Rain Table: Scalable Architecture for Group-Oriented Visualization of Real-Time Geoscience Phenomena

BY

Dmitri Nikolai Svistula,

B.S., University of Illinois at Chicago, 2004

PROJECT

Submitted as partial fulfillment of the requirements for
the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2008
Chicago, Illinois

TABLE OF CONTENTS

SECTION PAGE

1 Introduction and Motivation.... 6

2 Background and Related Work.... 10

2.1 Tiled Display Interaction.... 10

2.2 Parallel Simulation and Visualization.... 12

2.3 High Resolution Image Rendering.... 13

2.4 Interactive Applications for Horizontal Displays.... 13

3 Design and Implementation.... 15

3.1 Design Goals.... 15

3.2 Basic Renderer.... 15

3.2.1 Scene Graph.... 16

3.2.2 Scene Node.... 16

3.2.3 Event System.... 16

3.3 High Resolution Image Rendering.... 16

3.3.1 Tiling.... 17

3.3.2 Indexing.... 17

3.3.3 Compression.... 18

3.3.4 Request System.... 18

3.3.5 Tile Caching.... 21

3.3.6 Data Traversal.... 22

3.3.7 LOD Approximation.... 23

3.4 Particle-Based Simulation.... 24

3.4.1 Cluster Communication.... 24

3.4.2 Simulation on a Grid.... 28

3.4.3 Screen Space.... 29

3.4.4 Simulation Zones.... 33

3.4.5 Visualization Pipeline.... 34

3.4.6 Load Management Mechanisms.... 37

3.4.7 Data Caching....42
3.4.8 Multithreading....42
3.5 Synchronization....42
3.6 Scalability....44
4 Interaction....45
4.1 Basic Navigation....45
4.2 Magnifiers....45
4.3 Inputs, Triggers, and Modifiers....46
5 Applications....48
5.1 Flow Model Calculation....48
5.2 Rainfall Runoff....52
5.3 Sediment Flux....53
5.4 Lava Flow....53
5.5 Pyroclastic Flow....54
5.6 Glacial Movement....57
5.7 Water Pollution....60
6 Limitations and Future Work....61
7 Conclusion....63
8 Current System....64
9 Gallery....70

REFERENCES....73

ACKNOWLEDGEMENTS

I would like to thank Jason Leigh, my advisor, Paul Morin, and Pat Hamilton for supporting me in this work. Thanks to Andy Johnson and Luc Renambot for your suggestions and valuable input. To Lance Long and Pat Hallihan for all the technical support you have done. To my lab mates, Ratko Jagodic, Arun Rao, Don Olmsted, Hyejung Hur, and Cole Krumbholz for your help and suggestions. To Ryan Currier for working out and helping me understand and simplify some of the science mathematics.

D.S.

SUMMARY

The contribution of this work is:

1. Development of a decentralized particle-based simulation model that can be applied to many times of simulations.

2. Development of an approach to efficient coupling of visualizations and particle-based simulations on high resolution tiled displays.

3. Application of traditional out-of-core and LOD methods to interactive high resolution environments.

4. Application of visualization research technology to informal science education.

1 Introduction and Motivation

High resolution tiled displays are widely used by researchers to visualize large data. When moving to high resolution environments and large interaction spaces, the ability of software systems to scale up is crucial. Scalable high resolution tiled displays have broken the boundary of limited resolution by tiling together multiple LCDs or projectors that are built around clusters of computers. Scalable tracking systems have been developed for such devices, enabling multi-user interaction with visualizations. There is plenty of research in the area of simulation and visualization using clusters and grids, however most of it is aimed at speeding up these processes and displaying them back on a single screen. The assumed modes of interaction with these processes range from one user to completely non-interactive systems. The motivation for this work is to develop a scalable system to provide group-oriented interaction with very large scale simulations and visualizations using high resolution tiled displays and apply it to the domain of science education. Simulations and visualizations are both CPU and GPU intensive. It is possible to decouple these processes by running them on separate machines or clusters, however such setup is not suitable for museum or classroom settings and requires high speed network interconnection outside of the tiled display. This research project builds on the knowledge of scalable tiled displays and scalable tracking technology to provide guidelines for developing scalable interactive simulations and visualizations on tiled display hardware directly. As the proof of concept, this research project implements an application called Rain Table using a high resolution tiled tabletop system called LambdaTable [1] developed at the Electronic Visualization Laboratory (EVL), which provides 24 megapixels of interactive screen space.

Rain Table is an application for large-scale interactive 2D particle-based simulation and visualization on scalable high-resolution tiled displays for group-oriented interaction with geoscience content. The goal of this research project is to apply high resolution visualization technology to geoscience education in museums or classrooms to provide educators with new means of visualizing and simplifying the understanding of complex scientific phenomena. The work in this project is aimed at informal education in science museums, however it may potentially be of use to real scientists. This project implements software that allows users to interact with visualizations by taking advantage of novel scalable tracking technology for tiled displays, developed at EVL. Rain Table implements an out-of-core image rendering framework that uses tiling, on-demand texture paging, multi-level caching, compression, and threading techniques to minimize latencies and maintain system interactivity. The core of Rain Table is a decentralized particle-based simulation and visualization model that runs alongside the rendering nodes of tiled displays. The first goal of this project is to design and implement the architecture to run and display simulations of rainfall runoff and the context for these simulations, such as aerial photography in high detail. The second goal is to prepare this architecture for other types of simulations and assess its scalability in terms of data size, output resolution, communication, computation, and interaction space.

Figure 1. Students interacting with a physical model of a water table.

On the application side, the original idea and foundation for this work revolve around watershed education and various outreach projects conducted by the National Center for Earth-Surface Dynamics (NCED) at the University of Minnesota. Figure 1 shows a group of students sprinkling water on top of a physical model of a region. This is a great way to visualize the paths water takes though watersheds and how it connects different sub-regions. However, it is a poor substitute for the complexities of water flow in the real world. It is also very hard for students to think about the phenomena on a more global scale, because physical models are usually meant to represent very small regions. Physical models are not flexible enough because a separate model has to be built for different types of terrains. This process is time consuming and the resulting models take up a lot of space. A solution that is more modular and versatile can help in solving these problems.

The ideal solution is a large table surface that resembles a physical water table, but also enables students to:

1. Quickly experiment over any terrain map and switch between them.
2. Take advantage of very high resolution observatory data such as satellite topography and aerial photography.

The rainfall runoff simulation in Rain Table allows users to interactively explore the flow of water across maps and discover the concepts of watersheds, floods, and the interconnectivity of river systems. Multiple users can generate water flow independently of one another, so small groups work together interactively, eliminating passive viewing and equalizing user's control of the visualization model.

2 Background and Previous Work

This section goes over background and previous work.

2.1.1 Tiled Display Interaction

The usefulness of visualization on tiled displays is apparent as the size of visualized data becomes large. This applies to data produced by instruments in the fields of medicine, biology, geoscience, and physics. Tiled display systems can output more resolution and therefore display data in more detail. Various modes of multi-user interaction with large displays have been proposed. These techniques vary from laser pointers [2] to those using gestural input directly [3] and those using a combination of acoustics and computer vision [4]. Ball [5] suggests that tiled displays improve performance for basic visualization tasks. The findings also suggest that physical navigation is more common to high-resolution displays than low resolution and physical navigation is preferred. This research suggests that high resolution displays may be better fitted for group work. Display size and resolution need to grow in order to maintain the same space to user ratio for a group of people.

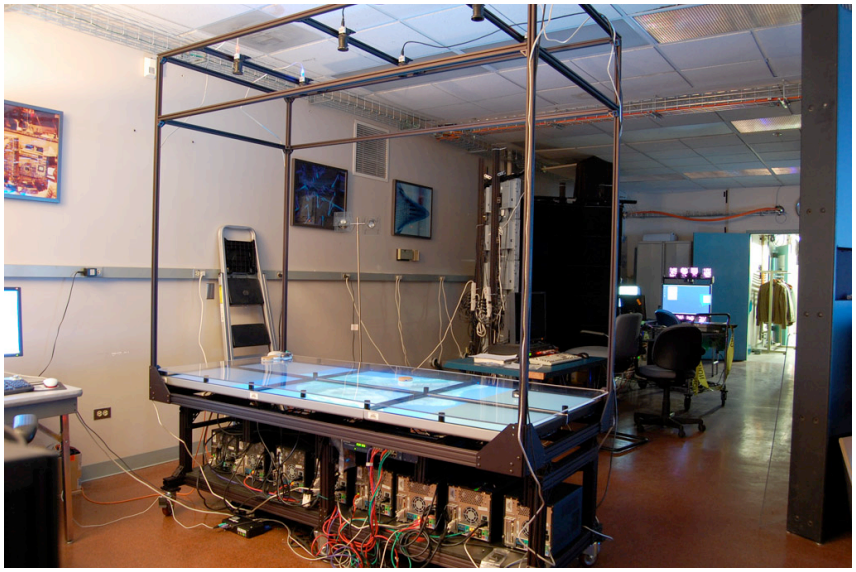


Figure 2. EVL's LambdaTable.

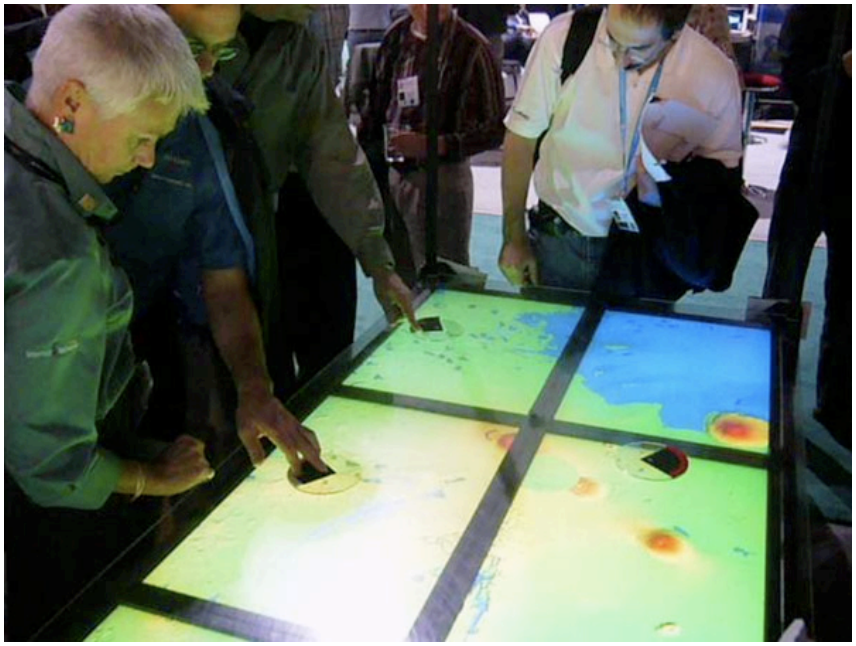


Figure 3. LambdaTable interaction.

Moving on to horizontal displays, previous research suggests that tabletops are better fitted for co-located group work than wall displays [6]. Much of the research related to high resolution tiled tabletop displays is outlined in [1], which also implements a solution for scalable multi-user

tracking for a high resolution tiled table, LambdaTable, shown in Figure 2. This system is used in this project. LambdaTable is a 7 by 3 foot, 24-megapixel cluster of 6 nodes build using six Dell LCD 2560 by 1600 pixel displays. There are three rendering nodes and three tracking nodes. Each tracking node is connected to an IR camera that is mounted overhead. Unique patterns of retro-reflective markers are used to determine locations and orientations of objects placed on the tabletop. Figure 3 shows a group of people interacting with LambdaTable.

2.1.2 Parallel Simulation and Visualization

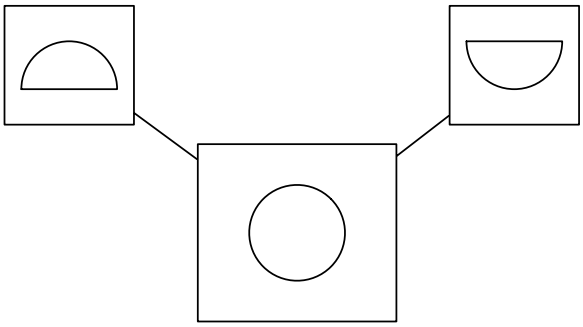


Figure 4. Parallel visualization.

Traditional parallel visualization techniques attempt to render a part of the entire visualized data on a cluster node and transmit these rendered pixels to the visualization host that combines pieces into the entire image. This is shown in Figure 4. Alternatively, the results can also be displayed on the cluster nodes. Traditional parallel or distributed simulations works in a similar fashion by decomposing computational tasks either functionally or spatially. This way, clusters of computers are efficient for simulating complex problems. However, the traditional use of clusters for simulations is to do intensive and non-interactive calculations. It is possible to use both computational and rendering facilities of cluster nodes to display intermediate results of simula-

tions. Allard [7] discusses coupling of parallel simulation with visualization on rendering nodes of a cluster. Their system is able to interactively run and display a small-scale simulation of two fluids and a simulation of a piece of cloth. The interest in parallelism in this project is to provide support for larger interaction area, simulation, and data given a scalable display and interaction system. It is not directly aimed to speed up the simulations or visualizations.

2.1.3 High Resolution Image Rendering

The current developments in the area of high resolution image rendering on tiled displays are aimed at bridging the gaps between data size and the network, visualization, computation, and storage resources available on remote or local clusters. All of the known image viewers have limitations and are not ready for highly interactive multi-user collaborative environments. TimV [8] relies highly on virtual memory. Argonne's image viewer [9] does not give an option of zooming. JuxtaView [10], developed at EVL, does not deal with maintaining interactivity during loading of data. The task of interactive navigation is not as crucial as the ability to view the data, however it is important to provide timely user feedback in a direct, interactive system that is also a learning tool.

2.1.4 Interactive Applications for Horizontal Displays

Low-resolution tabletop devices have become very popular in museums [11]. There is a number of interactive pieces employing either tabletops or tiled displays. Some of the more notable ones in the domain of entertainment and education are Reactable [12] and Shigureden's tiled floor [13]. Reactable is a collaborative tabletop music instrument that produces sounds when users put unique trackable objects on top of it. The unique objects represent modular com-

ponents of audio synthesis that are able to connect with each other to produce sound. Reactable is a multi-user, controlled, collaborative environment.

Figure 5. Shigureden tiled floor.

The tiled floor at Shigureden is a Nintendo-powered museum exhibit in Arashiyama, Kyoto. Location aware Nintendo DS devices are handled by users that walk around a large map of Kyoto. The devices are used by museum goers to select locations in Kyoto to be guided to by a virtual guide shown on the floor display. The large display also shows animations of city ponds and moving cars on the map. Shigureden tiled floor is an inspiration for this work.

3 Design and Implementation

This portion of the text goes over design and implementation. Section 3.1 outlines the overall design goals. Section 3.2 describes the rendering system used. Section 3.3 goes over the rendering system for high resolution images. Section 3.4 describes a decentralized model for particle-based simulation, its visualization, and the way of interacting with it. Section 3.5 outlines synchronization mechanisms. Finally, Section 3.6 analyzes the scalability of the system.

3.1 Design Goals

In order to achieve scalability, this research project will:

1. Analyze and develop a parallel visualization framework.
2. Analyze and develop a decentralized simulation model.
3. Analyze and employ decentralized data access and caching mechanisms.

The main goal is to design a system where individual components work independently of each other and independently of any common shared components as much as possible.

3.2 Basic Renderer

The rendering framework for this project was implemented in C/C++ using OpenGL for graphics, and FLTK [14] for windowing. It is based on a scene graph, which arranges objets in a hierarchical manner making it easy to prevent drawing visual content that is not visible on a single rendering node of the tiled display. MPI [15] and Quanta [16] are used for cluster communication and synchronization of graphics and simulations.

3.2.1 Scene Graph

Scene Graph is a data structure that is a representation of the spatial arrangement of scene objects. The need for a scene graph structure is useful to efficiently determine visibility of scene objects in a tiled display environment.

3.2.2 Scene Node

A scene node is an object that inherits certain basic functionality that is accessed during scene graph traversal and implements new functionality. This object is used as a building block to render visualizations.

3.2.3 Event System

Each scene node carries an identification that is unique on the entire cluster. This way, remote events can be delivered to the right scene objects. A hash table of all scene nodes is created at startup. Addition and removal of scene nodes is handled dynamically by performing relevant operations on the hash table. The user interface features in this project are implemented using the observer-listener design pattern. User interface objects follow the same identification scheme and are consistent across the rendering nodes of the cluster.

3.3 High Resolution Image Rendering

This part of the document goes the design and implementation of the high resolution image rendering framework implemented in this project. It builds on traditional techniques in LOD and employs concurrency mechanisms to achieve interactive frame rates.

This image rendering framework uses tiling, texture paging, multi-level caching, compression, and threading techniques to provide fast access to any region of interest in an image. High resolution image rendering of a single image is implemented within a single scene node. This allows overlaying of data and the addition of multiple images in the same context.

Certain assumptions have to be made when dealing with large data in a cluster environment:

1. All data may not fit into texture memory of a rendering node.
2. All data may not fit into main memory of a rendering node.
3. All data may not fit into distributed memory of a cluster.

These assumptions suggest the necessity for out-of-core techniques.

3.3.1 Tiling

Tiling is an important step in large image processing because regularly sized pieces of image can be paged to texture memory and cached more efficiently. For this reason, the entire image is processed into a quad-tree pyramid of tiles from coarsest to finest resolution, sub-sampling data by a factor at each coarser level. This standard data partitioning scheme allows us to design an efficient data structure for traversing the data.

3.3.2 Indexing

A multi-level index array of tiles is build for each image. This data structure is used for generating data tile requests and traversing the data using a quad tree, which is implicitly defined by the index array.

3.3.3 Compression

Compression of image data reduces its storage requirements on disk and in main memory. It also decreases the time required to transfer image data to texture memory. Original data is compressed in DXT1 format using the squish library [17]. Decompression of the data happens entirely on the GPU. This improves performance when many textures have to be paged into GPU client memory repetitively.

3.3.4 Request System

Requests for data are made at multiple system levels. This allows us to decouple rendering and data access. Figure 6 shows all possible data requests in the current implementation. L_1 is a request for a data tile from remote disk to node's main memory. L_2 is a request for data tile from node's main memory to node's texture memory. A more robust solution could also implement requests for transfer of data from a remote node's memory instead of the remote disk, however such system introduces an extra level of memory management on the entire cluster. The goal of this particular implementation is to provide a stable level of interactivity at all times and not necessarily improve the speed at which data may arrive and be displayed. Instead of a more robust memory management system, this solution implements loading of data into main memory (L_1) asynchronously on a thread that runs concurrently with the main thread. Main memory acts as a buffer between disk and texture memory. This prevents the main thread to stall when the remote disk is accessed. The remote disk is currently represented seamlessly via NFS [18] or PVFS [19] file systems. In the case of NFS, the data resides on a disk of only one node, the master node. Data access is therefore always from this remote disk. When a node tries to access a piece of

data, other nodes requesting data are locked until it finishes. PVFS is much more efficient since data is distributed across disks of the entire cluster, so the loading of data may or may not be directly from the local disk. This prevents locking from happening as frequently. The threads of L_1 and L_2 communicate only when new data has been loaded into the main memory. Figure 7 shows the request system in more detail represented in the context of the two threads, the main thread (L_1) and the fetcher thread (L_2). Each thread maintains a cache of tiles.

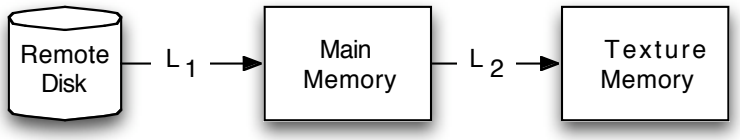


Figure 6. Levels of requests for a data tile.

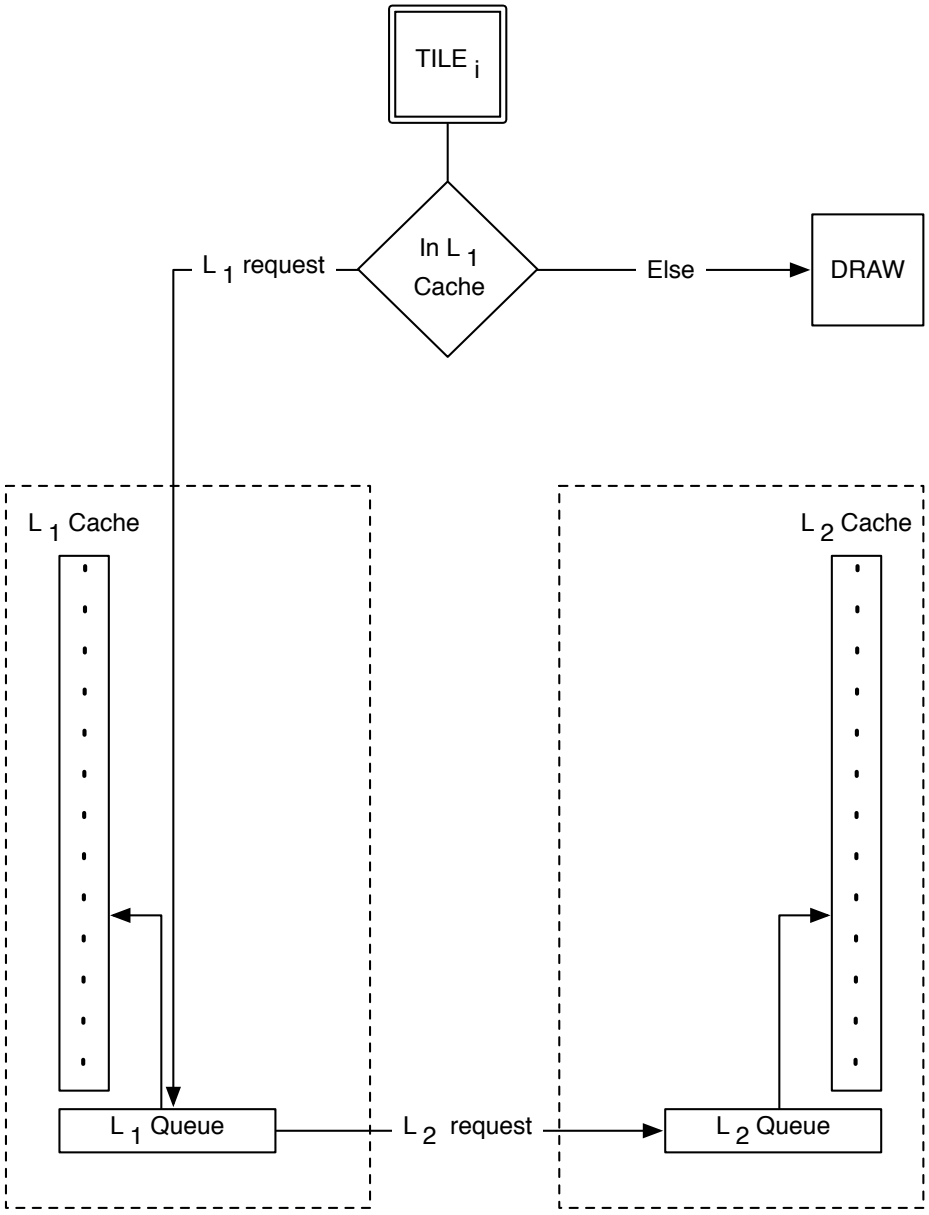


Figure 7. Request system and caches in detail. If a tile exists in L_2 cache, it is drawn, otherwise a request for it is made from L_1 cache. If it doesn't exist in either caches, a request is made to read it from a remote disk.

3.3.5 Tile Caching

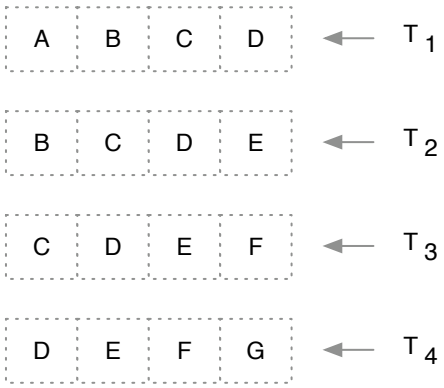


Figure 8. LRU Cache additions of 4 tiles.

The caches described here are caches that exist in memory of each rendering node. Sizes of the caches are set by the user and are represented by the number of data tiles that can be held in source memory of a cache. The caches of tiles in both L_1 and L_2 are implemented as LRU (least recently used) caches. If a tile is present in a cache when it is requested, that tile is pushed to the front of the cache. When a new tile is cached, the least recently used tile is popped off the cache and discarded. This is illustrated in Figure 8 for caching off four tiles. The L_2 cache introduces an extra step when popping because the data contained in this cache may currently be displayed. The L_2 cache is actually a restricted LRU implementation. Whether a tile can be safely popped off L_2 is determined during traversal of the data and is based on the region of interest (ROI) of the rendered image. This may occasionally produce a condition where the texture cache slightly exceeds the desired size limit. However, both caches are updated at regular intervals, so the condition is temporary and may last only a few seconds.

3.3.6 Data Traversal

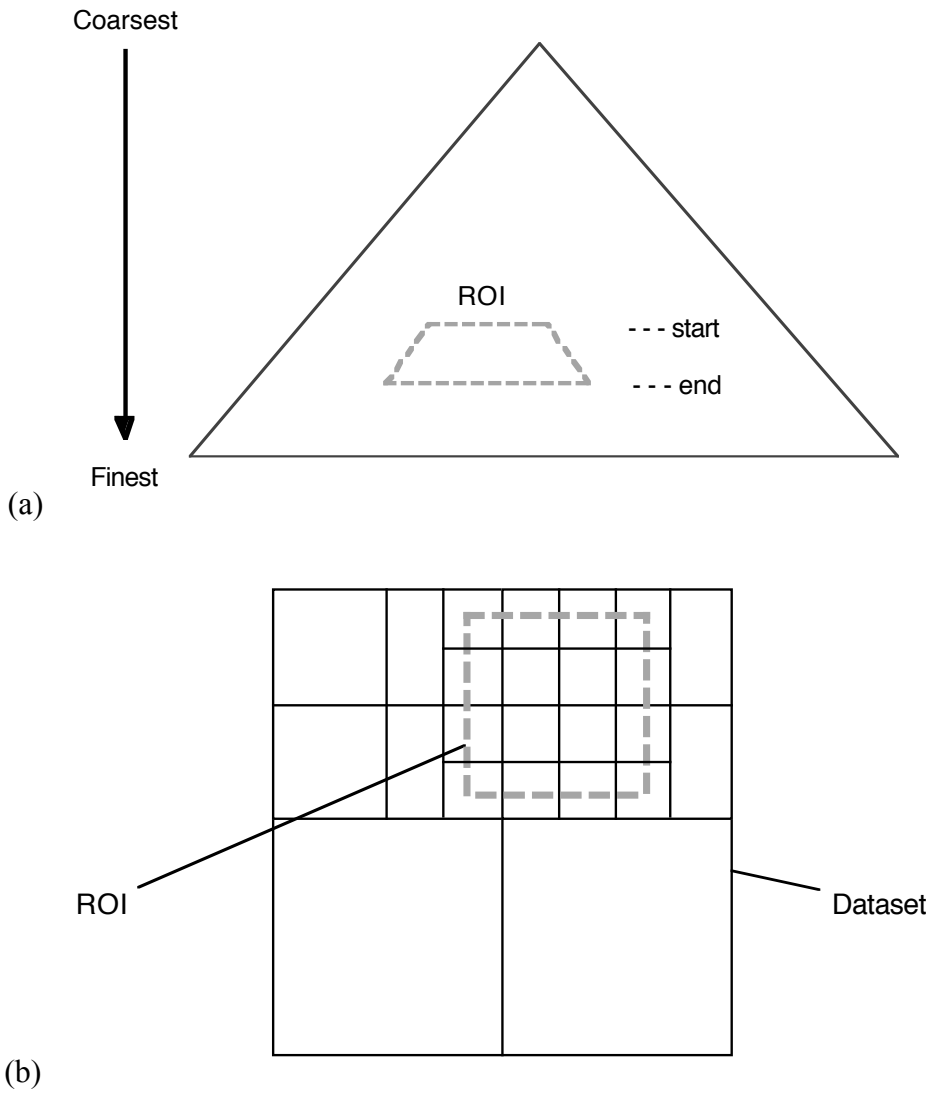


Figure 9. Region of interest traversal of the image pyramid. Side view (a), top view (b).

The data is traversed only within the region of interest, which is based on the view frustum. Data that falls outside the view frustum is culled by comparing the data tile's bounding box to the view frustum. Since we are working in 2D using orthogonal projection, this test is very simple. Each tile's bounding box is tested against the left, right, top, and bottom planes of the view frustum. The test that determines whether something is inside or outside the view frustum is

known as the frustum test. Only a portion of the entire quad-tree is traversed at a given time, however traversal may not start at the root node. Traversal from the root node for large data increases storage requirements and computation. Oftentimes coarser data is never actually displayed and the calculations to display it become unnecessary. For this reason, we start traversal at tiles that are several levels coarser than the current level of detail (LOD). The locations of these tiles are determined as the subset of all tiles at that coarser level that pass the frustum test. Figure 9 shows the traversal of data. Traversal stops at current LOD. The coarsest tile of the image is loaded and displayed at all times. This method decreases storage requirements, calculations, and data requests when traversing very large images, but creates more unpleasant blending artifacts when zooming in and out with a large frequency of cache misses.

3.3.7 LOD Approximation

Data tile sizes in node coordinates are searched from coarsest to finest until condition (3) is met. When the condition is met, the level of detail of that tile becomes the LOD level for that image. The following equations go over approximating LOD. The idea behind these calculations is to find a match between the rendered size of a tile and its pixel size to be displayed on the screen.

$$W_W = P_W T_W \quad (1)$$

$$H_W = P_H T_H \quad (2)$$

$$W_N H_N S^L \geq W_W H_W L^S \quad (3)$$

where,

W_N: width of data tile in scene node coordinates

H_N: height of data tile in scene node coordinates

W_W: width of data tile in world coordinates

H_W: height of data tile in world coordinates

P_W: width of one screen pixel in world coordinates

P_H: height of one screen pixel in world coordinates

T_W: width of data tile in pixels

T_H: height of data tile in pixels

L: quad-tree level

S: sampling factor

3.4 Particle-Based Simulation

This part of the document goes over a decentralized particle based simulation and visualization model that runs in the screen spaces of rendering nodes.

3.4.1 Cluster Communication

Communication between cluster nodes is essential in order to execute updates of the entire simulation in sync. The idea is to distribute particles spatially across all nodes. This means that the simulated particles may travel in and out of the extents of a node's screen space at any time during simulation. Figure 10 illustrates a possible path that a particle may take. As an obvious solution, it is possible to route the particles that go outside a node's extent through one node, such as the master node, and have it calculate where particles should go. However, as the number of particles grows, this becomes inefficient because one node has to handle sorting of all particles and forwarding them to appropriate nodes.

A more efficient solution is to set up communication between nodes that follows the screen layout of nodes in the tiled display. This provides direct communication links between adjacent

nodes without the need of a proxy node. In order to set up this communication properly, the physical layout of nodes has to be taken into account. This is because the layouts of physical screens on each rendering node of the entire tiled display may differ. Rain Table reads the configuration of the screens of rendering nodes from a file, it is shown in Figure 5. This file provides information about the physical mappings of screens according to nodes' host names and IP addresses. Assuming that the tiled display is set up as a regular grid of screens, this configuration makes it easy to calculate the exact connections that have to be established between nodes.

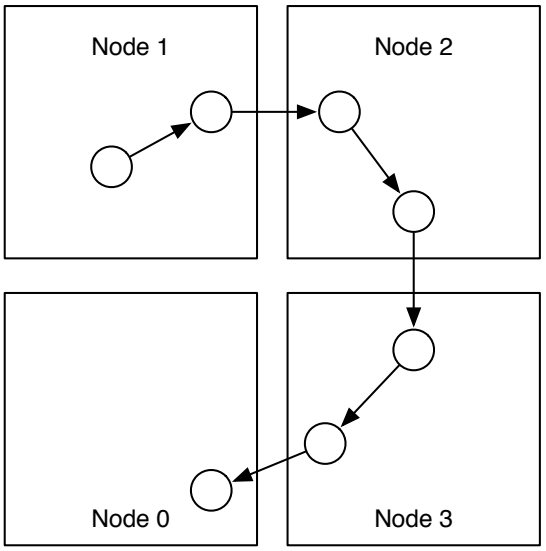


Figure 10. Example path of a particle traveling across multiple nodes.

Figure 11 shows all communication links necessary to execute simulation updates for the tiled display configuration in Figure 12.

```
TileDisplay
    Dimensions 4 2
    Mullions 2.5 2.5
    Resolution 400 400
    PPI 72
    Machines 6

DisplayNode
    Name teiburu1-10
    IP 10.0.8.11
    Monitors 1 (3,1)

DisplayNode
    Name teiburu2-10
    IP 10.0.8.12
    Monitors 1 (2,1)

DisplayNode
    Name teiburu6-10
    IP 10.0.8.16
    Monitors 1 (1,1)

DisplayNode
    Name teiburu3-10
    IP 10.0.8.13
    Monitors 2 (1,0) (2,0)

DisplayNode
    Name teiburu4-10
    IP 10.0.8.14
    Monitors 2 (0,0) (0,1)

DisplayNode
    Name teiburu5-10
    IP 10.0.8.15
    Monitors 1 (3,0)
```

Figure 11. Example of a tiled display configuration file. This file starts with some information about the tiled display. “Dimensions” keyword describes the number of columns and rows of the tiled display. “Mullions” keyword is followed by the vertical and horizontal mullions measured in inches. “Resolution” is the screen resolution of each tile. “PPI” is the pixels per inch of each tile. “Machines” is the number of display nodes which drive the tiled display for each “DisplayNode”. “DisplayNode” represents a machine that can have an arbitrary number of screens. Each screen has a mapping described by the tiled column and row within the entire display. “Name” describes the host name of the node. “IP” is the IP address of the node. “Monitors” describes the mappings of each screen attached to that node with (0,0) located in the lower left.

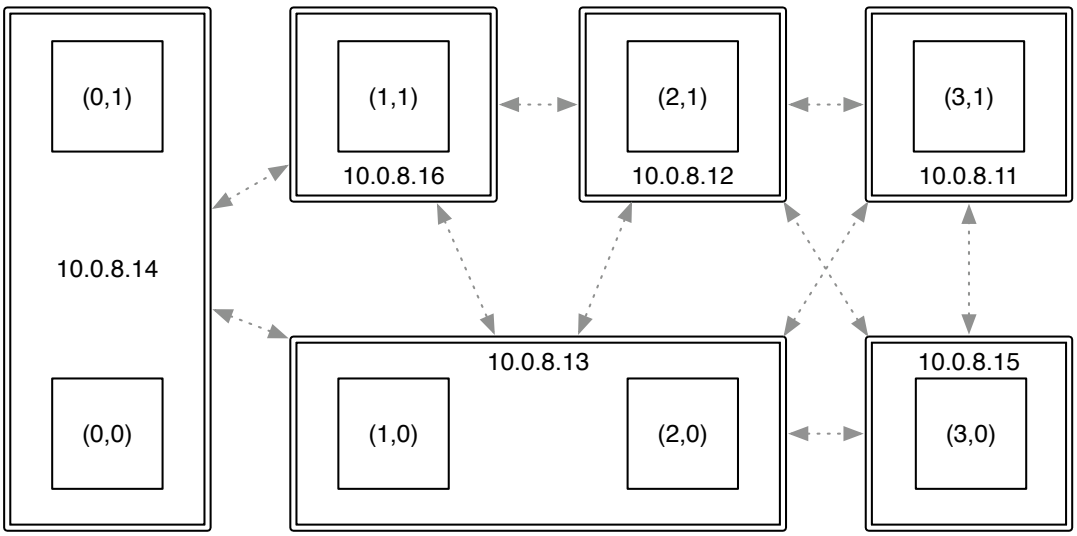


Figure 12. Tiled display for the tile configuration file described in figure 11. This figure also shows the communication setup for 6 visualization nodes of irregular physical layout.

The diagram in Figure 13 shows an example architecture for a system comprised of 4 nodes.

In terms of communication, each node:

1. Acts as a server that communicates outgoing simulation data to adjacent nodes.
2. Acts as a client that listens for incoming simulation data from adjacent nodes.
3. Acts as a client that communicates with the master node server to synchronize.

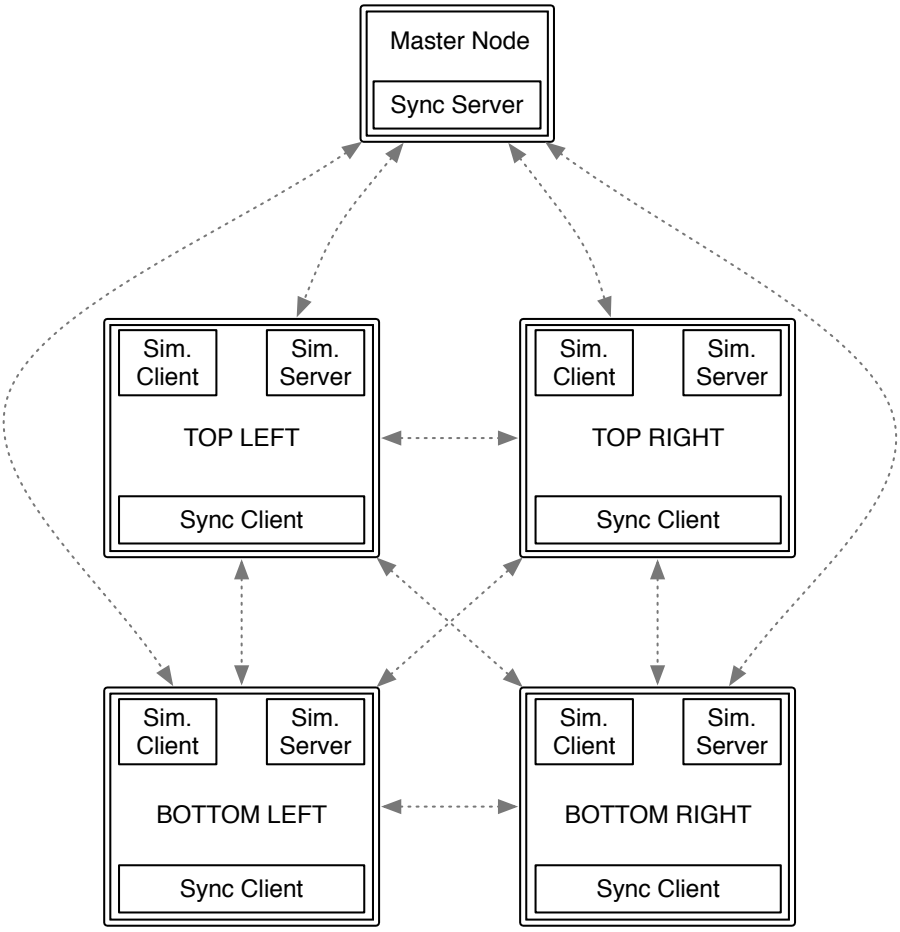


Figure 13. 4-node system architecture.

3.4.2 Simulation on a Grid

The simplest way to avoid $O(N^2)$ run time is to partition the simulation space into a static regular grid of spatial bins. Other more efficient dynamic methods can be employed, however local simulation methodology is out of the scope of this project. Each bin of the static grid contains particles currently in its field of view. When a particle moves from one bin to another, it is removed from its old bin and put into the new bin. In this setup, neighborhood lookups are simply a search in the particle's own bin and the neighboring bins. A simulation grid is local to each node. Its extent is slightly larger than the renderable extent of a node. This is necessary to ac-

commodate for remotely located particles in N-body type simulations. Figure 14 shows a particle and its field of view of the neighbors. Figure 15 shows approximate relative sizes of the rendering extent and the simulation grid. Each particle has a size, color, type, position, and velocity.

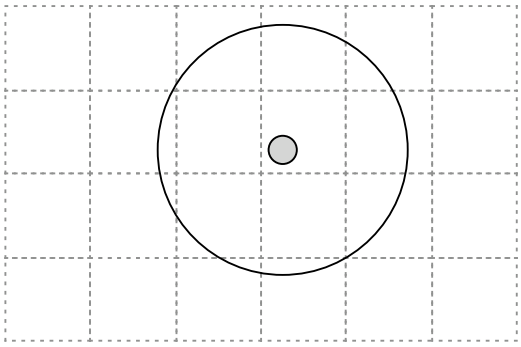


Figure 14. Particle neighborhood.

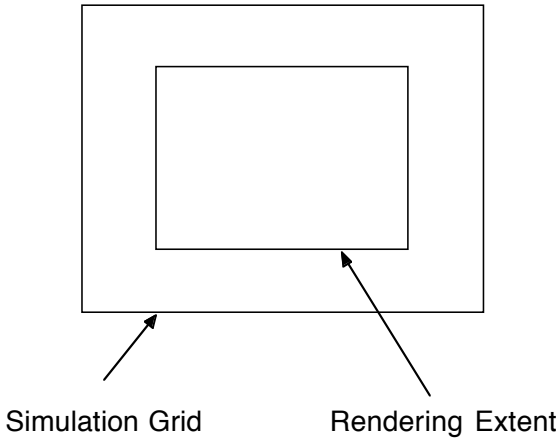


Figure 15. Relative sizes of rendering extent and simulation grid.

3.4.3 Screen Space

The particle simulation runs in the screen spaces of the tiled display. In order to effectively couple the visualization and computation procedures on the entire cluster, all nodes have to be well balanced. Load balancing will be discussed in a separate section in detail.

From the point of view of one node, the local procedures as related to simulation and visualization pipelines are (Figure 16):

1. Generate simulation input. This input is the primary source of particles. It is controlled entirely by the user.

2. Perform calculations and data access for the simulation input that was generated locally and needs to be rendered locally, forward it to the local visualization pipeline.

3. Perform calculations and data access for simulation input that was generated remotely but needs to be rendered locally, forward it to the local visualization pipeline.

4. Do load balancing. Perform calculations (data is forwarded from the origin node) for simulated input from any source that needs to be rendered remotely, send the results back.

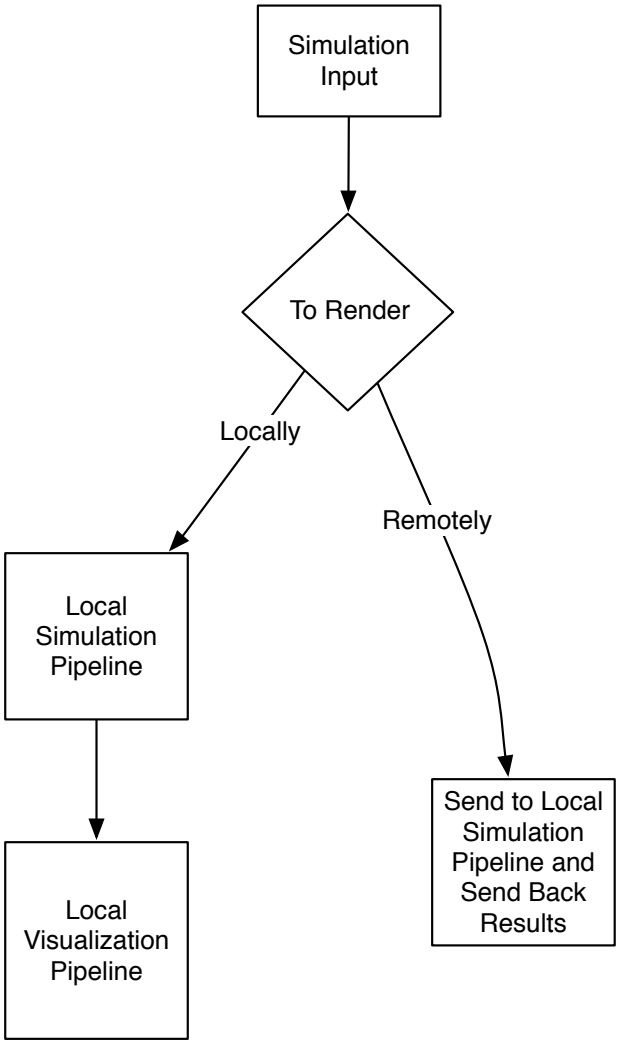


Figure 16. Local procedures. This figure shows an overview of the local processes taking place from the time a particle is generated until until it is visualized.

The local simulation pipeline is shown in detail in Figure 17.

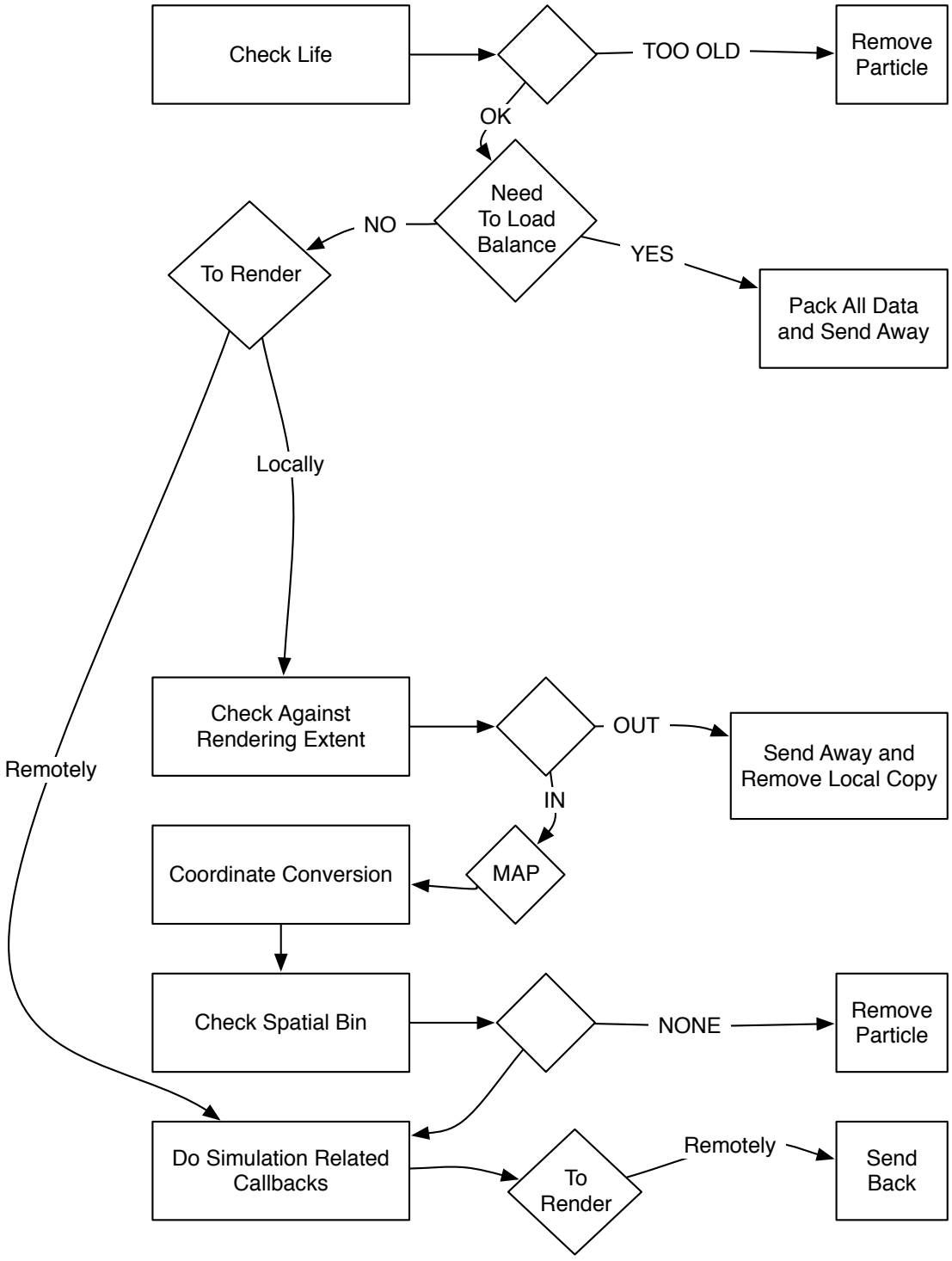


Figure 17. Local simulation pipeline. This figure shows stages during a particle's life cycle as it moves through the components of the pipeline on a local node.

3.4.4 Simulation Zones

In order to run N-body type simulations across borders of tiled displays, it is necessary to define regions of space close to borders. These regions or zones are used to map remote particles to local simulation grids and map local particles to a remote simulation grids. This is an extra step in the simulation pipeline when checking for a particle's intersection with the rendering extent. There are three major zones. The first zone is the local zone where no mapping is necessary. The second zone is a zone that forwards information about its particles to appropriate adjacent nodes. These particles are sent to adjacent node's "ghost" zone, which contains information about particles that are not local to the adjacent node located outside the node's rendering extent. This way, given proper synchronization, a neighborhood query close to a border on any node would produce correct results. The union of the local zone and the map zone correspond to a node's local rendering extent. The "ghost" zone is outside of a node's rendering extent. The zones are illustrated in figure 18. The map and ghost zones are further divided into sub-zones that follow the established 2-way connections to other nodes. Particles that end up in corner sub-zones, NW, NE, SW, and SE, are sent over to multiple nodes. For example, for a node that shares a border with other nodes on the north, west, and northwest sides, a particle in zone NW would need to be mapped to the north, and the northwest node. The figure illustrates two points of view, inbound and outbound, as nodes receive remote incoming mappings and send out local ones. When particles "die" locally or get removed when they jump outside of the map and local zones, messages have to be sent to appropriate nodes to indicate that these particles are no longer mapped. When moving between sub-zones, this procedure also has to check if multiple messages need to be sent out when particles move outside of the corner zones.

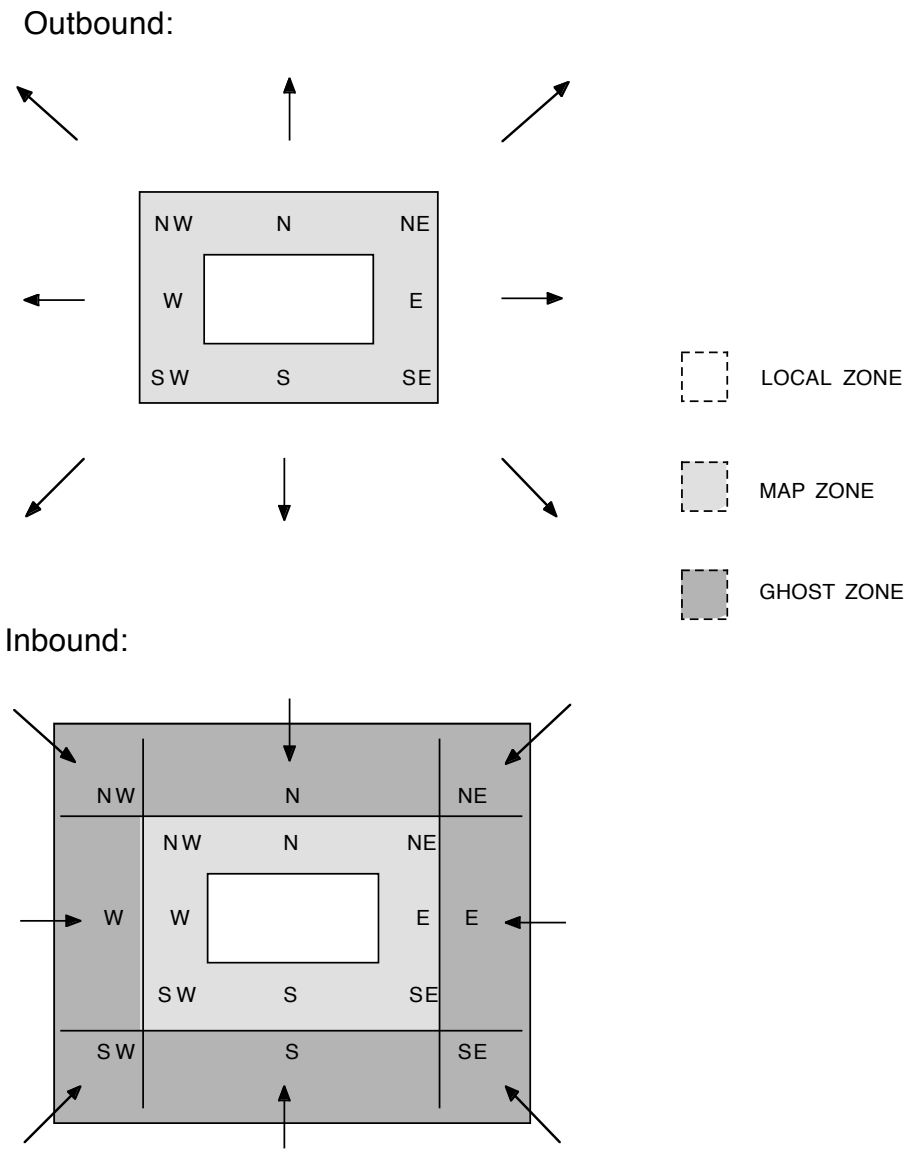


Figure 18. Simulation zones. This figure shows outbound and inbound views.

3.4.5 Visualization Pipeline

The local particles are drawn into a texture buffer using OpenGL's framebuffer object extension. Each particle is drawn as a point splat of certain size, color, and texture. In OpenGL, the splats are drawn as point sprites using GL_COORD_REPLACE_ARB extension that generates the correct texture coordinates for the vertex processor. A vertex program is used to correctly as-

sign point sizes and types from attribute arrays. The size of the particles is controlled by the number of particles hashed to a simulation grid cell. Larger sizes represent larger volumes. Information contained in the attribute arrays is generated during the simulation step while new particles are introduced into the system at a set rate at the locations of table pucks. Particles are colored according to their type directly in a fragment program. The result texture is a blending of two frames: the current frame, and the previous. Smoothing the two rendering frames results in better perception of motion. The result is blurred using 5x5 kernel Gaussian blurring to render metalball-style blobs. For efficiency, this is a two pass process. First, the texture is blurred horizontally in the fragment program, then vertically. This is illustrated in Figure 19. Figure 20 has a detailed view of the visualization pipeline.

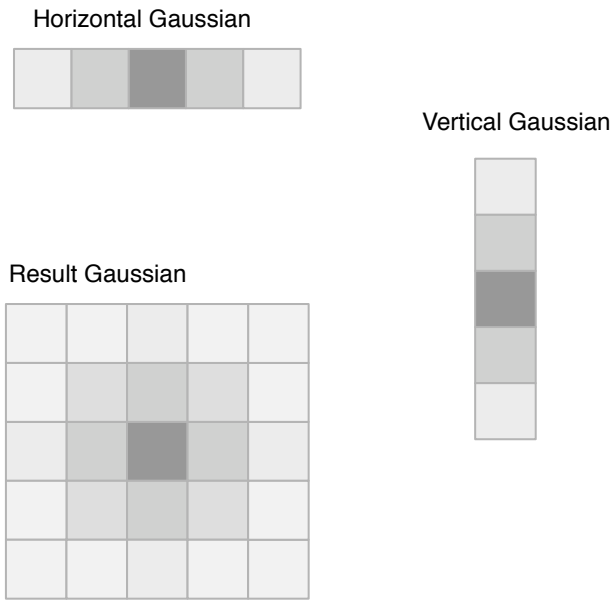


Figure 19. 2-pass Gaussian blur.

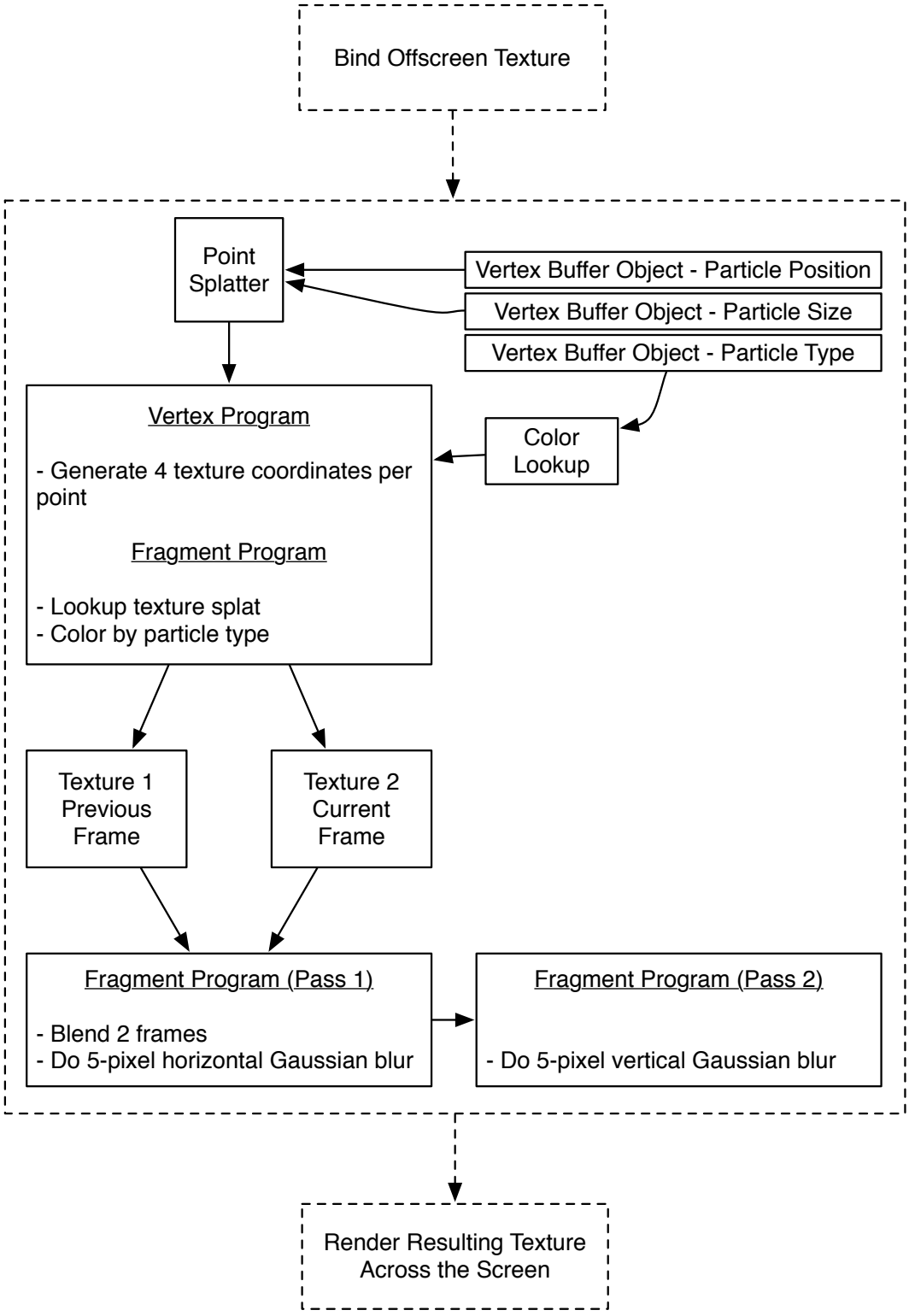


Figure 20. Visualization pipeline.

3.4.6 Load Management Mechanisms

Since particles are free to move anywhere in the space of interaction, it is possible for the system to become unbalanced. An example of this is when all of particles move to a single node. In this case, it is good to offload / distribute the computation if the simulation step is CPU-intensive. Communication is somewhat costly. If it is the case that both computation and communication have the same cost, increasing the number of computation nodes does not provide any performance benefit. Normally, the computation cost should be greater than the communication cost. Instead of using he entire cluster and doing a conventional broadcast at every simulation step, it would make sense to instead evaluate whether distribution is necessary. Say that each node can do calculations for N number of particles (normal computation plus load balancing computation). Then, we could say that if the number of particles grows larger than N/2, distribute the computation for N - N/2 particles. This would greatly minimize the communication cost for an under-loaded scenario. We can then define a discrete metric for each node to indicate whether it is under-loaded (< N/2), loaded (>N/2), or over-loaded (>N). If distribution is necessary, the particles can then be evenly distributed to under-loaded nodes. When interaction speed and system responsiveness is crucial, it is reasonable to employ other mechanisms to control the number of simulated particles to ensure a more balanced system. These mechanisms are hierarchical. On the lowest level, it is possible to limit the number of particles that can be present in a simulation grid bin. On the mid-level, it is possible to limit the number of particles that can be present on a node. On the highest level, the previous limitations control the number of particles that can be present on the entire cluster. On the lowest level, this method removes particles that are not really necessary for visualization and greatly decreases the local load. Another improvement is a good

balance between the rate at which particles are input into a node and the rate at which they exit. This is achieved by dynamically adjusting the input rate per node based on the rate particles exit a node.

To assess the possibility of offloading, it is necessary to define some metrics for communication and computation costs.

1. Computation cost =

(number of particles) * (computation time)

2. Communication cost =

(number of particles) * (size of data) + average latency

The communication bandwidth is usually good enough to transfer data. For example, in a 1Gbit local network, passing 1000 for offloading would mean passing a total of about (5 floats * 4 bytes * 1000) 20000 bytes.

In theory, this can be done 4000 times a second. However, theoretical bandwidth does not take into account the latency of packing the data, sending it across PCI bus to the wire, sending it from the wire though PCI bus, and unpacking it. It also does not take into account the overhead of TCP. These latencies depend on architecture, efficiency of the packing code, system drivers, and etc. Figure 21 lists round trip TCP latencies in milliseconds for a 1Gbit LAN using a frame size of 1500 byes for different packet sizes.

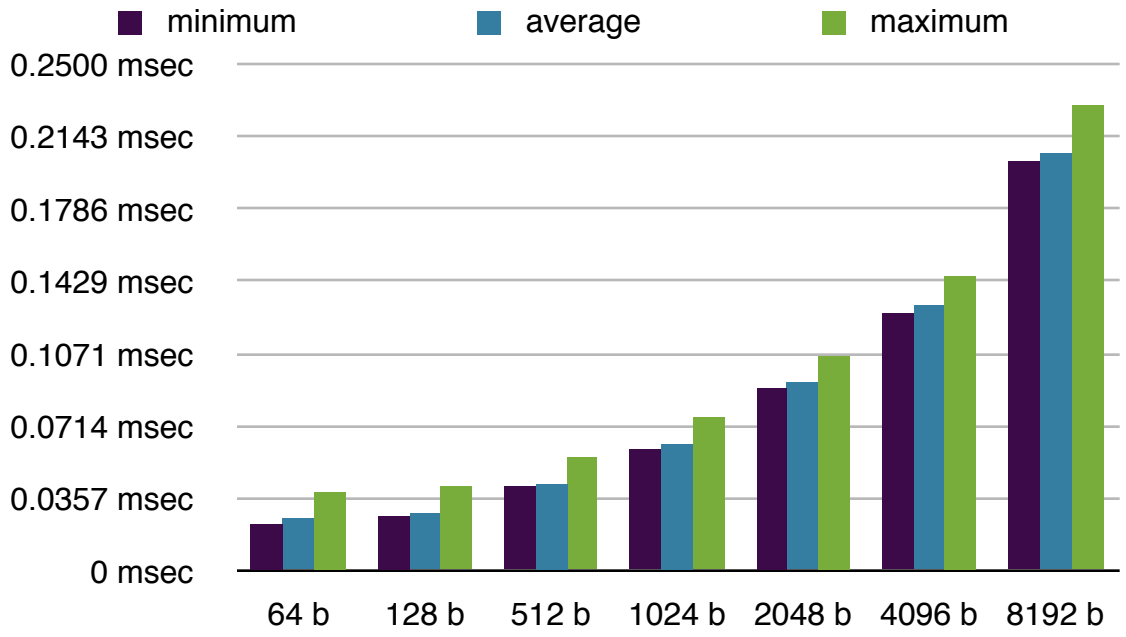


Figure 21. TCP round trip latencies for 1Gbit LAN.

If the communication cost is greater than the cost of computation, the computation is done locally. At the start up, an unused connection is established between each node, aka between each load balancing client and server, in the cluster. The metrics for each node are gathered by the master and distributed to the slaves. The slaves choose under-loaded nodes and distribute the computation evenly among them. This is illustrated in figure 22 (top). Alternatively, these load balancing servers may run on machines completely separate from the tiled display, however this is not implemented in this project because of the interest in avoiding the use of outside resources. The load balancing procedure does not assume that the data and the information to be gathered from the simulation grid is located on the load balancing server. This information is forwarded from the client to the computation server. This minimizes data access on the remote machine while taking advantage of data caching (described in the next section). It is not guaranteed that

offloading particles to a remote machine for computation would always increase performance. The chart in figure 22 (bottom) shows that performance is gained only for simulations that require longer to complete one simulation step (measured in nanoseconds). For this reason, offloading should be avoided for simple simulations because the latency of communicating data may be longer than the amount of time it takes to complete the simulation locally. Experiments on the LambdaTable suggest that offloading should be employed for simulations or subprocesses that take a significant amount of time to compute and, ideally, do not require immediate visualization. As the number of nodes increases, there is an optimal number of nodes that can provide performance benefit. This number would vary for different types of simulations due to the varied amount of information that is required to be transfered (based on the simulation type). However, when the number of tiled display nodes is low, local load management methods are more robust.

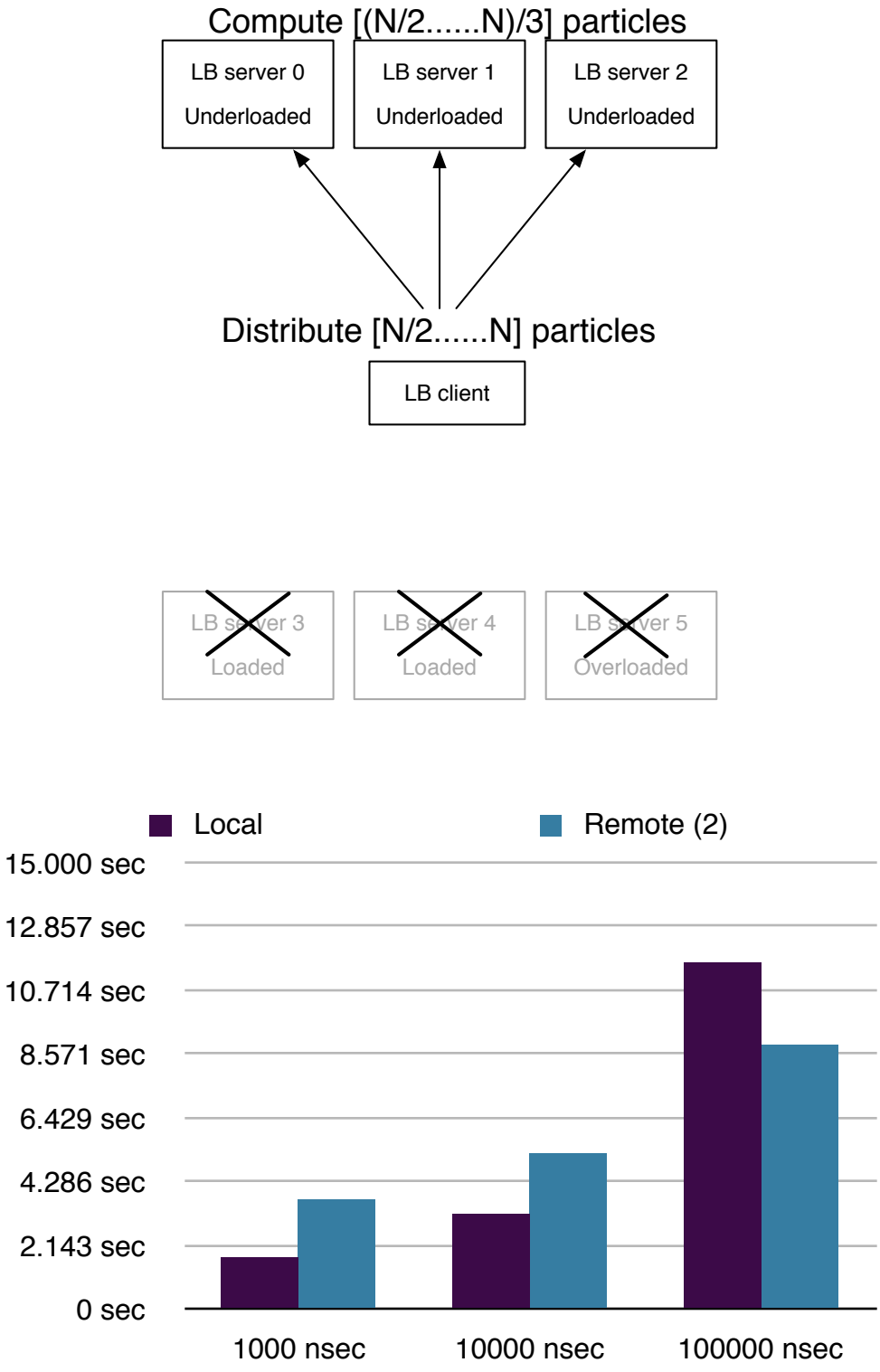


Figure 22. Diagram of offloading (top). Offloading 100 particles of varied computation time on LambdaTable, consisting of 3 nodes (bottom).

3.4.7 Data Caching

Similar to caching of image blocks in high resolution image rendering framework, the data required for simulation is cached locally. During a simulation, each particle accesses some data that is juxtaposed on the map of an area rendered by the image framework. This data access operation is a lookup of a single cell of data. This cell is cached locally if it is not already loaded. The frequency of cache misses is directly proportional to the amount of simulated input introduced into areas not already covered by any particles.

3.4.8 Multithreading

In order to maximize concurrency, the visualization and simulation pipelines run in a separate threads. Every time the simulation code is done with one update, it notifies the visualization code that new simulation data is available. This prevents the visualization pipeline from busy waiting. In addition, the generation of simulation input is done on a separate thread placing it randomly around the location of a user puck.

3.5 Synchronization

Synchronization is a achieved using constant communication with the master node. This is a centralized synchronization model. The synchronization server broadcasts a message that the simulation is ready to start and blocks until it hears back from all nodes that one simulation step is done. On the client size, the simulation pipeline is blocked until it hears for a message to start. Once the client is finished, it sends a proper message back. The procedure for synchronizing the visualization pipeline is similar, but it is done on a separate thread to ensure concurrency. In this

synchronization model, the frequency of simulation update and rendering update can be adjusted separately to meet system requirements. Figure 23 goes over rough synchronization commands.

Sync Server	Sync Client	
broadcast(Start_Simulation) blockingRecvAll(Stop_Simulation) broadcast(New_SimulationState)	blockingRecv(Start_Simulation) Do local simulation Communicate to resolve simulation state send(Stop_Simulation)	THREAD 1
broadcast(Start_Visualization) blockingRecvAll(Stop_Visualization)	nonBlockingRecv(New_SimulationState) Use new state if available blockingRecv(Start_Visualization) Draw local simulation data send(Stop_Visualization)	THREAD 2

Figure 23. Synchronization.

3.6 Scalability

This section presents an analysis of the scalability of this architecture.

Data size:

- Can be decentralized using PVFS file system, data is accessed only for cache misses.

Output resolution:

- Limited only by the fill rate of the video card on each node and the transfer bus (AGP/PCI/ PCI-Express).

Communication:

- Limited by bandwidth, communication increases for unbalanced system, stays consistent for a balanced system.

Computation:

- All computation is done locally, unless the load is too heavy, employs load balancing mechanisms that may either offload or decrease the load locally by removing particles that are not important to the visualization.

Interaction Space:

- Limited by scalability of the tracking system.

4 Interaction

This section goes over the interaction mode in Rain Table.

4.1 Basic Navigation

Rain Table lets users pan around and zoom into visualizations using trackable pucks. The pivot of zoom is determined by the location of the zoom puck on top of the table.

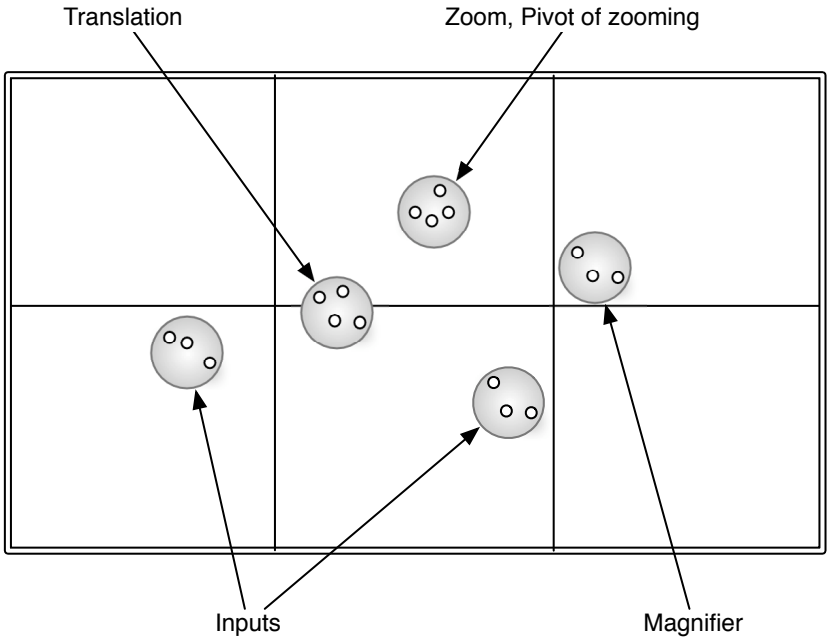


Figure 24. Pucks with unique retro-reflective markers.

4.2 Magnifiers

Magnifiers provide zoomed versions of areas under it. A single puck or multiple pucks can be assigned to be magnifiers. The magnifier also contains a version of the visualization under the magnifier. In order to make the imagery displayed by the magnifier more visible, the displayed visualization is a more transparent version of the original visualization. When a magnifier

crosses the border of local rendering extent, the magnifier typically displays data simulated and visualized by separate nodes. This is shown in figure 25.

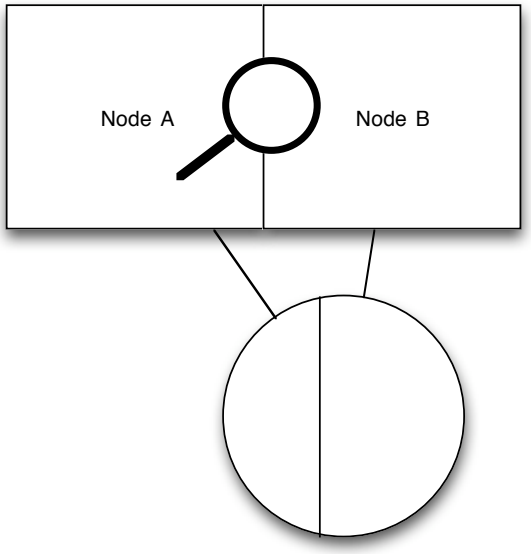


Figure 25. Magnifier on the border.

4.3 Inputs, Triggers, and Modifiers

In order to support a variety of visualizations, we define several widgets. Input widgets are considered to be the entry points of particles into simulations. They can an be radial or directed. Triggers are used to trigger events at certain areas, such as volcanos. Modifiers can be assigned to different types of parameters in simulations. For example, a modifier can be used to interactively control the rate at which particles enter simulations or the intensity of a volcano eruption. Figures 26 and 27 show the widgets described.

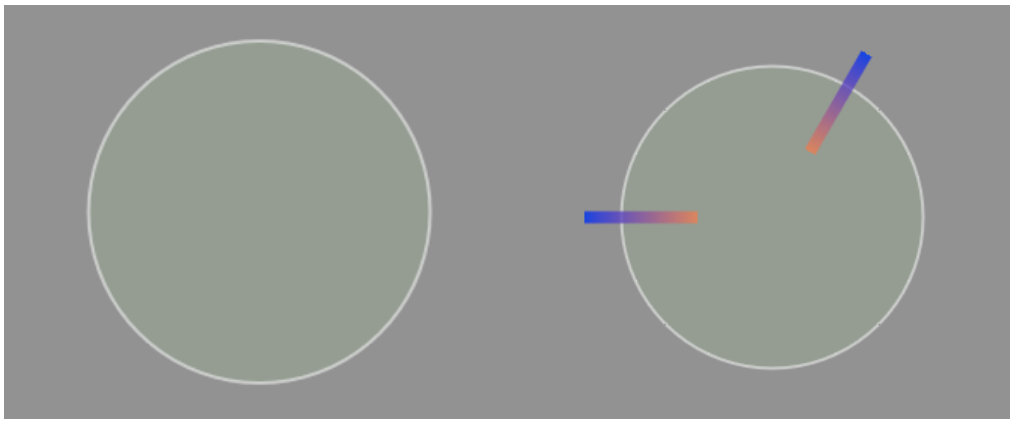


Figure 26. Radial and directed inputs.

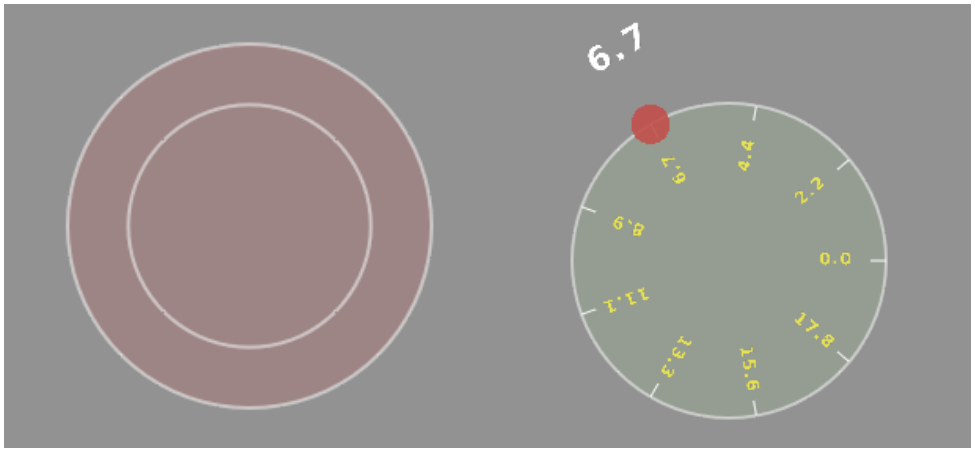


Figure 27. Trigger (right) and modifier (left).

5 Applications

This section goes over the applications implemented using the architecture and the preparation of data for them.

5.1 Flow Model Calculation

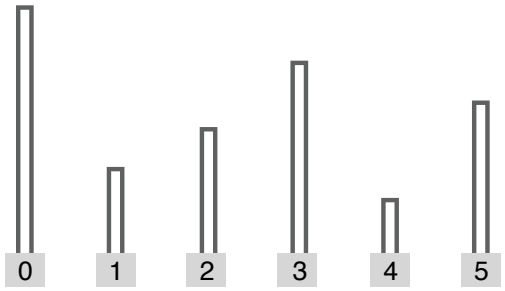


Figure 28. 1D elevation model.

In order to utilize the resolution of tiled displays, the data used to support simulations can be very large. Digital elevation models (DEMs) can be used to guide particles around a map juxtaposed with elevation data. However, the calculation of correct flow directly from large DEMs is not an option at run time due to a large number of natural depressions in DEMs. There is a number of GIS applications that provide the means to calculate flow models from digital elevation models. Some of this software is GRASS [20], ArcGIS [21], RiverTools [22], and LandSerf [23]. However, GRASS is the only known open-source application to provide this feature.

This project implements a single flow (SF) routing calculation on DEMs, however this calculation is not completely optimized for I/O and is meant to be a straight forward implementation of a well known algorithm [24]. The TerraFlow [25] project implements an I/O efficient method to do this calculation on very large grids.

The flow model is calculated directly from elevation data. In a single flow model, any given location has only one possible flow direction, the steepest down-slope neighbor. In the multi flow (MF) model, flow directions can be assigned to any steepest down-slope neighbors. Multi flow models are more realistic, but they take significantly more time to calculate and require more storage space. Given a regular grid of elevations, we can think of a depression as a single cell or a group of grid cells that do not flow out anywhere. Looking at Figure 28, we can see that this one dimensional grid of elevations contains two major depressions. One includes cells 1 and 2 and the other includes cell 4. The spill cells, the cells where all of the given cells in a depression should go through, are cell 3 for the first depression and cell 5 for the second. Figure 29 shows the distinction between single cell and multi cell depressions on a 2D grid.

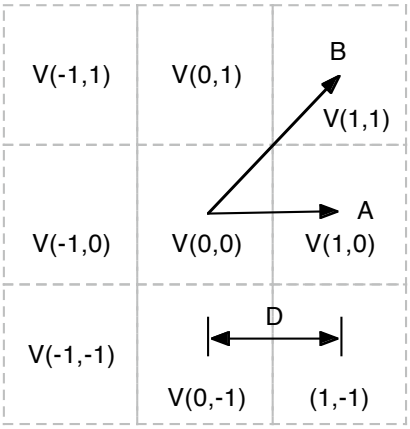
56	11	32
90	10	234
89	14	244

89	56	11	32	89
112	10	10	10	89
80	23	10	20	44
97	10	10	10	56
89	77	45	78	89

Figure 29. Singe-cell depression (left) and multi-cell depression (right).

The algorithm in Figure 31 goes over the calculation of flow models from DEMs. This algorithm is iterative. The algorithm makes multiple passes through the entire elevation grid. It identifies depressions and fills them in. To make sure that certain important areas do not get filled in, we introduce a depression threshold size. Small to medium sized depressions usually represent

insignificant features or errors in elevation created by the process used to record the data. Large depressions represent important geographical features and typically should not be filled in. After filling in all depressions that meet the threshold criteria, the algorithm calculates a vector of flow for each grid cell using the distance weighted drop. Figure 30 shows how to calculate this vector. This calculation uses V as the vector representing direction of the flow, D as the actual distance between the cells (depends on the resolution of elevation data), E_1 as the elevation at the target cell and E_0 as the elevation at origin cell. When all vectors are calculated, each cell vector is interpolated with the next grid cell's vector in the flow. This produces smoother flows for particle tracing routines.



$$A = V(1,1)\ (E_1 - E_0)\ D$$

$$B = V(1,0)\ (E_1 - E_0)\ D / \sqrt{2}$$

Figure 30. Vector calculation.

```
E = Regular grid of elevation values
N D = Number of depressions
N T = Number of depressions above threshold size
D THRES = Threshold cell size of a depression

While ( N D is not equal to N T )
Do
{
   N D = 0
   N T = 0
   Set the edges of E to flow out
   Find all single-cell depressions in E and fill them in
   L U = List of undefined flows
   For each cell in E
   {
     Calculate flow
     If the flow is undefined push it on to L U
   }
   For each cell in L U
   {
     Fill in undefined flows based on adjacent flows
     If flows back to the current cell, do not fill in
   }
   L D = List of multi-cell depressions
   For each cell in L U
   {
     D H = Highest bounding elevation
     Recursively examine cell's boundary
     {
       D L = List of connected cells of a depression
          If a neighbors value is greater than D H
          Then update D H with that value

          If a neighbor does not already belong to a depression
          Then assign it a unique depression ID and push it on to D L
     }
     Push each D L on to L D
     N D = N D + 1
```

```
    }
    For each depression in L D
    {
      If (its cell size is lower than D THRES )
      Then assign all cells the value of D H
       Else N T = N T + 1
    }
}
For each cell in E
{
  Calculate the flow vector based on its flow direction
}
For each cell in E
{
  Interpolate the cell's flow vector with its target cell's flow vector
}
```

Figure 31. Pseudo code for computing a flow model from DEMs.

5.2 Rainfall Runoff

This is a simple problem of particle tracing in a vector field calculated directly from a DEM. It requires minimal calculation since we are not interested in visualizing complex CFD, but the direction water takes on a map. The most time consuming step is the access of vector data at each step of the trace. The movement of a particle across vector field can be described by the following equations:

$$P_{T+\Delta T} = P_T + V_T \quad (4)$$

$$V_{T+\Delta T} = MAX\,(V_T + D\,(V_T - V_{T-\Delta T}), V_{MAX}) \quad (5)$$

$$L_{T+\Delta T} = L_T - \Delta T \quad (6)$$

where,

P is position vector

V is velocity vector

T is time

D is damping

L is particle life

5.3 Sediment Flux

Here, we model the movement and deposition of sediment in rivers and channels. The amount of sediment that can be picked up by water depends on its velocity and volume. We discretize this affect to describe three sizes of sediment that would be present in water: small, medium, and large sediment. The algorithm uses a lookup table to map particle's velocity and the volume of water in the local area to the sediment size that can be picked up. The deposition of sediment works in a similar way. Another lookup table maps a particle's (particle that is carrying some amount of sediment) velocity and the volume of water in the local area to the sediment size that may be deposited. In this model, large amounts of small sediment are always be present in water. On the visualization side, larger sediment is represented by darker colored brown particles contained within water.

5.4 Lava Flow

The flow of lava follows the topography similarly to the flow of water, however the physics are different. In order to properly simulate lava, a particle has to do a few neighborhood lookups. The task of modeling a simple flow of lava is to mimic insulated flow. In an insulated flow, a particle cools down faster if there are fewer particles around it. Let's develop a simple velocity

decay scheme. If the diameter of a visualized particle is approximately the size of a spatial bin on the simulation grid, then the velocity decay scheme can be approximated as:

$$V_{T+\Delta T} = MAX\left(V_T + D\left(V_T - V_{T-\Delta T}\right) - ID\left(1+E\right)^2, V_{MAX} \right) \qquad (7)$$

This equation is similar to particle tracing with the addition of insulated flow physics. In this equation, LD represents insulation damping and E is the number of non-empty spatial bins around a particle.

5.5 Pyroclastic Flow

Figure 32. Pyroclastic flow of Mayon Volcano, Pyilippines.

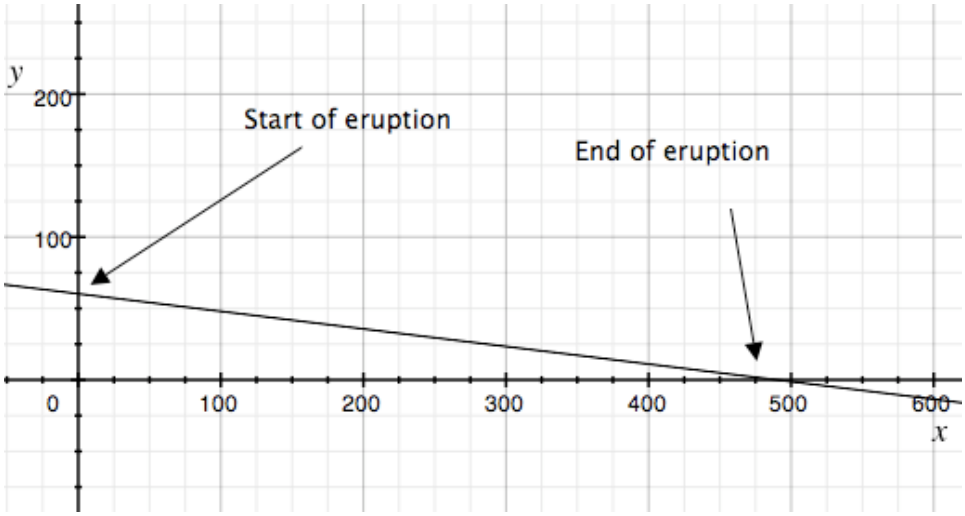


Figure 33. Eruption graph.

Pyroclastic flows are a result of volcanic eruptions (Figure 32). It is the movement of a combination of hot gas, ash, and rock, also known as tefra, around the volcano. The basic equation for computing the velocity of a particle in the flow is defined by (8). It is a simplified version of the plinian eruption model. The equation represents a linear relationship between and the velocity and the time since the initial eruption, where M is the magnitude of the eruption, D is damping, and T is the time. For larger eruptions, it is necessary to increase M and decrease D. This is because larger plinian columns tend to reach higher altitudes prior to collapse. Degassing of a larger volume flow should take longer than degassing of smaller flows. Figure 33 shows a graph of an eruption where M is 60.0 and D is 0.132. Equation (9) defines the relationship between stagnation height and particle velocity. Stagnation height is the accumulative height a particle has to overcome as it travels across terrain. If the velocity of a particle is less than the minimum velocity necessary to overcome this height, the flow is effected by the terrain, otherwise the flow conserves its velocity.

$$V_{T+\Delta T} = MAX\,(M - DT, V_{MAX}) \tag{8}$$

$$H_{STAG} = V_{MIN}^{\,2} \qquad V_{MIN} = \sqrt{H_{STAG}} \tag{9}$$

The relationship between the stagnation height and its relative minimum velocity is shown in Figure 34.

Stagnation Height	Minimum Velocity
20 meters	20 meters / sec.
80 meters	40 meters / sec.
5 meters	10 meters / sec.

Figure 34. Stagnation heights and their relative velocities.

In order to calculate the stagnation height correctly, it is necessary to keep track of the last lowest elevation the particle passed through. This is illustrated in figure 35. The last lowest elevation would be used to calculate the change in elevation. Then, in order to calculate the stagnation height, we would only need to know the change in horizontal position a particle has traveled. Equation (10) shows this calculation. Figure 36 goes over the algorithm to calculate and update stagnation height.

$$H_{STAG} = \sqrt{(\Delta E^{\,2} + \Delta D^{\,2})} \tag{10}$$

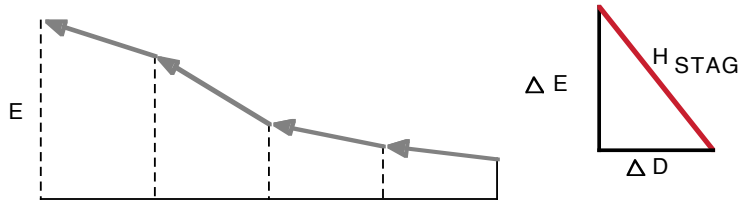


Figure 35. Stagnation height.

```
E p = Previous elevation
E C = Current elevation
D = Horizontal distance traveled

H = Change in height = E C - E p
DIST = Total distance traveled = sqrt( H2 + D2 )
MIN_VELOCITY = Stagnation velocity = sqrt(2 * g * DIST)

If MIN_VELOCITY > CURRENT_VELOCITY
  Take into account the topography
Else
  Velocity is conserved

If E p > E C
Then E p = E C
```

Figure 36. Algorithm to calculate stagnation height.

5.6 Glacial Movement

Figure 37. Glacier.

Of the fresh water on Earth, over 75% resides as ice in the ice caps and as alpine glaciers (Figure 37). Alpine glaciers are important topics of interest in the issue of global warming. The physics of glaciers are very complicated. When snow accumulates at the thickness of about 50 meters, the pressure is sufficient to turn snow into glacial ice. Ice, under atmospheric pressure, behaves as a brittle solid. Under enough pressure, ice will flow. Near the sides of the glacial channel, the ice flows slowest due to frictional forces. It is also possible for the entire glacier to slip when there is a build up of water under the glacier. This process is called basal slip and is analogous to ice-skating, where the pressure from the weight of the skater is focused into the blade o the skate. For visualization simplicity, we will model only the first type of movement. Glaciers form "U" shapes when moving down slopes. The following equation defines the velocity of a particle.

$$V_{T+\Delta T} = MAX\left(V_T + D\left(V_T - V_{T-\Delta T}\right) + GD\left(L + R\right), V_{MAX} \right) \qquad (11)$$

In this equation, GD is the glacial movement damping, L is the number of non-empty bins to the left of a particle, and R is the number of non-empty bins to the right of the a particle. This models the pressure distribution through the glacier. Particles that are in the middle typically move down faster than the ones on the edges forming "U" shapes. The following diagram (Figure 38) shows an example of velocity vector distribution.

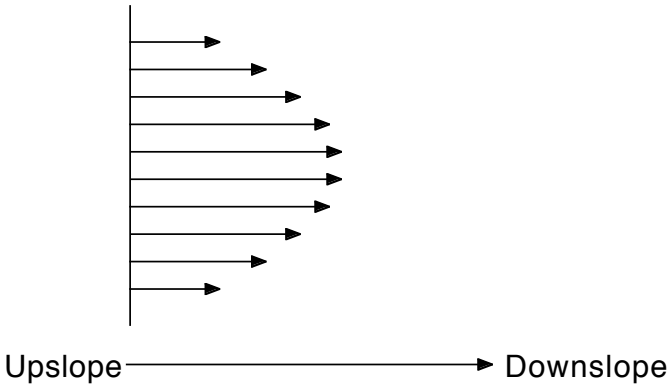


Figure 38. Velocity distribution.

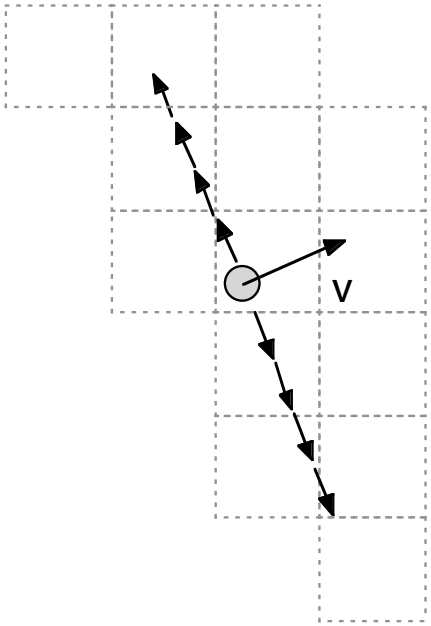


Figure 39. Neighborhood lookup.

To calculate the number of non-empty bins to the left and right of a particle, a ray is shot in both directions perpendicular to the particle's velocity vector. This is shown in Figure 39. These rays travel through bins and collect information. There is a maximum number of bins a ray can travel through to prevent this neighborhood query to travel outside of the node's local simulation grid.

5.7 Water Pollution

The implications of water pollution are very significant. The visualization of rainfall runoff shows how water connects different geographic regions. In order to visualize the pollution of water, we set up several controllable pollution spots. When rain originates at that spot, it is color coded according to the pollution origin's color. The result visualization makes it clear how pollutants travel with water, where they mix together, and where they end up.

6 Limitations and Future Work

This paper presents core software architecture for interactive simulations and visualizations using the hardware of tiled displays directly. With rapid development of high speed networks, the future work may include interaction and display of remote visualizations, interconnection of several remote tiled displays to visualize and interact with real-time simulations.

The current implementation runs in screen spaces of tiled display nodes and assumes that anything that moves outside of the entire display should be discarded. Future work would include routines to offload and update particles around the edges.

The load management mechanisms suggested in this work are not tuned to do well for a particular simulation. Future work may include adaptive methods to choose appropriate load management based on simulation requirements. Furthermore, the current system relies on manual adjustment of simulation parameters for different data. This can be improved by using more information about the data at run time, such as its current level of detail.

Some of the developments in the domain of GPGPU produced impressive results in the area of interactive simulations. The parallelism in the fragment processing of GPUs is used to simulate complex phenomena more efficiently. The use of GPUs for simulation, however, is limiting when the simulation is required to use large data. Frequent transfer of data between the GPU and CPU can create very significant bottlenecks. However, asynchronous methods can be employed to transfer data more efficiently. On demand paging of data for simulations to the GPU using textures as storage units would allow direct lookup into the data from fragment programs in screen space. This data can then be used for in a GPU-based particle system. When moving to scalable,

distributed design, such system would still constantly need to do readbacks from the GPU memory in order to synchronize the simulation state with adjacent nodes.

To reduce the frequency of cache misses, distributed memory and caching can be employed. This can greatly reduce remote disk access and speed up visualizations. At EVL, an application called LambdaRAM [26] is being developed to address access of massive data over high speed gigabit networks in data-intensive applications.

An infrastructure called OptiStore [27], also developed at EVL, can be employed in the future to efficiently create large data repositories for use in simulations and visualizations, which would make Rain Table software more robust and usable for real geo-scientists.

7 Conclusion

Scalable software systems are needed when moving to high resolution interactive environments. This paper presents a method to use high resolution tiled displays to support interactive group-oriented visualizations of real-time geoscience phenomena for science education in museums or classrooms.

The contribution of this work is:

1. Development of a decentralized particle-based simulation model that can be applied to many times of simulations.

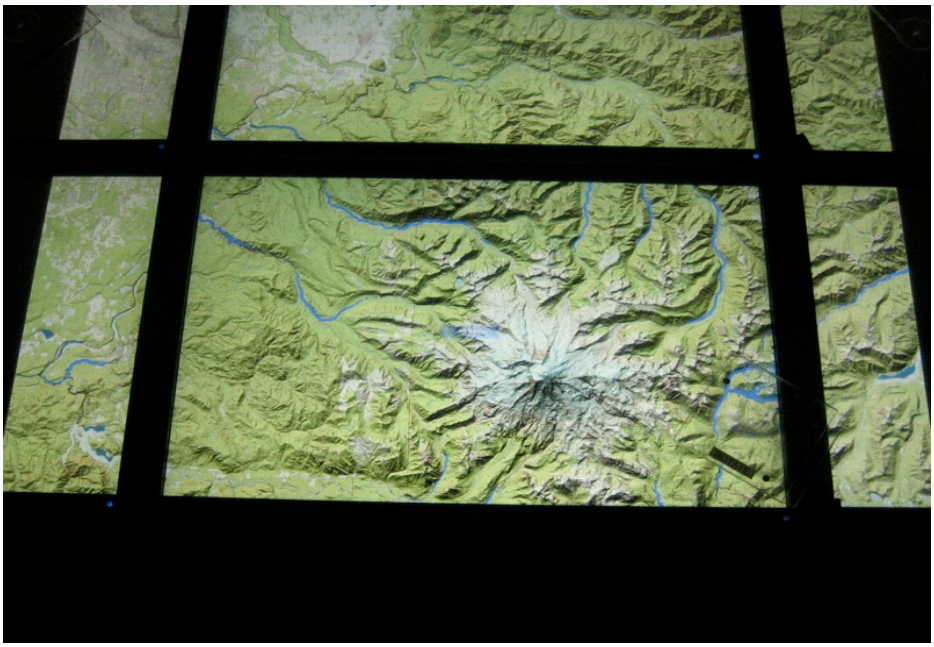
2. Development of an approach to efficient coupling of visualizations and particle-based simulations on high resolution tiled displays.

3. Application of traditional out-of-core and LOD methods to interactive high resolution environments.

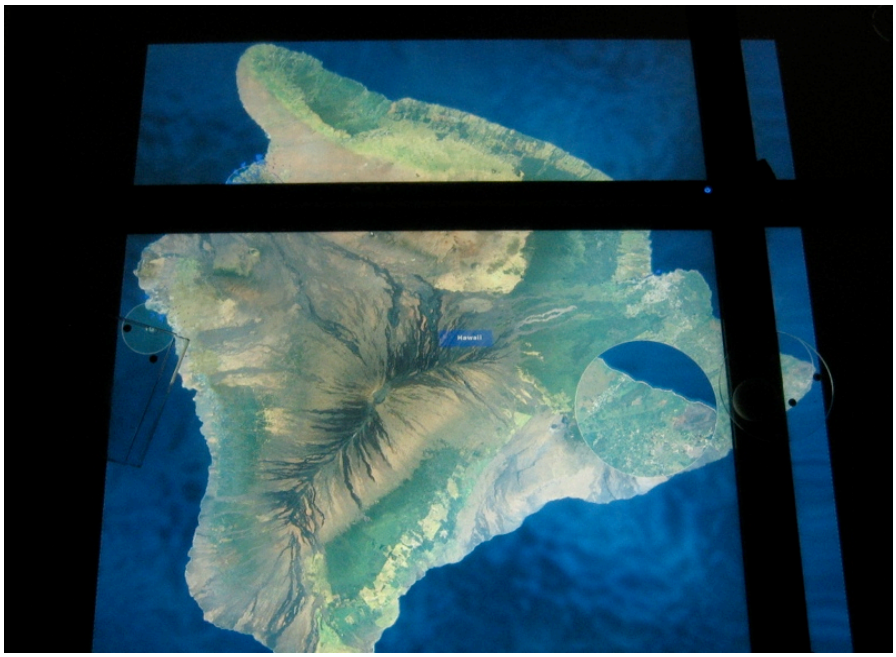
4. Application of visualization research technology to informal science education.

8 Current System

Rain Table software was demoed at SC (Supercomputing) 2007. It will be deployed at the Science Museum of Minnesota this summer. The current system includes a few datasets:

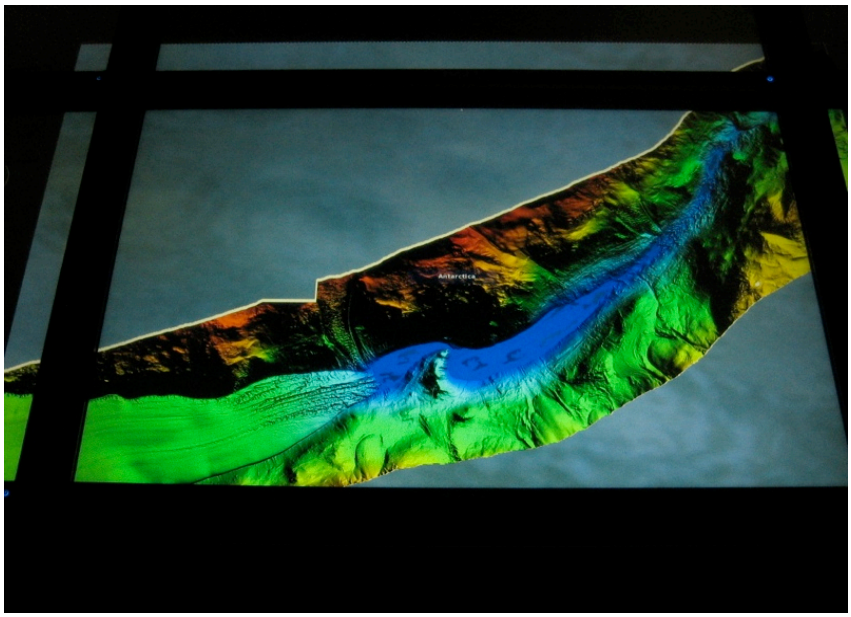


1. Mount Rainier National Park located 54 miles south east of Seattle, Washington. Mt. Rainier is an active volcano covered in 35 square miles of snow and glacial ice. Image pixel dimensions: 13,938 by 20,282, elevation data dimensions: 1,194 by 1,738.

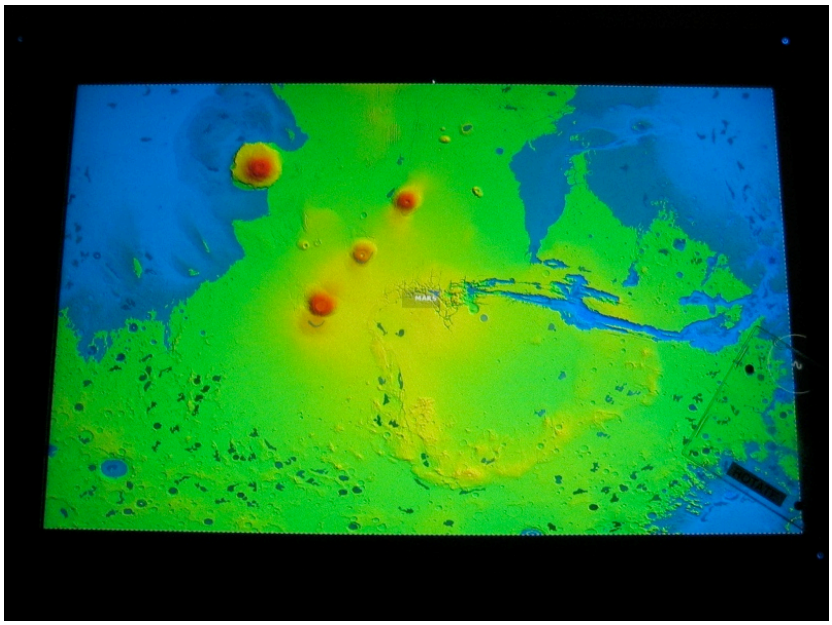


2. The Big Island of Hawaii. It is home to a few dozen volcanos including the world's largest volcano Mauna Loa. 40% of Mauna Loa's surface is covered by lava flows that are less than 1000 years old. Image (30 meter) pixel dimensions: 4,096 by 4,096, elevation data dimensions: 2,048 by 2,048.

3. Minnehaha Creek, Minneapolis, Minnesota. The area is one of the testbed sites of St. Anthony Falls Laboratory (National Center for Earth-Surface Dynamics). Image (0.5 meter) pixel dimensions: 8,821 by 5,032, elevation data dimensions: 1,198 by 488.



4. The ice covered Lake Bonney, Antarctica. Located in McMurdo Dry Valleys. Image pixel dimensions: 2,471 by 1,678, elevation data dimensions: 1,119 by 823.



5. Mars data collected using Mars Orbiter Laser Altimeter over 2 years beginning in fall 1997. Shows 4 large volcanoes. Olympus Mons (upper left) is 24 km high and 550 km in diameter. Image (4 meter) pixel dimensions: 9,027 by 5,599, elevation data dimensions: 2,256 by 1,399.

6. McMurdo station antarctica. The data was used to visualize drainage around the station to find problematic areas. Image (1 meter) pixel dimensions: 3,922 by 2,658, elevation data dimensions: 3,412 by 2,556.

9 Gallery

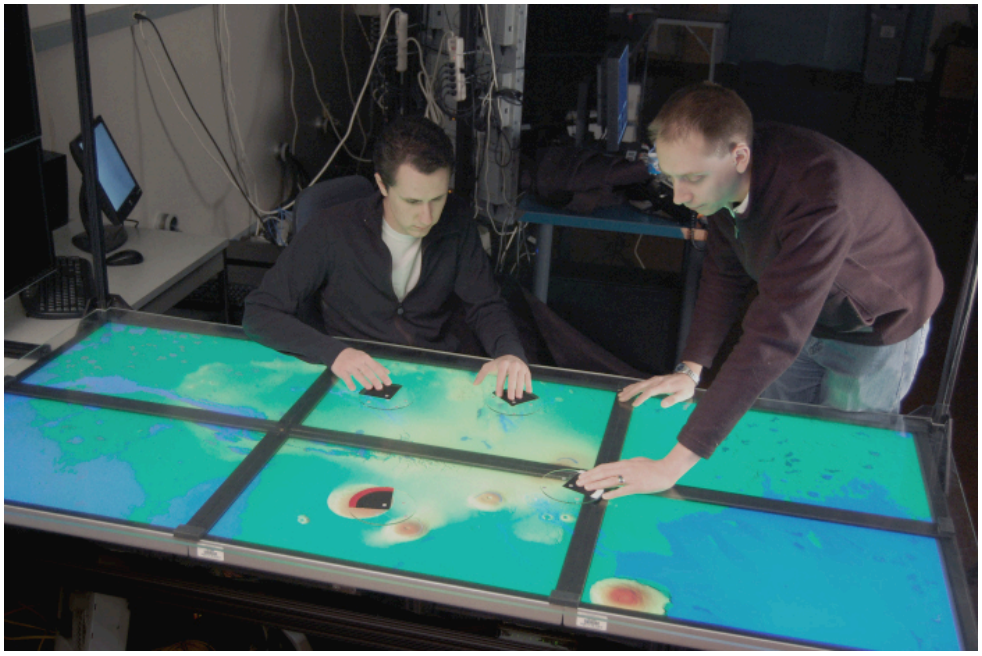


Figure 40. Users interacting with Mars data.

Figure 41. Users interacting with McMurdo data (Antarctica).

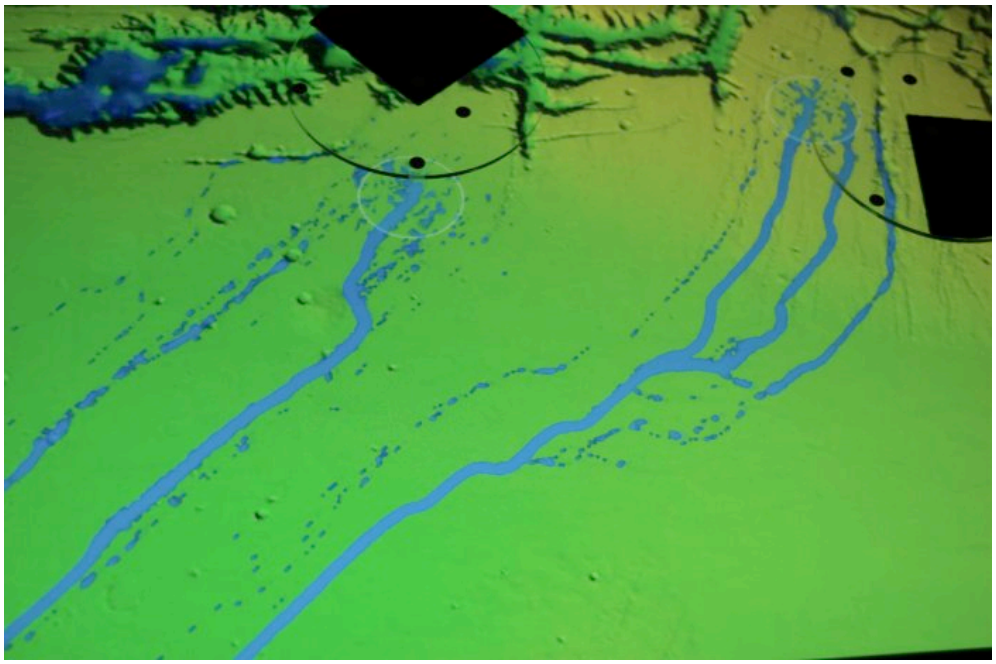


Figure 42. Rainfall Runoff (Mars data).

Figure 43. Sediment Flux (Hawaii Big Island data).

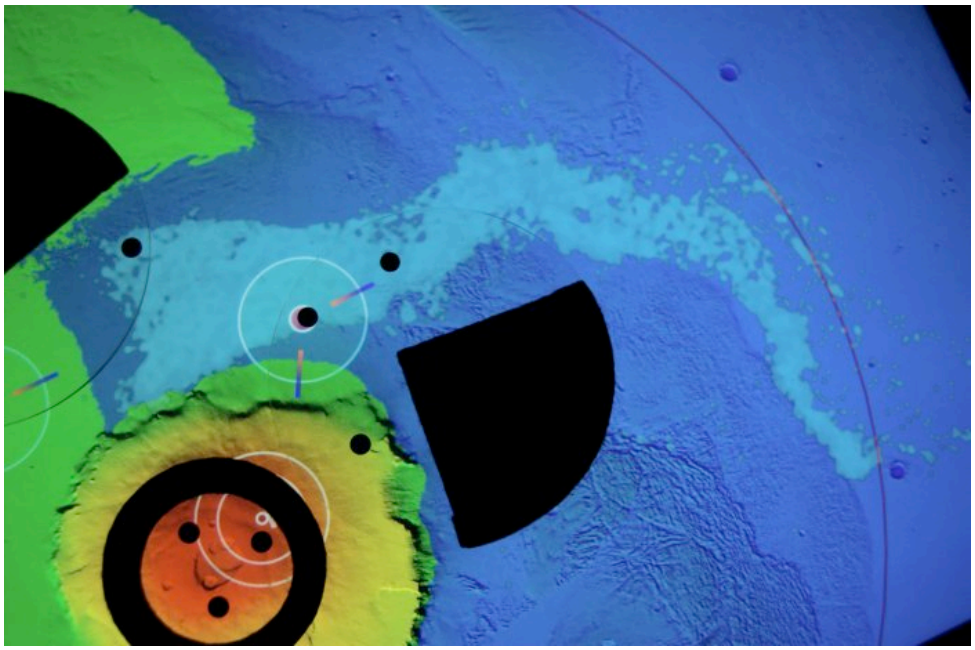


Figure 44. Pyroclastic Flow (Mars data).

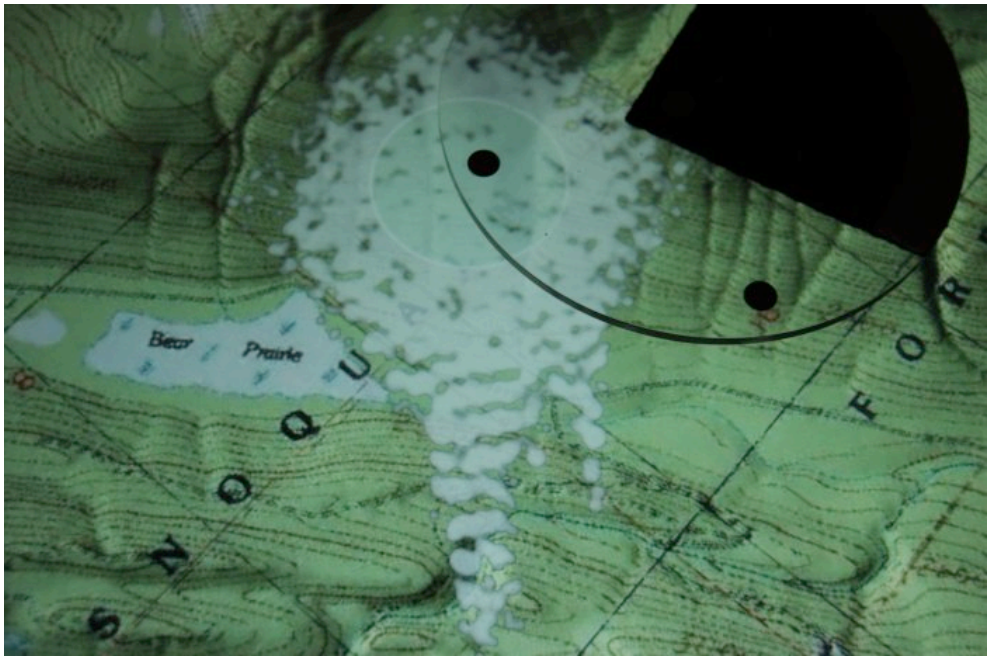


Figure 45. Glaciers (Mount Rainier National Park).

REFERENCES

[1] Krumbholz, C., Leigh, J., Johnson, A., Renambot, L., and Kooima, R.: "LambdaTable: High resolution tiled display table for interacting with large visualizations*,"* In *Workshop for Advanced Collaborative Environments (WACE)*. 2005.

[2] Chen, X. and Davis, J. "LumiPoint: Multi-User Laser-Based Interaction on Large Tiled Displays". In *Displays*, Vol. 23, pp. 205-212. 2002.

[3] Stodle, D. Hagen, T. Bjorndalen, J. Anshus, O. "Gesture-Based, Touch-Free Multi-User Gaming on Wall-Sized, High-Resolution Tiled Displays". In *Proceedings of the 4th International Symposium on Pervasive Gaming Applications*, PerGames 2007, pp. 72-83. 2007.

[4] Stodle, D. Hagen, T. Bjorndalen, J. Anshus, O. "A system for Hybrid Vision and Sound Based Interaction with Distal and Proximal Targets on Wall-Sized, High-Resolution Tiled Displays". In *CVHCI07*. pp. 59-68. 2007.

[5] Ball, R. and North, C. "Effects of Tiled High-Resolution Display on Basic Visualization and Navigation Tasks". In *Extended abstracts of ACM Conference on Human Factors in Computer Systems (HCI 2005)*, pp. 1196-1199. Portland, Oregon. 2005.

[6] Rogers, Y. and S. Lindley, "Collaborating around vertical and horizontal large interactive displays: which way is best?*"* *Interacting with Computers*, 2004. 16(6): p. 1133-1152.

[7] J. Allard , B. Raffin , F. Zara . "Coupling Parallel Simulation and Multi-display Visualization on a PC Cluster," In *Euro-par 2003,* Klagenfurt, Austria, August 2003.

[8] *Tiled Image Viewer (TimV).* San Diego Supercomputing Center.

[9] *Cluster-based Image Viewer*. Justin Blinns, Michael E. Papka, Rick Stevens. Argonne National Labs.

[10] N. K. Krishnaprasad et al, "JuxtaView – a tool for interactive visualization of large imagery on scalable tiled displays," In *Proceedings of IEEE Cluster*, 2004.

[11] Tom Geller. "Interactive Tabletop Exhibits in Museums and Galleries". *IEEE Computer Graphics and Applications,* 26(5): 6-11. 2006.

[12] Jorda, S. Geiger, G. Alonso, M. Kaltenbrunner, M. "The reacTable: Exloring the Synergy between Live Music Performance and Tangible Tabletop Interfaces". *Proceedings of the first international conference on "Tangible and Embedded Interaction" (TEI07)*. Baton Rouge, Louisiana.

[13] An exhibition facility themed around the Ogura Hyakunin Isshu (a classic anthology of 100 traditional Japanese poems from the 7th to 13th centuries composed by 100 poets) built in Arashiyama, Kyoto, in January 2006 and operated by the Ogura Hyakunin Isshu Cultural Foundation.

[14] FLTK - Fast Light Toolkit, (http://www.fltk.org).

[15] MPICH Implementation of the Message Passing Interface (MPI), (http://www-unix.mcs.anl.gov/mpi).

[16] E. He et al, "Quanta: a toolkit for high performance data delivery over photonic networks," *Journal of Future Generation Computer Systems*, volume 19, issue 6, pp. 919-933, August 2003.

[17] Brown, C. Squish DXT Compression Library, (http://www.sjbrown.co.uk/?code=squish).

[18] NFS - Networked File System (http://nfs.sourceforge.net)

[19] PVFS - Parallel Virtual File System (http://www.parl.clemson.edu/pvfs)

[20] GRASS GIS - Geographic Resource Analysis Support System, (http://grass.itc.it).

[21] ArcGIS, (http://www.esri.com/software/arcgis).

[22] RiverTools, (http://www.rivertools.com).

[23] LandSerf, (http://www.soi.city.ac.uk/~jwo/landserf).

[24] Jenson, S. and Dominique, J. "Extracting topographic structure from digital elevation data for geographic information system analysis". *Photogrammetric Engineering and Remote Sensing*, 54(1), pp. 1593-1600. 1988.

[25] Laura Toma, Rajiv Wickremesinghe, Lars Arge, Jeffrey S. Chase, Jeffrey Scott Vitter, Patrick N. Halpin, and Dean Urban. "Flow computation on massive grids". In *Proc. ACM Symposium on Advances in Geographic Information Systems*, 2001

[26] Vishwanath, V., Shimizu, T., Takizawa, M., Obana, K., Leigh, J. "Towards Terabit/s Systems: Performance Evaluation of Multi-Rail Systems". *Proceedings of Supercomputing 2007 (SC 2007)*. Reno, NV. 2007.

[27] Zhang, C. "OptiStore: An On-Demand Data Processing Middleware for Very Large Scale Interactive Visualization". Thesis. 2007.