

Point patches from sampling frames can be packed without performing redundancy elimination. In this case, point patches of a point cluster may introduce overlapping geometry coverage. Overlapping patches with different sample rates are

saved inside the packing so that a specific point patch with the best matching sample rate can be selected for an output screen-space re-sampling. Picking one point patch with the best matching sample rate for current view re-sampling can improve the viewing quality but can also introduce more artifacts such as gaps and holes.

From a point patch's center position and average normal vector attributes and its sampling matrix, a proxy surfel representation can be computed. Using the re-sampling contour calculation algorithm introduced in section 4.4.1, an output screen-space re-sampling kernel contour for the proxy surfel can be calculated. Assume the computed output re-sampling kernel contour for a point patch is an ellipse with long axis radius r_0 and short axis radius r_1, the splat quality is computed by equation (4.1). The point patch which has a maximum q value will be selected for final point splatting.

4.5 Volumetric Splatting

Volumetric datasets represent a 3D sampling of the interior structure of objects, including amorphous and semi-transparent features, over a uniform/non-uniform 3D grid. A volume is a compact point-based embodiment of an object with each voxel as a point sample. Volume splatting is a re-sampling process from voxel sampling space to output screen space.

4.5.1 Splatting in Shear-warping Context

The shear-warp [Lacroute94] technique is one of the fastest software based volume rendering algorithms. The basic idea of shear-warp is a factorization of the viewing matrix into a 3D shear parallel to the slices of the volume data to form a distorted intermediate image, and a 2D warp to produce the final image. Applying the shear transformation to the volume means transforming each volume slice such that all viewing rays are parallel to the principal viewing axis. The coordinate system defined by this property is called *sheared-object space*. For parallel projections, this means a translation of every volume slice. For perspective projections, each slice has to be scaled as well.

The shear-warp factorization in volume rendering has four stages: permutation, shearing, compositing and warping. The permutation stage changes the storage order of voxels in memory in order to maximize cache coherency. During the shearing stage the volume is treated as a set of slices which are re-sampled on a sheared grid. Figure 4.8 show how a volume is transformed to *sheared-object space* for perspective projection by translating and scaling each slice. The quality of this re-sampling process depends on the filter used for the reconstruction. In the compositing stage all the slices are composted into an *intermediate image*. The final stage performs a 2D warping to form a correct projection consistent with the current view matrix.

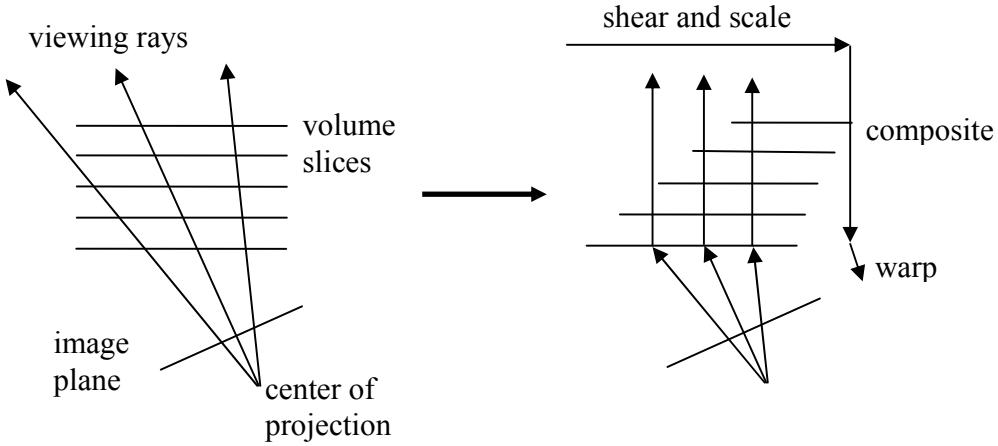


Figure 4.8: Shear operation of a volume in perspective transformation

A matrix representation of shear-warp factorization of the perspective viewing transformation is briefly reviewed below. For more details, see [Lacroute94] and [Schulze03]. Here, M_{view} is the original view matrix, M_{warp} is the warp matrix, M_s is the shear matrix, and M_p is the permutation matrix.

$$M_{view} = M_{warp} * M_s * M_p \tag{4.9}$$

Assume the object space eye position is e^o, and then the permuted object space eye position is:

$$e^p = M_p * e^o \tag{4.10}$$

Let's define:

$$s_x = -e_x^p / e_z^p, \quad s_y = -e_y^p / e_z^p, \quad s_w = -e_w^p / e_z^p \tag{4.11}$$

The shear matrix is computed as:

$$M_s = \begin{bmatrix} 1 & 0 & s_x & 0 \\ 0 & 1 & s_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & s_w & 1 \end{bmatrix} \tag{4.12}$$

It's clear that the shear matrix is only determined by the permuted object space eye position. To transform a particular slice z_0 of voxel data from object space to sheared object space the slice must be translated by $(z_o s_x, z_o s_y)$ and then scaled uniformly by $1/(1+z_o s_w)$. In the local object space of the volume, the slice plane with z_0=0 is the compositing plane where the intermediate image is generated and treated as a textured polygon in the final warping stage.

In the software-based implementation described in [Lacroute94], the shear operation is performed by traversing the volume in a scanline-based scheme and resampling and compositing the volume slices into an intermediate image. Using

splatting in the shear-warp context is straightforward [Cai00] and provides hardware-accelerated and high-quality re-sampling of the volume slices into *sheared-object space*. The splatting-based shear operation represents each voxel's contribution in the intermediate image buffer as a sheared footprint. The factorization of the transformation matrix makes the sheared footprint's shape and size remain the same for one slice of data for an arbitrary perspective viewing transformation. The size of a sheared footprint depends on the size of its reconstruction filter and the shear transformation matrix. The splatting-based shear algorithm is more efficient compared to the original software-based shearing algorithm, and has a higher re-sampling quality. After the shear operation, the intermediate image is warped by texture mapping hardware to form the final view.

4.5.2 Parallel Shear Warping

In shear-warp factorization, the transformation to sheared-object space is independent of the perspective projection plane. This makes the shear operations amenable to parallel processing even if the final view projection actually has a non-planar configuration, such as the view configuration for a cylindrical tiled display. The parallel volume shear-warping algorithm presented in this thesis involves two stages. In the first stage, all of the processing nodes hold an object-space data partition of the original volume and perform parallel shearing to produce a set of intermediate slices. In the second stage, the intermediate slices are distributed over the processing nodes by an

image-space data partition scheme, and each node performs parallel shear-warping to reconstruct the final view for its own display configuration.

An object-space data partition scheme prevents constant view-dependent data re-transmission. A slab-based object-space data partition is used in this thesis, where each processing node holds a slab of the original volume. After setting up a local volume coordinate system, each node computes a shear matrix based on equation (4.12) and then shears and composites the volume slices into a distorted intermediate image.

The intermediate volume is re-scattered among the processing nodes by an image-space data partition scheme. No further data exchange is needed during the view construction. Similarly, each node computes the shear matrix, shears and composites its own sub-intermediate-volume into another distorted image, and then warps this distorted image into the final view.

Figure 4.9 illustrates the two stages of the parallel volume shear-warping algorithm.

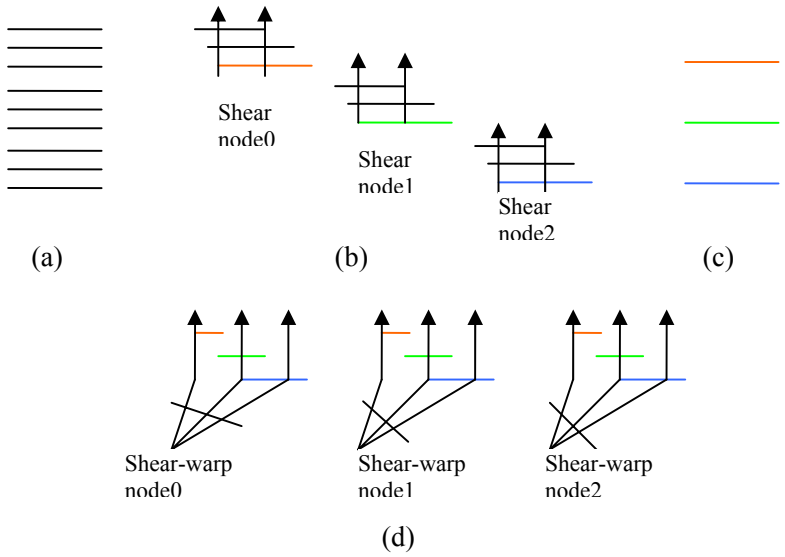


Figure 4.9: The parallel shear-warping algorithm

(a) The original volume slices; (b) Parallel shear, with colored compositing plane; (c) The resulting intermediate volume; (d) Parallel shear-warping on intermediate volume.

4.6 Point-based Visualization in Distributed Pipeline

Because of the conceptual simplicity of point based graphics, point based visualization functionalities can be easily discomposed into several stages over the distributed remote computation pipeline presented in this thesis. Balanced computation and communication over the pipeline is the basic criteria for the functionality distribution.

Typical point based visualization for surface datasets can break into three stages: point sampling, point packing and point splatting. Similarly, for volumetric datasets, the

shear-warping algorithm can be straightforwardly divided into a shear stage and a warp stage. These sub-functions can be assigned to the computation server and the viewing client in a balanced way for a specific visualization task.

4.7 Summary

The conceptual simplicity and rendering performance of points make it a good choice as a modeling and display primitive for efficient VR-end geometry caching and view reconstruction. Sampling, packing and rendering algorithms are discussed in this thesis to transform the original dataset into point samples, cache the point geometry, and reconstruct seamless 2D viewing from the point geometry. Different implementations of these algorithms with different levels of computational complexity are studied and made available to fit various visualization requirements for specific VR applications.

CHAPTER 5 SCALABLE PIPELINE DESIGN: COMPUTING AND COMMUNICATION

5.1 Chapter Organization

At the implementation level, each subsystem in the remote visualization framework represents a SIMD parallel rendering program in the distributed MIMD parallel pipeline. Each subsystem should have a scalable computing configuration. From a functionality point of view, there is a communication module between two adjacent computing subsystems through which data flows. Each subsystem employs a specific scalable computing algorithm to implement its functionalities. The inter-subsystem communication algorithm is adaptive to the scalabilities of the connected computing subsystems. It's important to balance the computation complexity of the inter-subsystem communication algorithm along with the scalability of its connected subsystems.

Section 4.2 and its subsections elaborate the parallel computing algorithms for each subsystem, including data partition, communication, and synchronization. Section 4.3 and its subsections describe the inter-subsystem communication algorithm which guides the data flow through the pipeline. Section 4.4 discusses the system configuration optimization that balances the computing and communication complexity of the pipeline. Section 4.5 summarizes this chapter by restating the system scalability and computational complexity for the remote VR visualization framework.

5.2 Scalable Subsystem Computing

In this distributed pipeline, each subsystem achieves parallelism by cluster computing and data partitioning. Local MPI (Message Passing Interface) communicator is established at startup time for intra-subsystem message passing and synchronization. A scalable computing algorithm includes local parallel computing, intra-system communication, and synchronization.

5.2.1 Data Server

The original dataset is distributed among the data server nodes. The computation server sends a data request to the data server indicating its required data partition scheme. Each data server node decides how to divide its own data storage and distribute different parts of data to different computation server nodes. Local computations are done in parallel for each node. In situations where information about global data storage is also needed for a final decision, intra-system communications are necessary. Data server nodes communicate with each other using a local MPI communicator.

5.2.2 Computation Server

Usually an evenly-distributed object space data partition is desired at the computation server. All server nodes perform parallel computation on their own data. Computing functionalities for each computation server node may include but are not limited to:

- Synchronize with other nodes to read the same viewing request from the client.
- Generate the intermediate data for the request viewing condition on its own data.
- Apply redundancy elimination and/or data packing if necessary. Intra-system communication may be involved, depending on the algorithm used to implement the function.
- Divide the intermediate data by culling for each client node's view frustum.
- Call the inter-system communication module to scatter data to the client.

5.2.3 Visualization Client

An image-space data partition scheme is applied at the client-end and all client nodes perform parallel view construction. Computing functionalities for each client node may include but are not limited to:

- Synchronize with other nodes to read from consistent data buffers for the same requested view.
- Apply multi-resolution data packing if necessary. Intra-system communication may be involved, depending on the algorithm used to implement the function.
- Reconstruct stereo view from the intermediate data for its own display configuration.
- Post-processing to construct stereo/auto-stereo effects for VR.

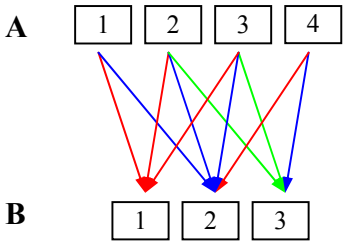
5.3 Inter-subsystem Communication

From a functionality point of view, each computing subsystem is attached to a communication module to exchange data with adjacent subsystems. Inter-subsystem communication is concurrent with subsystem data computing. For two scalable subsystems which communicate with each other, there is an all-to-all TCP connection among the computing nodes of those two subsystems. Theoretically all data sending nodes can send data in parallel to arbitrary data receiving nodes. But since all data that share the same network data link will eventually be transferred and processed in a sequential order, it's desirable that the order of data communication between the two subsystem nodes be arranged to make the best overall throughput out of the available network bandwidth between them. For example, a typical data communication between subsystem **A** and subsystem **B** includes a request message from **B** to **A** and data flow from **A** to **B**. According to the data request, each data sending node of **A** will compute its **data contribution** to each data receiving node of **B**. Then the root node of **A** will gather all the data contribution information and create a **data transfer assignment sheet**. A data transfer assignment sheet indicates the arrangement of the order of data communication from **A** to **B**.

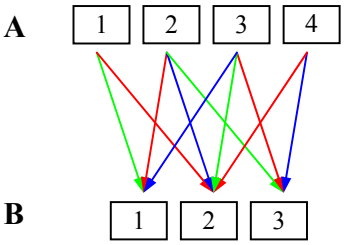
Figure 5.1 shows example arrangements of the order of data communication from system **A** to **B**, illustrated by the data transfer layout graph and the assignment sheet. In the example, system **A** has 4 nodes which can send data concurrently; system **B** has 3

nodes which can also receive data concurrently. Data links are available among all the nodes of **A** and **B**. Node A1 needs to send data to node B1 and B2; node A2 needs to send data to node B1, B2, and B3; node A3 needs to send data to node B1, B2, and B3; node A4 needs to send data to node B2 and B3. In the data communication arrangement illustrated by the layout graph (a) and its corresponding assignment sheet (c), the data transfer happens in three rounds. At the first round, data transfer relations are: A1 to B1, A2 to B1, A3 to B1, and A4 to B1. All the first round data transfers are indicated by directed red lines in graph (a) and the first and second rows in table (c). Similarly, data transfer relations at the second round are: A1 to B2, A2 to B2, A3 to B2, and A4 to B3. All the second round data transfers are indicated by directed blue lines in graph (a) and the first and third rows in table (c). Finally, data transfer relations at the last round are A2 to B3 and A3 to B3. All the last round data transfers are indicated by directed orange lines in graph (a) and the first and last rows in table (c). Another data communication arrangement is illustrated by layout graph (b) and its corresponding assignment sheet (d), where the data transfer also happens in three rounds. At the first round, data transfer relations are: A1 to B2, A2 to B1, A3 to B3, and A4 to B2. At the second round, data transfer relations are: A2 to B2, A3 to B1, and A4 to B3. At the last round, data transfer relations are: A1 to B1, A2 to B3, and A3 to B2. In the first data transfer arrangement scheme, the order of data communication causes a lot of data link conflicts because multiple nodes in **A** will try to send data to the same node in **B** at the same time. On the other hand, the second arrangement scheme minimizes the data link conflicts by

staggering the data links for each round. It is clear that the data transfer represented by the second arrangement scheme is more efficient by making better use of the available networking capacity. To optimize the data transfer assignment, the data receiving node of system **B** should be as diverse as possible for each round of data transfer.



(a) Data transfer layout 1



(b) Data transfer layout 2

A	1	2	3	4
B(round1)	1	1	1	2
B(round2)	2	2	2	3
B(round3)		3	3	

(c) Data transfer assignment sheet 1

A	1	2	3	4
B(round1)	2	1	3	2
B(round2)		2	1	3
B(round3)	1	3	2	

(d) Data transfer assignment sheet 2

Figure 5.1: Example data communication arrangements between subsystem A and B

Before the actual data transfer, the data transfer assignment sheet is sent from the root node of **A** to the root node of **B** and broadcast inside **B** so every node will know how many data are expected at each of their data links and in which order. After each node of

B receives its expected data from **A**, a synchronization call will be issued and further processing based on the received data will be executed after the call returns.

5.4 Pipeline Configuration Optimization

Proper granularity is desirable for a best overall performance of a parallel computing system. In the distributed pipeline presented in this thesis, the inter-subsystem communication algorithm is adaptive to the scalabilities of the connected computing subsystems. Generally the communication complexity grows by $O(N^2)$ when the connected subsystem's process number scale at O(N). It's important to balance the inter-subsystem communication complexity along with the scalability of its connected subsystems.

The pipeline configuration optimization is visualization task-specific. Given a specific dataset, there exists a tradeoff between the server computation performance and the network transfer latency. Using more server nodes results in faster computation, but it also increases the data generating rate, thus introducing longer data transferring latency. There are two ways to alleviate the non-linear growth of communication complexity along with the linear growth of the subsystem configuration. One way is to design the server-end scalable computing algorithm so that the growth of data generating rate is slower than the growth of the number of processes. The other way is to limit the number

of overall data connections by avoiding the situation where every server node need to connect to every client node in each data flow cycle.

Usually different server setups will be tested and a best configuration will be chosen for the most effective client visualization.

5.5 Summary

Scalability makes a visualization task adaptive to the available computing and visualizing resource configurations. Each subsystem of the distributed system can perform either cluster-based parallel computing or single workstation-based sequential computing. Synchronized intra-subsystem computing and inter-subsystem communication ensure consistent pipeline computing. The pipeline configuration can be optimized based on a balanced granularity as the ratio of computation to communication.

CHAPTER 6 SUBSYSTEM COUPLING SCHEMES

6.1 Chapter Organization

The term ***coupling*** refers to the connection between the subsystems inside a distributed pipeline. The degree of coupling defines the level at which a subsystem can work independently without waiting for data from another subsystem. In a remote visualization system, the coupling between the computation server and the visualization client usually means different inter-system communication rates, and may introduce different system characteristics. In this thesis, Virtual Reality interaction is more graphics-intensive and frame-rate critical than a normal graphics rendering task, so different client-server coupling schemes could be applied to meet the VR visualization requirements.

Section 6.2 introduces the performance metrics used in this thesis to measure the effectiveness of the remote visualization system. Section 6.3, 6.4 and 6.5 discuss several subsystem coupling schemes and their implementation details. Section 6.6 summaries this chapter by restating the usage of various coupling schemes for VR applications.

6.2 System Performance Metrics

To measure the performance of the distributed visualization system, besides the image quality, the following performance metrics are introduced, similar to the performance metrics described in [Ma00].

- *Data generating rate* is the actual bit rate the server generates data at given a specific server-client configuration.

- *Data transfer rate* is the actual data transfer throughput during the run time.

- *Data consuming rate* is the actual bit rate the client consumes data at given a specific server-client configuration.

- *Start-up latency* is the time until the client receives data after it sends a steering request to the server.

- *View update latency* is the time until the client gets the correct view construction for a requested tracker position.

- *View construction time* is the time the client needs to reconstruct a view from the data received from the server.

- *Inter-frame delay* is the average time between the consecutive view reconstructions, or the reciprocal of the frame rate.

For a specific visualization task, the computation server and visualization client would normally generate and consume data in a stable rate if executed independently. By working concurrently with the network communicating module in the pipeline, the data generating rate, the data consuming rate, and the data transfer rate should be balanced during run time.

In a head-tracked VR visualization environment where interactive viewing is desired, image quality, coherency and frame rate are the key metrics to indicate the quality of the VR exploration. The dataset characteristics, subsystem computing and inter-system communication performance, network conditions, and the subsystem coupling scheme all play crucial roles to determine the effectiveness of the system.

6.3 Synchronous Coupling

Synchronous coupling of a computation server and a visualization client means that new frame of data can't flow through the pipeline if the old data is not consumed yet. Synchronous coupling introduces a large amount of inter-system waiting between the server and the client, thus aggravating the inter-frame delay.

Figure 6.1 shows an example timeline diagram indicating how the server computation and the client view construction work together in a synchronized coupling mode. Performance metrics such as the latencies and the view construction time and the inter-frame delay are indicated in the figure.

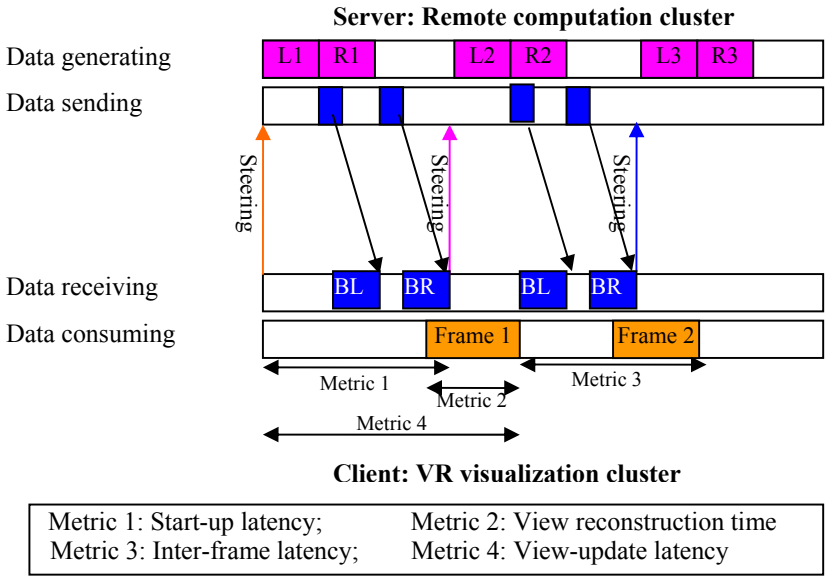


Figure 6.1: Workflow timeline in a synchronized client-server coupling mode.

6.4 Loose Coupling by Buffering Algorithm

Loose coupling of a computation server and a visualization client means that adjacent work flow cycles can interleave with each other by introducing a circular buffering algorithm at the client side. Data coming from different server computation frames can be stored in different buffers, thus avoiding unnecessary inter-waiting between the server and client. The number of buffers determines the looseness of the

coupling. Loose coupling needs more data storage space at the client side, but also improves the view updating frame rate.

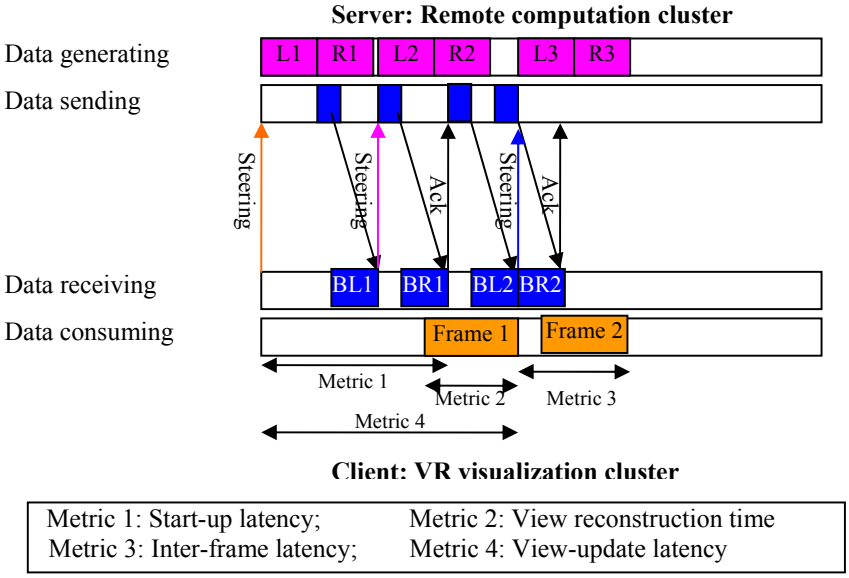


Figure 6.2: Workflow timeline in a loose client-server coupling mode.

Figure 6.2 shows an example timeline diagram indicating how the server computation and the client view construction work together in a loose coupling mode. Performance metrics such as the latencies and the view construction time and the inter-frame delay are indicated in the figure.

The order of data transfer needs to be maintained carefully in a loose coupling scheme. Even though new data may be generated concurrently with the current data

transfer, the new data will not be transferred before receiving an acknowledgment message indicating the completion of the current data transfer.

6.5 Asynchronous Coupling

In this mode, client view construction is isolated from the normal server computation – client visualization work flow cycle. Asynchronous coupling is enabled by data caching and packing at the client side. A client continuously reconstructs an intermediate view for arbitrary viewing conditions using the intermediate geometry constructed during runtime and cached in its local memory. Intermediate views may introduce visual artifacts such as holes and gaps and fuzziness due to insufficient geometry or resolution for current viewing condition. These visual artifacts are expected to decrease as new data comes in from the server, especially when smaller tracker movement is expected when users want to examine a particular area of interest. Asynchronous coupling needs a large amount of data storage space at the client side and introduces visual artifacts, but it can improve the interactive frame rate as much as possible by constructing intermediate views for the current tracker position without waiting for a corresponding data update from the server.

Figure 6.3 shows an example timeline diagram indicating how the server computation and the client view construction work together in an asynchronous coupling mode. In the example, the server end point sampling is much slower than the client end

view construction, and the client end intermediate visualization doesn't wait for stereo sampling of the original dataset from the server. Performance metrics such as the latencies and the view construction time and the inter-frame delay are indicated in the figure.

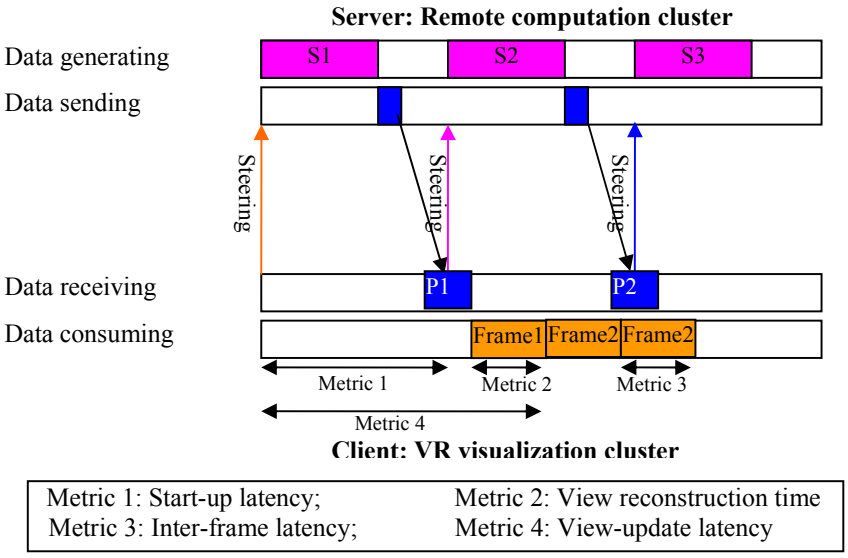


Figure 6.3: Workflow timeline in an asynchronous client-server coupling mode.

6.6 Summary

Several subsystems coupling schemes and their implementation details are discussed in this chapter. The degree of coupling between the VR client and the computation server indicates the inter-waiting time inside the view construction cycle, but with possible viewing artifacts due to delayed view updating. A proper coupling scheme can be selected to fit specific VR application requirements.

CHAPTER 7 EXPERIMENTS AND RESULTS

7.1 Chapter Organization

In sections 7.2, 7.3 and 7.4, case studies using different type of datasets under the proposed visualization framework are discussed in detail. Experimental results are given for each case study. Section 7.5 summarizes this chapter by restating the system adaptability over different dataset characteristics.

7.2 Case study for Mesh Dataset

Several mesh datasets are experimented using the remote visualization framework presented in this thesis. The system configuration is in a one-one mode, which means there are one remote computer as the computation server and one local computer as the visualization client. For the experiments, the server computer is a Linux machine with an Intel Xeon 1.8 GHz CPU, Quadro4 900 XGL graphics card and 2 GB main memory. The client computer is also a Linux machine with dual 64 bit AMD Opteron 246 2 GHz processors, Quadro FX 4400 graphics card and 4 GB main memory. The client computer drives a Varrier^{TM} autostereo VR display.

To achieve the autostereo effect, the visualization scene needs to be drawn twice (once for each eye) and interleaved together in every frame [Sandin05]. When the size of

a mesh dataset reaches to several million triangles, visualizing it on a single auto-stereo VR workstation can only achieve roughly 1-2fps. This is too slow for smooth interaction in head-tracked VR. To solve the frame rate problem, instead of local rendering, the visualization is carried out through the remote computation-local view construction pipeline. Figure 7.1 shows the customized remote visualization pipeline for the mesh dataset in one-one mode. The pipeline includes a remote computation server which also serves as the data server, and a VR desktop (client). The server and client are connected through high-speed network. Both server and client processes are multi-threaded so that the functional modules, such as data communication, server-end point sampling and client-end point splatting can run concurrently, taking advantage of multi-processors if applicable. The server sampling is at 1280*800 resolution, and the client visualization is at 2560*1600 resolution. The client and server resolutions are independent; they do not need to be integral multiples of each other nor have the same aspect ratio.

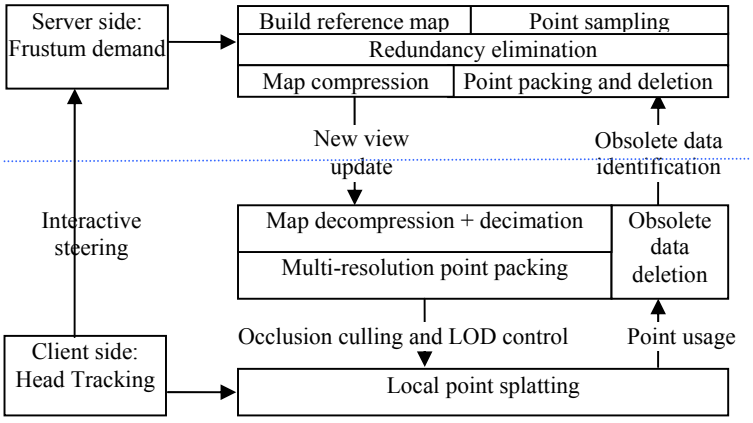


Figure 7.1: Pipeline diagram for case study on mesh dataset

Given a frustum request, server side computation includes sampling the visible part of the original mesh into a 2D sampling map (depth-image) of proper resolution and eliminating the redundant sampling before transmitting the sampling map to client. For each new view updating frame, the client will decimate the received sampling map, produce multi-resolution 3D points from the decimated maps and pack them into the level-limited octree-based geometry cache. The server also keeps a compact packing of the previous samplings for redundancy elimination's purpose. To maintain a compact point model in both server and client's main memory, obsolete points will be deleted regularly. Obsolete point deletion is originated by client and needs to be synchronized between server and client to ensure correct redundancy elimination.

Networking between the server and client provides data communication, such as view-frustum demanding, map transmitting and obsolete data notification. After redundancy elimination, the updated map (depth-image) is compressed before transmission. A lossless LZW compression [Nelson89] is used for map compression. For a depth-image with resolution of 1280*800, the original data size is about 7MB; compression reduces the size to about 2MB. The compressed maps are sent to the client using the reliable TCP/IP protocol. Because the server sampling rate is expected to be low, e.g. one frame per second, the network traffic is small and bursty.

At every view construction frame, the client splats visible 3D points which are stored in the geometry cache into seamless 2D views. LOD control is applied when splatting the multi-resolution point model onto the screen. The server and client work together in an asynchronous coupling mode, which means that the view updating and view construction of the client don't need to synchronize with each other at the frame level. The view updating frame rate is limited by the server computation and the client packing performance; the view construction frame rate is determined by the client's data retrieval from the cached geometry and splatting performance.

View reconstruction by splatting locally cached 3D points with constant head movement will introduce holes, gaps, and dis-occlusions because of data incompleteness, and splatting artifacts because of inconsistent data resolution. By replenishing new points every view updating frame, a better view will be constructed in exchange for a short waiting period. The interactive VR visualization experience is expected to be quite smooth if the waiting time for the new view updating is no more than 2-3 seconds since the user can still move their head and see the existing splats at about 15fps.

Figure 7.2 shows the 2D view reconstruction by point splatting for the Crater Lake dataset before and after a new view update from remote server. In image (a), the view reconstruction contains holes, gaps and fat splats, but they will disappear in image (b) after new view update without any noticeable artifacts.

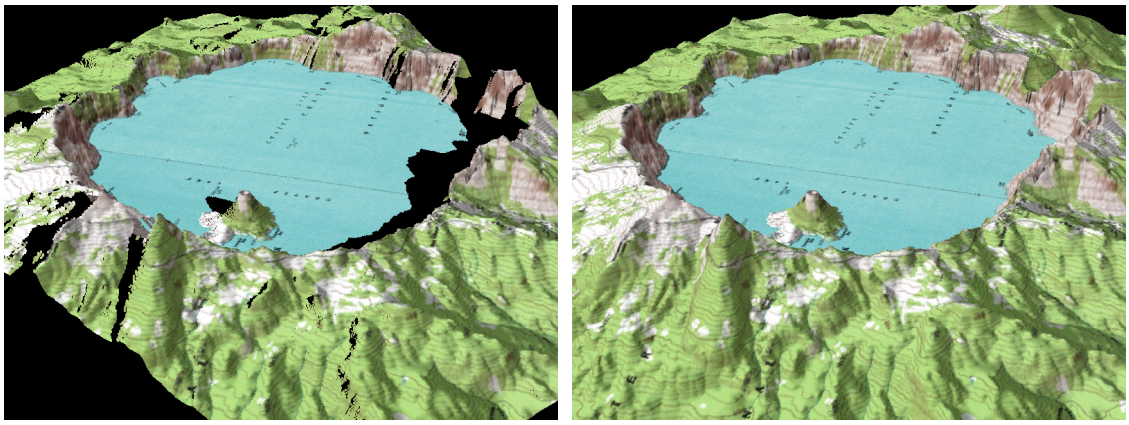


Figure 7.2: View reconstructions by point splatting for the Crater Lake

Left: View reconstruction by splatting available points;

Right: View reconstruction after new view update

Besides the Crater Lake dataset, some classic medical datasets, such as the bone, skin, head, and skull, are also experimented. Table 7.1 lists the description of the experimental mesh datasets and the resulting performance under the remote visualization pipeline. All datasets are non-transparent surface triangle meshes.

Table 7.1: Experimental mesh dataset description and visualization performance

Dataset Name	Dataset Size (triangles)	Frame rate without remote visualization	Frame rate with remote visualization
Crater lake	5M	1 fps	17 fps
Bone, skin, head together	1.2 M	4 fps	18 fps
Skull	0.45M	12fps	20fps

The overall performance of the client-end view reconstruction is more than 15 fps, independent of the original dataset complexity. And the average server-end view-udpating rate is about 0.5 fps. See figure 7.3 for the point-based visualization results of experimented datasets.

(a) Crater Lake (b)Bone (c)Skin

(d)Skull (e)Head

Figure 7.3: Point-based view construction of experimental mesh datasets.

(a) The Crater Lake dataset is courtesy of USGS

(b)(c) The bone and skin datasets are courtesy of the Visible Human Project, 2003

(d)(e) The skull and head datasets are coutesy of the VR Medical Laboratory, University of Illinois at Chicago

7.3 Case Study for Julia Sets

The French mathematician Gaston Julia invented the Julia set in 1918. There was a renewed interest in this creation in the 80s when computers made possible the visualization of these fractal forms. Julia's idea was to observe the behavior of the orbit of a complex number under iteration of a function f. That is, begin with a complex number z_0, visualized as a point in the plane, and apply f to z_0. The resulting value is fed back into the function f to obtain a new complex number z_1. This in turn is fed back into f to obtain z_2, and so on. The resulting sequence of complex numbers $\{z_0, z_1, z_2, ...\}$ is called the *orbit* of z_0 under f. We will refer to a complex number z_0 as a *prisoner* if its orbit under f is bounded, and an *escapee* if the orbit is unbounded -- that is, terms in the orbit get arbitrarily far away from 0. The set of all prisoners for a given function f is called its *prisoner set*, and the set of all escapees is called the *escape set*. The *prisoner set* of function f is also called the *filled-in Julia set*. The Julia set for f is defined to be the boundary between the prisoner set and the escape set of the function *f*. One very popular function is the quadratic: $f(z) = z^2 + c$. Julia sets of quadratic functions, along with other deterministic fractals, exist in higher dimensional hypercomplex spaces. A hypercomplex number has multiple imaginary components. Our implementation uses quaternions which are four-dimensional extention of complex numbers.

The common visualization techniques of the quaternion Julia sets include bounding tracking method [Nortan82], inverse iteration algorithm [Holbrook83], ray-

tracing [Hart82], etc. Inverse iteration is the best method for quickly visualizing the global shape of the Julia set whereas ray tracing is the best method for investigating the finer details of the set. Boundary tracking also provides a global view that is significantly better than inverse iteration but at the expense of much more time and memory. John Holbrook computed the quaternion Julia sets by sampling every point in a 3-D grid and then rendered the resulting binary voxel array. John Hart [Hart90] used a quaternion square root function to adapt the classic inverse iteration algorithm to the quaternions. The augmented version produces a 3-D Julia set defined by a point cloud that can be interactively manipulated on a graphics workstation. Dan Sandin and Louis Kauffman have used iterative distance estimation techniques to *ray trace* 3D slices of the fractals, providing high quality 2D images of these sets at various levels of detail. Because of the long processing time to produce a high-quality image by ray-tracing, the animated Julia set visualization is achieved by pre-computing a huge number of high-quality images in advance and playing them back in high frame rate like a movie. Recently an auto-stereo playback version of their Julia set animation has been implemented in the Personal Varrier^{TM} system, at Electronic Visualization Laboratory, University of Illinois at Chicago. While providing extremely smooth visualization experience of the Julia set animation, the shortcoming of this movie-style visualization is the fixed navigation path and lack of human interaction.

As an experiment, the remote visualization framework presented in this thesis is customized to visualize a high quality Julia set animation in VR, where free navigation and fast view reconstruction of at least 15 frames per second are required. The goal is to help scientists examine the 3D slice of the quaternion Julia sets by great detail in a stereo virtual reality mode, and be able to study the animation by varying Julia set's parameters. Ray-tracing based Julia set generation technique is used here to get high-detailed images. The system configuration is in a many-one mode, which means there are a scalable computer cluster as the computation server and one local computer as the visualization client. In actual experiments, the client end of the Julia set visualization system is the Personal Varrier^{TM} autostereo display. The net resolution of the display is 2560*1600, while the sampling resolution is 1024*640 for each eye before the auto-stereo image interleaving. The Varrier runs on a Linux machine with dual 64 bit AMD Opteron 246 2 GHz processors, Quadro FX 4400 graphics card and 4 GB main memory. The experimental server is a Linux PC cluster with 32 nodes. Each node has dual Intel Xeon 1.8 GHz processors, Quadro4 900 XGL graphics card and 2 GB main memory.

Figure 7.4 is the diagram showing the customized functional framework of the point-based Julia set visualization system. Steered by the client-side interaction, the server produces view-dependent point sampling of the continuous Julia set surface by the perspective back-projection from a 2D grid input image space. The client caches point samples provided by the server sampling process with multiple sample rates and organize

them into an octree-based spatial partition structure. The client-end view reconstruction is another perspective projection to splat the cached point samples with a selected sample rate onto the output image space.

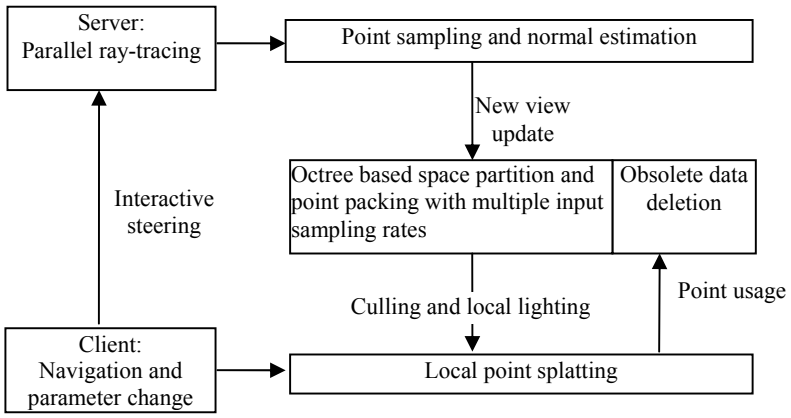


Figure 7.4: Pipeline diagram for case study on the Julia set

One of the key ideas of implementing the ray-tracing based real-time Julia set is to speed up the ray-tracer by parallel server processing. A load-balanced parallel ray-tracing algorithm is implemented in a one-row-per-node computation mode. At first, all server computation nodes get the view frustum request and each node begins to sample one row for the image. After a server node finishes sampling the current row for the final image, it sends the point samples back to a compositing node and the compositing node assigns another sample-row to it until the last row of the final image is sampled. This one-row-per-node computation scheme can achieve good load balancing and make the parallel processing very scalable. Table 7.2 shows the ray-tracing time of producing the same Julia image with different number of server computation nodes. The data shows that

the performance boost by using more computation nodes is nearly linear. The parallel ray-tracing running on a computer cluster of 40 nodes can be almost 40 times faster than the single-computer ray-tracing. Each point sample from the ray-tracing has attributes of color $\boldsymbol{c}$, spatial coordinate $\boldsymbol{p}$, and normal $\boldsymbol{n}$.

Table 7.2: The parallel ray-tracing performance

Cluster node number	1	2	4	8	16	32	40
Ray-tracing time (seconds)	69.3	34.8	17.3	8.7	4.36	2.19	1.76

The server and client work together in an asynchronous coupling mode. For each view updating frame, point patches from sampling frames are packed without performing redundancy elimination. Since the size of point packing grows fast without redundancy elimination, data deletion along the packing plays a more important role. Both access time and data sample rate can be the weighing factor for a data deletion decision.

The view reconstruction works by picking one point patch with the closest matching sample rate for current view re-sampling and splat it onto screen, as discussed in chapter 4, section 4.4.3.3. Splatting artifacts, such as holes, gaps and aliasing, can appear in a constructed view because of data incompleteness and/or simplified re-sampling process with circular splatting kernel. By replenishing new points every view updating frame, a better view will be constructed in exchange for a short waiting period.

The interactive VR visualization experience is expected to be quite smooth if the waiting time for the new view updating is no more than 2-3 seconds since the user can still move their head and see the existing splats at about 15fps. Figure 7.5 show the results of rotating the Julia set over y axis. Figure 7.6 shows some more experimental results when visualizing the Julia set animation. The cached point geometry for one animation step is completely deleted before visualizing the next animation step. A similar active data deletion mechanism can also be used in mesh dataset visualization when the old point model becomes large enough to slow down the client-end local view construction.

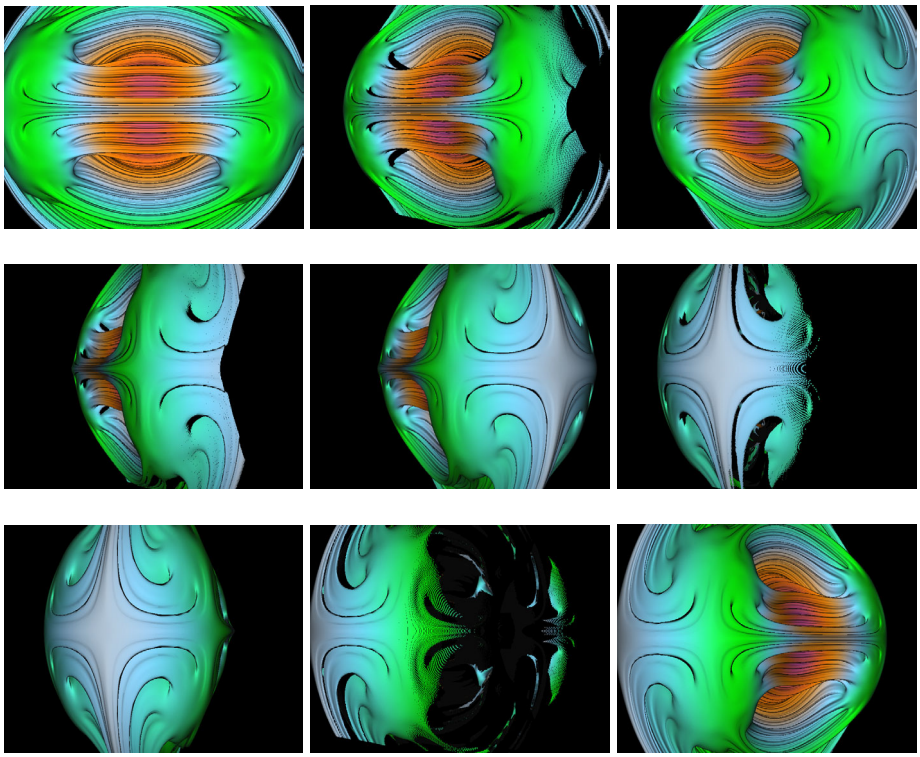


Figure 7.5: A series of intermediate Julia images by remote visualization.

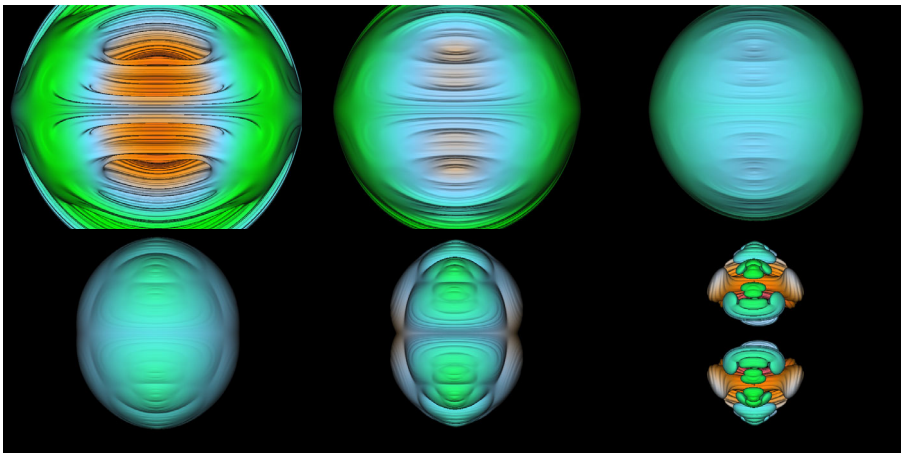


Figure 7.6: Julia animation by varying one of its parameters.

7.4 Case Study for Volumetric Datasets

Volumetric data sets are widely used in scientific and medical applications. They can arise both from scans of real-world phenomena (such as a CT or MRI scan of the brain) and from simulation (for example, fluid flow near an airplane engine intake). One of the key advantages of volumetric data is that, unlike surface-based representations, it can embody the interior structure of the objects, including amorphous and semi-transparent features. Additionally, operations such as cutting, slicing, or tearing, while challenging for surface-based models, can be performed relatively easily with a volumetric representation. While volume rendering is a very popular visualization technique, the lack of interactive frame rates has limited its widespread use. Volume rendering is very memory and computation intensive even for the newest graphics hardware.

As an experiment, the remote visualization framework presented in this thesis is customized for a real-time VR system to visualize volumetric datasets. The system configuration is in a many-many mode, which means there is a scalable tiled-display virtual reality environment connected to a scaleable parallel server computer cluster via a high-speed network. The distributed shear-splat-warping volume rendering algorithm is used for the remote visualization pipeline. In the actual system configuration, the visualization client is a cluster-driven cylindrical tile-display with Varrier^{TM} autostereo technology [Sandin05]. Every cluster node has a dual-head Nvidia Quadro FX3000 graphics card which drives two screens, each with 1600x1200 pixel resolution. In such a hardware configuration, the upper limit frame rate for an autostereo viewing is about 30fps. The computation server is also a PC cluster with the same graphic cards. The splat is rendered as graphics hardware accelerated point sprites with a Gaussian texture. The splatting performance for the graphics is about 60M splats per second. Figure 7.7 shows the customized functional framework for volume rendering.

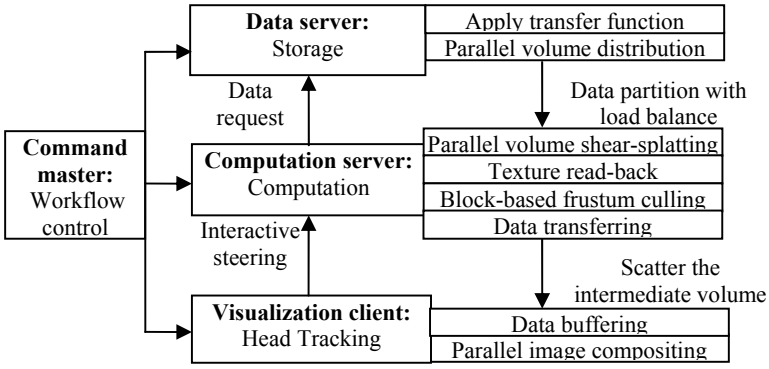


Figure 7.7: The distributed volume rendering pipeline

A slab-based object-space data partition is used at the server side, where each server node holds a slab of the original volume. An object-space data partition scheme prevents constant view-dependent data re-transmission. An even data load and back-to-front sequence is easily maintained for balanced server computation and straightforward compositing. Each server process computes a shear matrix in its local coordinate system and then splats and composites its own volume slab into one distorted intermediate image. Off-screen rendering is used for the intermediate image rendering. The intermediate image is usually set to be larger than the original slice size to remedy the scaling and translation caused by the shear operation. Dynamic intermediate image sizes can also be used during runtime to speed up the performance of the intermediate image read-back from the graphics card texture memory to main memory. Loose coupling scheme is performed in the volume rendering pipeline. Up to three set of left and right data buffers can be allocated at the VR client side. New data frame can be received and saved into another buffer while data are consumed from the current buffer. It's important to synchronize among all the client nodes before every draw frame to make sure all nodes get the same tracker position and read from consistent data buffers.

The output of the server computation is an intermediate volume, and the server scatters the resulting intermediate volume to the client by an image-space data partition scheme. No further data exchange among the client nodes is needed during the view construction. Similarly, each client computes the shear matrix, splats and composites its

own sub-intermediate-volume into another distorted image, and then warps this distorted image into the final view. Figure 7.8 shows an example of the intermediate images produced by the server computation. The experimental dataset is a 256^3 voxel foot, and 3 server computation nodes are used, thus 3 intermediate images are produced. For the same experiment, figure 7.9 shows the client-end reconstructed viewing for a 4-screen tiled display driven by 2 compositing nodes. There are display side bands between the 4 tiled screens.

Figure 7.8: The intermediate images produced by 3 server nodes for the foot dataset

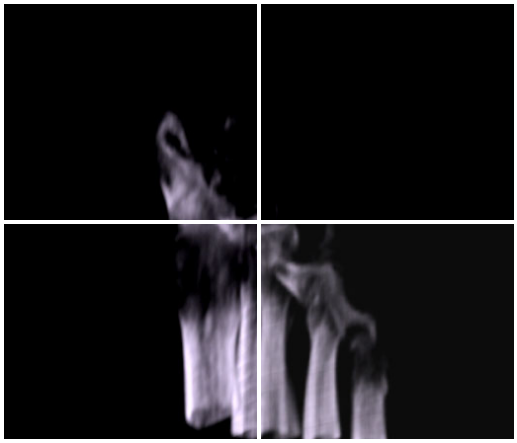


Figure 7.9: The final viewing in a 4-screen tiled display for the foot dataset

Different volumetric datasets are visualized under this remote computation architecture. Given a specific volume dataset, there exists a tradeoff between the server computation performance and the network transfer latency. Using more server nodes results in faster volume computation, but it also increases the data generating rate, thus introducing longer data transferring latency. Usually different server setups will be tested and a best configuration will be chosen for the most effective client visualization.

Table 7.3 shows the view construction frame rate with different system configurations for the Foot dataset. The dataset is in courtesy of Philips Research, Hamburg, Germany. The volume dimension is 256x256x256 in 8 bits sample precision. In the experiment, about 1k non-transparent voxels are processed during the actual view construction after applying a transfer function to the volume.

Table 7.3: View construction frame rate with different system configurations for the Foot dataset, volume size 256x256x256

	Server number	1	2	3	4
Client number	Frame rate (fps)				
1		27	25	21	18
2		27	25	20	17
3		27	24	20	17
4		26	23	18	15

Table 7.4 shows the view construction frame rate with different system configurations for the Christmas dataset. The dataset is in courtesy of the Department of Radiology, University of Vienna and the Institute of Computer Graphics and Algorithms, Vienna University of Technology. The volume dimension is 512x499x512 in 16 bits sample precision. In the experiment, about 7k non-transparent voxels are processed during the actual view construction after applying a transfer function to the volume.

Table 7.4: View construction frame rate with different system configurations for the Christmas tree dataset, volume size 512x499x512

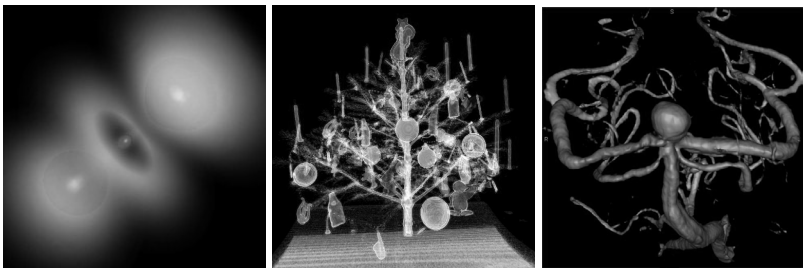
	Server number	1	2	3	4	5	6
Client number	Frame rate (fps)						
1		4.2	6.8	9.3	12.3	8.6	5.4
2		4.2	6.7	9.3	12.3	8.6	5.4
3		3.4	6.2	9.2	12.1	8.4	5.4
4		3.2	5.6	8.7	11.8	8.4	5.3

Beside the foot dataset as shown in figure 7.9, Figure 7.10 shows the final view construction of some other the experimental datasets visualized using this distributed volume rendering pipeline.

7.5 Summary

As case studies to prove the feasibility of the proposed visualization strategy, datasets with different characteristics, such as triangle meshes and volumes, are used as

customized visualization instances of the proposed framework. Several pipeline configurations, such as single server to single client, server cluster to single client, and server cluster to client cluster, are tested for different applications. Also, different point based algorithms and subsystem coupling schemes are selected in each case study and their functionalities can be merged together seamlessly for a specific application. All experiments show improved VR interaction.



(a) (b) (c)

Figure 7.10: The final view reconstruction of some experimental datasets

(a) 128x128x128 Hydrogen Atom dataset, in courtesy of SFB 382 of the German Research Council (DFG).

(b) 512x499x512 Christmas Tree dataset, in courtesy of the Department of Radiology, University of Vienna and the Institute of Computer Graphics and Algorithms, Vienna University of Technology.

(c) 512x512x512 Head Aneurism dataset, in courtesy of Philips Research, Hamburg, Germany

CHAPTER 8 CONCLUSIONS AND FUTURE WORK

8.1 Summary

State-of-the-art Virtual Reality technologies such as Varrier™ bring better interaction and comprehension into visualization experience. But VR applications are still limited in the area of large-scale scientific visualization mostly because of the intensive graphics computation for VR viewing.

The goal of this thesis is to design and implement a distributed visualization framework which combines VR technologies and remote computing resources through a high speed network, so that large-scale scientific datasets can be visualized in real-time on local VR devices.

The framework is designed to be a scalable distributed system with pipelined data retrieval, computation, and visualization for various datasets. Scalability makes the system adaptive to the available computing and visualizing resource configurations. Each subsystem of the distributed system can perform either cluster-based parallel computing or single workstation-based sequential computing. The pipeline configuration can be optimized based on a balanced granularity as the ratio of computation to communication.

The pipeline is an MIMD design which explores computing and networking parallelism along with the data flow.

Special implementation features of the pipeline are presented in this thesis based on the requirements of interactive VR exploration. First of all, point samples are introduced as an intermediate format of data which flow through the pipeline. The conceptual simplicity and rendering performance of points make them a good choice as modeling and display primitives for efficient VR-end geometry caching and view reconstruction. Sampling, packing, and rendering algorithms are discussed in this thesis to transform the original dataset into point samples, cache the point geometry, and reconstruct seamless 2D viewing from the point geometry. Different implementations of these algorithms with different levels of computational complexity are studied and customized to match the various visualization requirements for specific VR applications. The straightforward functional decomposition of point-based graphics enables flexible and balanced workload distribution through the computation pipeline. Secondly, different subsystem coupling schemes are discussed and can be selected to fit for different VR application requirements. Looser coupling of the VR client from the computation server means less waiting time inside the view construction cycle, but with possible viewing artifacts due to delayed view updating.

As case studies to prove the feasibility of the proposed visualization strategy, datasets with different characteristics, such as triangle meshes and volumes, are used as customized visualization instances of the proposed framework. Several pipeline configurations, such as single server to single client, server cluster to single client, and server cluster to client cluster, are tested for different applications. Also, different point based algorithms and subsystem coupling schemes are selected in each case study and their functionalities can be merged together seamlessly for a specific application. All experiments show that VR interaction can be improved for various visualization tasks by utilizing the visualization framework presented in this thesis.

8.2 Future Work

As a flexible and extendable visualization framework, the graphics-processing algorithms as well as pipeline configurations can always be enhanced based on the current implementation. Missing functionality can be added and out-of-date algorithms can be improved for new requirements of visualization tasks. Given the current implementation, the future work for this thesis may include:

- Adding a new algorithm to get correct point sampling for transparent mesh datasets.

- Improving existing splatting algorithms to make better use of GPU programming, especially for faster blending of splats with semi-transparent Gaussian filters, and per-pixel shading.

- Improving existing parallel processing algorithms, such as data partition, to balance the computational complexity of each scalable subsystem in the pipeline and the communication complexity between them.

- Adding new algorithms for animated dataset visualization.

BIBLIOGRAPHY

[Alexa01] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, and C. Silva. Point set surfaces. In *Proc. IEEE Visualization 2001*, pages 21-28, 2001.

[Blinn92] J. F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *Proceedings of SIGGRAPH 82*, pages 21–29, 1992.

[Botsch02] M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In *Proceedings Eurographics Workshop on Rendering*, pages 53–64, 2002.

[Botsch03] M. Botsch, L. Kobbelt. High-quality point-based rendering on modern GPUs. In *Proceedings Pacific Graphics2003*, pages 335–343. IEEE, Computer Society Press, 2003.

[Botsch04] M. Botsch, M. Spernat, L. Kobbelt, Phong Splatting, In *Proceedings of Point-based Graphics 2004*, pages 25-32, Symposium on Point-Based Graphics, 2004.

[CAVEwave] CAVEwave™ End-to-End 10 Gbps Wavelength Inaugurates National LambdaRail, www.evl.uic.edu

[Cabral94] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *Workshop on Volume Visualization*, pages 91–98. Washington, DC, October 1994.

[Cai00] W. Cai, G. Sakas, DRR volume rendering using splatting in shear-warp context, in *IEEE Nuclear Science Symposium Conference Record*, pages 19/12-19/17 vol.3, Lyon, France, 2000

[Carceroni01] R. Carceroni, K. Kutulakos. Multi-view scene capture by surfel sampling: From video streams to non-rigid 3D motion, shape & reflectance. In *Proceedings of t7th International Conference on Computer Vision,* pages 60–67. 2001.

[Chang99] C. Chang, G. Bishop, and A. Lastra. LDI tree: A hierarchical representation for image-based rendering. *Computer Graphics (SIGGRAPH'99)*, pages 291–298, ACM Press, August 1999.

[Chen93] S. Chen and L.Williams. View interpolation for image synthesis. *Computer Graphics (SIGGRAPH'93)*, pages 279–288, August 1993.
[Chen95] S. E. Chen. QuickTimeVR – an image-based approach to virtual environment navigation. *Computer Graphics (SIGGRAPH'95)*, pages 29–38, August 1995.

[Chen01] B. Chen and M. Nguyen. POP: A hybrid point and polygon rendering system for large data. In *Proceedings IEEE Visualization 2001*, pages 45–52, 2001.

[Coconu02] L. Coconu, H. Hege. Hardware-oriented point-based rendering of complex scenes. In *Proceedings Eurographics Workshop on Rendering*, pages 43–52, 2002.

[Cohen01] J. D. Cohen, D. G. Aliaga, W. Zhang. Hybrid simplification: Combining multi-resolution polygon and point rendering. In *Proceedings IEEE Visualization 2001*, pages 37–44, 2001.

[Correa02] W. T. Corrêa, S. Fleishman, C. T. Silva. Towards Point-Based Acquisition and Rendering of Large Real-World Environments. In *SIBGRAPI 2002*, pages 59-67, 2002

[Cruz-Neira92] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. The Cave Audio Visual Experience Automatic Virtual Environement. In *Proceedings of SIGGRAPH 92*, pages 64-72, 1992.

[Csuri79] C. Csuri, R. Hackathorn, R. Parent, W. E. Carlson, and M. Howard. Towards an interactive high visual complexity animation system. In *Proceedings of SIGGRAPH 79*, pages 289–299, 1979.

[Dachsbacher03] C. Dachsbacher, C. Vogelgsang, and M. Stamminger. Sequential point trees. In *Proceedings ACM SIGGRAPH 03*, pages 657–662. 2003.

[Dally96] W. Dally, L. McMillan, G. Bishop, and H. Fuchs. The Delta Tree: An Object-Centered Approach to Image-Based Rendering. *AI Memo 1604*, AI Lab, Massachusetts Institute of Technology, 1996.

[Dey02] T. K. Dey, J. Hudson. PMR: Point to mesh rendering, a feature-based approach. In *Proceedings IEEE Visualization 2002*, pages 155–162. 2002.

[Fleishman03] S. Fleishman, D. Cohen-Or, M. Alexa, and C. T. Silva. Progressive point set surfaces. In *Proceedings of ACM SIGGRAPH 2003*, pages 997-1011, 2003.

[Flynn72] M. Flynn. Some Computer Organizations and Their Effectiveness, *IEEE Transaction of Computer*, Vol. C-21, pp. 948, 1972.

[Frécon 98] E. Frécon and M. Stenius. Dive: A scalable network architecture for distributed virtual environments. *Distributed Systems Engineering Journal*, 5:91_100, 1998.

[Gortler96] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In *Proceeding of SIGGRAPH 96*, pages 43–54. New Orleans, August 1996.

[Gross01] M. H. Gross. Are points the better graphics primitives. In *Computer Graphics Forum 20(3), 2001*. Plenary Talk Eurographics 2001.

[Gross03] M. Gross, S. Würmlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Meier-Koller, T. Svoboda, L. Van Gool, S. Lang, K. Strehlke, A. Vande Moere, O.Staadt. blue-c: A Spatially Immersive Display and 3D Video Portal for Telepresence, In *Proceedings of ACM SIGGRAPH 2003*, pages 819-827, ACM Press, 2003.

[Grossman98] J. P. Grossman and W. J. Dally. Point sample rendering. In *Proceedings Eurographics Rendering Workshop 98*, pages 181–192. Eurographics, 1998.

[Garcia03] A. Garcia, H. W. Shen: Asynchronous rendering for time-varying volume datasets on PC clusters. In *Proceedings of the IEEE Visualization 2003 Conference,* October 2003.

[Guennebaud04] G. Guennebaud, L. Barthe, M. Paulin. Deferred splatting. In *Proceedings of Eurographics 2004*. Computer Graphics Forum, Conference Issue. 2004.

[Hart82] J. C. Hart, D. J. Sandin, L. H. Kauffman. Ray tracing deterministic 3-D fractals. *Computer Graphics*, 23(3):289-296, 1989

[Hart90] J. C. Hart, L. H. Kauffman, D. J. Sandin. Interactive Visualization of Quaternion Julia Sets. *Proc. of Visualization '90. IEEE Computer Society Press*, pp. 209-218

[Heckbert89] P. S. Heckbert. Fundamentals of Texture Mapping and Image Warping. *Master's thesis*, University of California at Berkley, 1989.

[Holbrook83] J. Holbrook. Quaternionic asteroids and starfield. *Applied Mathematical Notes*, 8(2):1-34, 1983

[Hoppe92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of SIGGRAPH '92*, pages 71-78, July 1992.

[Huang96] M. Huang, D. Levine, L. Turner, L. Kettunen, and M. E. Papka. Virtual Reality Visualization of 3-D Electromagnetic Fields, Argonne National Laboratory, Argonne, Preprint ANL/MCS-P599-0596, August 1996.

[Hudson96] R. Hudson and A. Malagoli. Networked Virtual Reality for Real-Time 4D Navigation of Astrophysical Turbulence Data. *Simulation Multi-conference of the Society for Computer Simulation*, New Orleans, LA, 1996.

[Humphreys99]G. Humphreys, P. Hanrahan, A distributed graphics system for large tiled displays, In *Proceedings of Visualization '99*, pages 215-224, 1999.

[Humphreys00]G. Humphreys, I. Buck, M. Eldridge, P.Hanrahan. Distributed rendering for scalable displays, In PorACM/IEEE conference on supercomputing, pp.30, 2000.

[Humphreys02]G. Humphreys, M. Houston, R. Ng, R Frank, S.Ahern, P. D. Kirchner, J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *Computer graphics and interactive techniques*, pages 693-702, 2002.

[Kajiya84] J. T. Kajiya, B. P. Von Herzen: Ray tracing volume densities. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 165–174, 1984

[Kalaiah01] A. Kalaiah, A. Varshney. Differential point rendering. In *Proceedings Rendering Techniques*, pages 68–74. Springer-Verlag, 2001.

[Kalaiah03] A. Kalaiah, A. Varshney. Modeling and rendering points with local geometry. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):30–42, January-March 2003.

[Lacroute94] P. G. Lacroute, M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. *Computer Graphics* Vol28, Annual Conference Series (1994), 451–458.

[Lamboray04] E. Lamboray, S. Würmlin, M. Gross. Real-time Streaming of Point-based 3D Video, *Proceedings of the IEEE Virtual Reality (VR) 2004 Conference*, pages 91-98, IEEE Computer Society Press, 2004.

[Lario04] R. Lario, R. Pajarola, F. Tirado. Cached geometry manager for view-dependent LOD rendering. *Technical Report UCI-ICS-04-07*, Department of Computer Science, University of California Irvine, 2004.

[Laur91] D. Laur and P. Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. In *Proceedings of SIGGRAPH '91*, pp 285–288. Las Vegas, NV, 1991.

[Levoy85] M. Levoy and T. Whitted. The use of points as display primitives. *Technical Report TR 85-022*, Department of Computer Science, University of North Carolina at Chapel Hill, 1985.

[Levoy88] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, May 1988.

[Levoy90] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[Levoy96] M. Levoy and P. Hanrahan. Light Field Rendering. In *Proceedings of SIGGRAPH 96*, pages 31–42. New Orleans, August 1996.

[Levoy00] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of Siggraph 00*, pages 131-144, ACM SIGGRAPH, 2000.

[Li97] P. Li, S. Whitman, R. Mendoza, J.Tsiao. ParVox – a parallel spaltting volume rendering system for distributed visualization. In *Proceedings of 1997 Symposium on Parallel Rendering*, pp. 7–14. 1997

[Lippert95] L. Lippert and M. H. Gross. FastWavelet Based Volume Rendering by Accumulation of Transparent Texture Maps. *Computer Graphics Forum*, 14(3):431–444, August 1995.

[Lischinski98] D. Lischinski, A. Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Proceedings of the 9th Eurographics Workshop on Rendering 98*. Rendering Techniques. 301–314. 1998.

[Kooima07] R. L. Kooima, T Peterka, J. I. Girado, J. Ge, D. J. Sandin, T. A. DeFanti: A GPU Sub-pixel Algorithm for Autostereoscopic Virtual Reality, to be appeared at IEEE VR 2007.

[Ma00] K. L. Ma, D. M. Camp, High Performance Visualization of Time-Varying Volume Data over a Wide-Area Network Status, In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing,* pp. 59-69**.** Dallas, Texas, 2000.
[Ma03] K.L. Ma , A. Stompel , J. Bielak , O. Ghattas , E. J. Kim. Visualizing Very Large-Scale Earthquake Simulations, In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, p.48, 2003

[Mao96] X. Mao. Splatting of Non Rectilinear Volumes Through Stochastic Resampling. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):156–170, June 1996.

[McMillan95] L. McMillan and G. Bishop. Plenoptic Modeling: An Image-Based Rendering System. In *Proceedings of SIGGRAPH 95*, pages 39–46. ACM SIGGRAPH, Los Angeles, August 1995.

[Mitra03] N. J. Mitra, A. Nguyen. Estimating surface normals in noisy point cloud data. In *Proceedings of the 2003 Symposium on Computational Geometry*, pages 322–328. ACM Press, 2003.

[Molnar94] S. Molnar, M. Cox, D. Ellsworth,H. Fuchs, A Sorting Classification of Parallel Rendering, *IEEE Computer Graphics and Algorithms*, pp. 23-32, 1994.

[Mueller96] K. Mueller, R. Yagel. Fast Perspective Volume Rendering with Splatting by Utilizing a Ray-Driven Approach. *IEEE Visualization '96*, pages 65–72, October 1996.

[Mueller98] K. Mueller, R. Crawfis. Eliminating Popping Artifacts in Sheet Buffer-Based Splatting. In *Proceedings of IEEE Visualization '98*, pages 239–246, October 1998.

[Nortan82] A.Nortan. Generation and rendering of geometric fractals in 3-D. *Computer Graphics*, 16(3):61-67, 1982

[Nyland01] L. Nyland, A. Lastra, D. K. McAllister, V. Popescu, and C. McCue. Capturing, processing and rendering real-world scenes. In *Videometrics and Optical Methodsfor 3D Shape Measurement, Electronic Imaging 2001, Photonics West*, volume 4309, pages 107–116. SPIE, 2001.

[Oosterbaan98] C. Oosterbaan. Motion and Deformation of Surfel Objects. *Master's thesis*, Delft University of Technology and MERL, 1998.

[Pajarola03a] R. Pajarola. Efficient level-of-details for point based rendering. In *Proceedings IASTED International Conference on Computer Graphics and Imaging (CGIM 2003)*, 2003.

[Pajarola03b] R. Pajarola, M. Sainz, P. Guidotti. Object-space point blending and splatting. In *ACM SIGGRAPH Sketches & Applications Catalogue*, 2003.

[Pajarola03c] R. Pajarola, M. Sainz, Y. Meng. Depth-mesh objects: Fast depth-image meshing and warping. *Technical Report UCI-ICS-03-02*, The School of Information and Computer Science, University of California Irvine, 2003.

[Pajarola04a] R. Pajarola, M. Sainz, P. Guidotti. Confetti: Object-space point blending and splatting. *IEEE Transactions on Visualization and Computer Graphics*, pages 134–140, 2004.

[Pajarola04b] R. Pajarola, M. Sainz, Y. Meng. DMesh: Fast depth-image meshing and warping. *International Journal of Image and Graphics (IJIG)*, pages 361–370, 2004.

[Papka97] M. E. Papka, R. Stevens, and M. Szymanski. Collaborative Virtual Reality Environments for Computational Science and Design, *Computer-Aided Design of High-Temperature Materials*, Santa Fe, New Mexico, 1997.

[Pauly01] M. Pauly, M. Gross. Spectral processing of point-sampled geometry. In *Proceedings ACM SIGGRAPH 2001*, pages 379–386. ACM Press, 2001.

[Pauly02] M. Pauly, M. Gross, L. P. Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings IEEE Visualization 2002*, pages 163–170. Computer Society Press, 2002.

[Pauly03] M. Pauly, R. Keiser, L. Kobbelt, M. Gross. Shape modeling with point-sampled geometry. In *Proceedings ACM SIGGRAPH 2003*, pages 641–650. 2003.

[Pfister99] H. Pfister, Architectures for Real-Time Volume Rendering, *Journal of Future Generation Computer Systems (FGCS)*, Vol. 15, No. 1, p. 1-9, February 1999

[Pfister00] H. Pfister, M. Zwicker, J. van Baar, M. Gross. Surfels: Surface elements as rendering primitives. In *Proceedings SIGGRAPH 2000*, pages 335–342. ACM SIGGRAPH, 2000.

[Policarpo05] F. Policarpo, F. Fonseca: Deferred Shading Tutorial.

[Ren02] L. Ren, H. Pfister, M. Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Proceedings EUROGRAPHICS 2002*, pages 199–206. 2002.

[Reeves83] W. T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactionson Graphics*, 2(2):91–108, Apr. 1983.

[Reeves85] W. T. Reeves, Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems, *Computer Graphics*, vol. 19, no. 3, pp 313-322, 1985.

[Reynolds87] C. W. Reynolds, Flocks, Herds, and Schools: A Distributed Behavioral Model, *Computer Graphics*, vol. 21, no. 4, pp 25-34, 1987.

[Rusinkiewicz00] S. Rusinkiewicz, M. Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings SIGGRAPH 2000*, pages 343–352. ACM SIGGRAPH, 2000.

[Rusinkiewicz01] S. Rusinkiewicz, M. Levoy,Streaming QSplat: a viewer for networked visualization of large, dense models, symposium on Interactive 3D graphics,pp 63-68, 2001.

[Rusinkiewicz02] S. Rusinkiewicz, O. Hall-Holt, M. Levoy. Real-time 3D model acquisition. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages: 438 – 446. ACM Transactions on Graphics, 2002

[Sainz04] M. Sainz, R. Pajarola, A. Susin, A. Mercade. SPOC: Simple point-based object capturing. *IEEE Computer Graphics & Applications*, pages 121–128, July-August 2004.

[Saito90] T. Saito, T. Takahashi. *Comprehensible Rendering of 3-D Shapes*. In *Proceedings of ACM SIGGRAPH 90*, Computer Graphics, volume 24, pp. 197–206, 1990.

[Sandin05] D. Sandin, T. Margolis, J. Ge, J. Girado, T. Peterka, T. DeFanti. The VarrierTM Autostereoscopic Virtual Reality Display, In *Proceedings of ACM SIGGRAPH 2005*, SIGGRAPH 2005, July 2005

[Schulze03] J. P. Schulze, R. Niemeier, U. Lang. The Perspective Shear-Warp Algorithm in a Virtual Environment. In *Proceedings of IEEE Visualization 2001*, pp. 207-213, IEEE, ISBN 0-7803-7200-X, 2003

[Schaufler00] G. Schaufler and H.W. Jensen. Ray tracing point sampled geometry. In *Eurographics Rendering Workshop Proceedings*, pages 319–328, 2000.
[Shade98] J. W. Shade, S. J. Gortler, L. He, and R. Szeliski. Layered Depth Images. In Proceedings of ACM *SIGGRAPH 98*, pages 231–242. ACM SIGGRAPH 98, July 1998.

[Shalf03] J. Shalf, E. W. Bethel: The Grid and Future Visualization System Architectures. *IEEE Computer Graphics and Applications* 23(2): 6-9 (2003)

[Smith84] A. R. Smith. Plants, fractals and formal languages. In *Proceedings of SIGGRAPH 84*, pages 1–10, 1984.

[Soucy92] M. Soucy, D. Laurendeau. Multi-resolution surface modeling from multiple range views. In *Proceedings of CVPR'92*, pp. 348-353, June 1992.

[Stamminger01] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In *Eurographics Workshop on Rendering 2001*, pages 151-162, 2001.

[Swan97] J. E. Swan, K. Mueller, T.M¨oller, N. Shareef, R. Crawfis, and R. Yagel. An Anti-Aliasing Technique for Splatting. In *Proceedings of the 1997 IEEE Visualization Conference*, pages 197–204. Phoenix, AZ, October 1997.

[Szeliski93] R. Szeliski. Rapid Octree Construction from Image Sequences. *CVGIP: Image Understanding*, 58(1):23–32, July 1993.

[Turk94] Turk, G. and Levoy, M., Zippered polygon meshes from range images. In *Proceedings of SIGGRAPH'94*, pp. 311-318, July 1994.

[Teller98] S. Teller. Toward urban model acquisition from geolocated images. In *Proceedings of Pacific Graphics '98*, pp. 45-52, Oct. 1998.

[Torborg96] J. Torborg and J. Kajiya. Talisman: Commodity Real-time 3D Graphics for the PC. In *Proceedings of ACM SIGGRAPH 96*, pages 353–364. ACM SIGGRAPH 96, New Orleans, August 1996.

[VanGelder96] A.Van Gelder and K. Kim. Direct Volume Rendering with Shading via Three-Dimensional Textures. In *ACM/IEEE Symposium on Volume Visualization*, pages 23–30. San Francisco, CA, October 1996.

[Wand01] M. Wand, M. Fischer, I. Peter, F. M. Auf Der Heide, W. Strar. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *Proceedings of SIGGRAPH 2001*. ACM Press / ACM SIGGRAPH, 361–370. 2001.

[Watson98] K. Watsen and M. Zyda. Bamboo - a protable system for dynamically extensi-ble, real-time, networked, virtual environments. In *IEEE Virtual Reality Annual Internationnal Symposium*, Georgia, USA, 1998.

[Wellner93] Wellner, P., Mackay, W. & Gold, R. Eds. Special issue on computer augmented environments: back to the real world. *Communications of the ACM*, Volume 36, Issue 7 (July 1993).

[Westover89] L. Westover, Interactive volume rendering, *Chapel Hill workshop on Volume visualization*, pages 9-16, 1989.

[Westover90] L. Westover. Footprint Evaluation for Volume Rendering. In *Proceedings of SIGGRAPH 90*, pages 367–376. August 1990.

[Wes91] L. A. Westover. Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm. *PhD thesis*, The University of North Carolina at Chapel Hill, Department of Computer Science, July 1991.

[Wurmlin03] S. Würmlin, E. Lamboray, O. Staadt, M. Gross. 3D Video Recorder: A System for Recording and Playing Free-Viewpoint Video, *Computer Graphics Forum 22 (2)*, pages 181-193, Blackwell Publishing Ltd, Oxford, U.K., 2003.

[Wurmlin04] S. Würmlin, E. Lamboray, M. Waschbüsch, M. Gross, Dynamic Point Samples for Free-Viewpoint Video, In *Proceedings of the Picture Coding Symposium (PCS) 2004*, 2004.

[Yu04] H. Yu, K.-L. Ma, J. Welling, I/O strategies for parallel rendering of large time-varying volume data, In *Proceedings of Parallel Graphics and Visualization 2004*, Eurographics/ACM SIGGRAPH Symposium 2004, pages 31-40.

[Zwicker01a] M. Zwicker, H. Pfister, J. van Baar, M. Gross. Surface splatting. In *Proceedings of SIGGRAPH 2001*, pages 371–378. ACM SIGGRAPH, 2001.

[Zwicher01b] M. Zwicker, H. Pfister, J.v.Baar, M.Gross, Ewa volume splatting, In *Proceedings of IEEE Visualization 2001*, pages 29-36, 2001.

[Zwicker02] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. PointShop 3D: An Interactive System for Point-Based Surface Editing. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 322-329, 2002

[Zwicher04] M. Zwicher, J. Ren., M. Botsch, C. Dachsbacher, M. Pauly. Perspective accurate splatting. In *Proceedings of Graphics Interface 2004*, pages 247–254. 2004

VITA

NAME JINGHUA GE

EDUCATION

2000-2007 **University of Illinois at Chicago, Chicago, USA**
Ph.D. student

1997-2000 **Tsinghua University, Beijing, China**
Master of Computer Science

1993-1997 **Beijing Information Technology Institute, Beijing, China**
Bachelor of Computer Science

PUBLICATIONS

1. J. Ge, D. J. Sandin, A. Johnson, T. Peterka, R. L. Kooima, J. I. Girado, T. A. DeFanti. Point-based VR visualization for large-scale mesh datasets by real-time remote computation, In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, VRCIA '06 , HongKong, June 2006

2. J. Ge, D. J. Sandin, A. Johnson, T. Peterka, R. L. Kooima, J. I. Girado, T. A. DeFanti. A Point-based Asynchronous Remote Visualization Framework for Real-time Virtual Reality, to be appeared at *International Journal of Image and Graphics*, 2007

3. D. J. Sandin, T. Margolis, J. Ge, J. Girado, T. Peterka, T. DeFanti. The VarrierTM Autostereoscopic Virtual Reality Display, In *Proceedings of SIGGRAPH 2005*, pages 894-903, SIGGRAPH 2005, July 2005.

4. T. Peterka, D. J. Sandin, J. Ge, J. Girado, R. Kooima, J. Leigh, A. Johnson, M. Thiebaux, T. A. DeFanti. Personal Varrier: Autostereoscopic Virtual Reality Display for Distributed Scientific Visualization. In *Journal of Future Generation Computing Systems*, vol 22, no 8, pp. 976-983

5. J. Ge, D. Sandin, T. Peterka, T. Margolis, T. DeFanti. Camera Based Automatic Calibration for the VarrierTM System, In Proceedings *of IEEE International Workshop on Projector-Camera Systems* (PROCAM 2005), San Diego, 2005.

6. J. Ge, T. Peterka, R. L. Kooima, V. Vishwanath, D. J. Sandin, A. Johnson. A Distributed Volume Rendering Pipeline for Networked Virtual Reality, to be appeared at *International Workshop on Network-based Virtual Reality and Tele-existence* (INVITE 2007).

7. T. Peterka, R. L. Kooima, J. I. Girado, J. Ge, D. J. Sandin, T. A. DeFanti. Evolution of the Varrier™ Autostereoscopic VR Display: 2001-2007, In *Proceedings of SPIE 2007*.