Supercomputing visualization made simple

By

Huy Nguyen

B.A, Mathematics and Computation

Oxford University, 2005

Submitted to Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Feb 2008

Author.....

Department of Electrical Engineering and Computer Science

Feb 2008

Certified by .....

Alan Edelman

Professor of Applied Mathematics

Thesis advisor

Accepted by.....

Professor Terry P. Orlando

Chair, Department Committee on Graduate Students

Abstract

In this thesis, we propose a solution for remote visualization for supercomputers. Our solution consists of two tools that help users visualize data from high performance computers. The first one takes advantage of the Web and AJAX technology [25], is simple, light weight and does not require any pre-installation which can be a perfect tool for demonstration supercomputing data. The second tool, a 3D Viewer on MATLAB Star-P [8], is to utilize more resources in the user's workstation to achieve better quality visualization and more flexibility in data navigation and analysis. Both solutions strive to create a simple and user-friendly framework that supports researchers' goals to create, analyze, test and debug numerical algorithms in supercomputing world.

Thesis supervisor: Alan Edelman
Title: Professor of Applied Mathematics

Acknowledgment

I would like to sincerely thank my advisor Prof. Alan Edelman. He first introduced to me the concept of supercomputing visualization and inspired me to conduct my own exploration in this area. Without his guidance, stimulating support and great suggestions, this thesis could not have been completed. This project was funded by Alan's Research Assistantship.

I'm also deeply grateful to Prof. Bill Roscoe and Dr. Michael Collins for their valuable supervision and encouragement during my undergraduate studies at Oxford University; to Steve Leibman, Ron Choy, Viral Shah, Raghunathan Sudarshan and the Star-P team for their constructive feedbacks, which helps refine the content and structure of the project.

I also would like to thank Interactive Supercomputing and Mathworks, for creating excellent products, MATLAB and Star-P, on which my solution was built.

Lastly, I would like to give my special thanks to my family, my father Nguyen Ngoc Dung, my mother Nguyen Thuc An, my brother Nguyen Ngoc Hung and my girlfriend Bui Phuong Anh, whose love, patience, and continued support has fuelled me the lasting energy to finish this project.

Contents

Figures

Chapter 1
Introduction

Scientific visualization has been long known as an efficient tool to assist researchers to achieve better and deeper understanding of objects. However, such an application on supercomputing data faces many difficulties due to the huge size of data, issues of parallel processing, processes communication, shared memories etc. In this project, we hope to build a simple solution for that problem which can help researchers visualize supercomputing data on a server from a commodity workstation. In addition, as the tool is built on top of the Star-P system, it automatically inherits the simple and easy-to-use approach of Star-P, allowing users to visualize data on popular environments like the Web or MATLAB. In combination with the Star-P system, this visualization tool can be used as a tool to assist researchers in designing, debugging and testing numerical algorithms.

Related works:

Tuchman *et al.* [17] presented a 2D-matrix-visualization with features such as panning, zooming, number coloring to enhance visualization and remote visualization. They also showed an example of applying it to numerical algorithm design. However, due to the lack of hardware and software at the time, this system was rather limited in many ways including matrix size, graphic quality and network usage, etc. The mathematical tool, gCluto[21], for analyzing and clustering data, has matrix visualization in two modes: *mountain view* and *tree view*. While the

mountain view mode shares similar features with our 3D Visualization (Chapter 4), the tree view presents a clustered matrix in an intuitive hierarchy structure. However, this tool is solely compatible with local matrices, thus limiting its use to small data analysis.

Chen *et al.* [18] adopted a novel approach for general remote visualization using structured, pre-rendered imagery. This approach even allows users to have 3D Visualization on the Web using the industrial standard QuickTime VR Movies. However, as this approach requires users to download the movies *before* they can start visualizing, it is not real time and wastes huge amounts of resources when users just want to view a small part of the object. VNC [1] is another general purpose approach that can be used for visualization. This method seeks to build a local visualization system *on the server side,* and transfer screenshots of the server to the clients using VNC technology. However, even if the network bandwidth and the download speed are unlimited, this method still suffers from the latency effect. Visapult [5][27] and VisIt [4] are other general purpose visualization systems which are designed to optimizing graphic, network consumption and guaranteeing interactive frame rate. However, as they both require a lot of server power for rendering, only a limited number of users can access visualization concurrently. In addition, to use these applications, user data must be converted to compatible formats before visualizing. This step is inconvenient and cumbersome if the data is repeatedly updated such as in algorithm debugging.

Taking into account various limitations of the applications listed above, our system not only supports remote matrix visualization with high graphic quality but also guarantees a real time interactive frame rate. Furthermore, importing data is not required in our system as it is built on top of the Star-P system which already hosts the data.

The general design of the system will be described first in Chapter 2. In Chapter 3, we will discuss about the 2D Visualization in Web. Chapter 4 will present in greater details the design and implementation of our 3D Visualization on MATLAB Star-P. Future work and conclusion will be discussed in the last chapter.

Chapter 2
General analysis and design

Our goal is to build a system that not only allows researchers to visualize matrices that reside on supercomputers but also guarantees an interactive frame rate in unstable network conditions. Therefore, in our design, we strive to achieve the following objectives:

1. The matrix size can be as large as the supercomputer can afford. The system should be able to work with hundreds of Gigabyte matrices or more in the future.
2. The supercomputing server maybe expensive and shared among a lot of users. We should take into account the computation cost of visualizing on the server side. In our system, we hope to keep the server computation complexity and space as low as possible.
3. Although rapid progress has been made in the area of supercomputing in both computation and storage, the communication technology has seen much more gradual changes. Current high speed Internet only allows network transfer speeds of gigabits per second. Due to this reason, the network is the bottle-neck of our remote visualization system and most of our work should be done to solve this problem.
4. Another communication problem, more subtle than network speed (megabytes/second), remains: latency. The typical latency needed for a packet to travel back and forth between client and server is hundreds of milliseconds. Hence, our application cannot

make a request to research for data for each visualizing frame, since if this happened, the frame rate could be dragged to as low as 5 frames per seconds (fps) which is unacceptable for an interactive application.

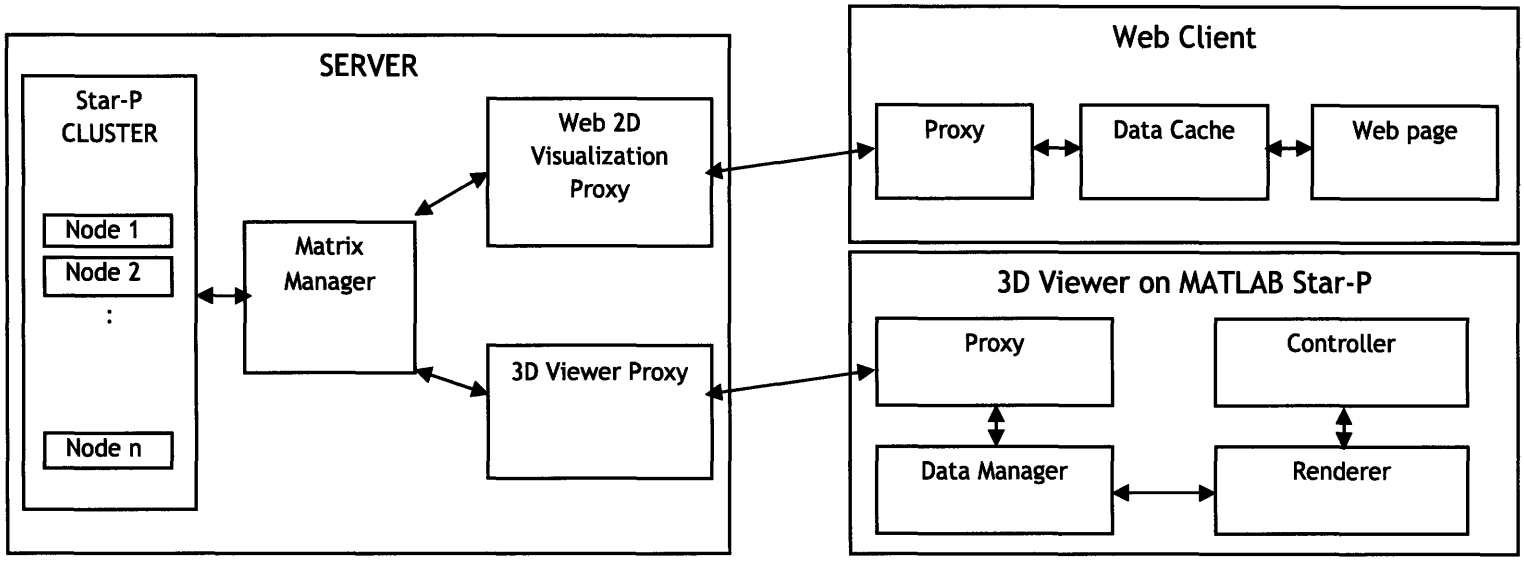


Figure 2 General architecture

System architecture

Clients

Currently, there are 2 types of clients in our system: the Web 2D Visualization and the 3D Viewer on Star-P. Building two types of clients in one system, we want to try different approaches to support mathematical researchers in supercomputing visualization. The first approach is based on the popularity and user-friendliness of the World Wide Web empowered by the recent development of AJAX technology [25]. In this approach, our goal is to build a visualization application that is reachable to billions of web-users around the world with minimal

effort. No installation or special skill is required. All users need is basic computer skills and a computer with Internet access. The second approach, a 3D Viewer on MATLAB Star-P aims to utilize more resources in the user's workstation to achieve better quality visualization and more flexibility in data navigation and analysis.

The protocol used for communication between clients and server is TCP/IP. More details about the techniques and clients design will be discussed in chapter 3 and 4.

Server:

On the Server side, Star-P is a distributed system that works as the computation and storage engine. On top of it, we build a Matrix Manager and server proxies. While the server proxies only implement communication protocols to correspond with clients, the Matrix Manager can be considered as the heart of our server; which is responsible to process all client queries forwarded from the proxies, talk to distributed processors in the clusters and gather requested information from them. The Matrix Manager works with all matrix distributions supported by Star-P including row distribution, column distribution for dense matrices and CRS for sparse matrices [10].

Thanks to this modular architecture and the general interface of the Matrix Manager, adding a new type of client in to this system becomes very easy; we only need to build the new client and a proxy on server, and plug them into our system.

Matrix Manager

The Matrix Manager is responsible for processing the client's queries forwarded from the visualization servers. It uses the Message Passing Interface (MPI) and the Star-P SDK [9] to gather information from different processors in the Star-P distributed system. Also, in the Matrix Manager, data is pre-processed to minimize query cost.

Data pre-processing in the Matrix Manager:

Queries from clients in our visualizing system are always in the following format: $f(x1, y1, x2, y2)$ where f is a *decomposable* function mapping the rectangular sub-block $(x1{:}x2, y1{:}y2)$ in the matrix to a real or complex value.

Definition: *Let f be a function well-defined on every rectangular block of a matrix A. For all blocks B in A and for all $P(B)$ which is a partition of B into smaller sub-blocks, f is decomposable on A if and only if the value of $f(B)$ can be retrieved from the set of values $f(P(B))$.*

Normally, without pre-processing, the computational complexity of any query is $O(n^2)$ where n is matrix size. However, when data is pre-processed, the query cost can be reduced to linear ($O(n)$) with linear storage requirement.

Basically, a pre-processing process is established to create a set of auxiliary matrices which contains pre-computed values of f at some chosen blocks. Then, when a query

$f(x1, y1, x2, y2)$ is received from client, it will be partitioned into these chosen blocks and pre-calculated values are then used to retrieve query result.

Create auxiliary matrices:

Fix a constant integer a. At level 1, partition our matrix into $a \times a$ equivalent blocks of size $\frac{n}{a}$. Recursively, at level k+1, blocks at level k are partitioned in to $a \times a$ equivalent blocks of size $\frac{n}{a^{k+1}}$. At the last level, the block size should be less than a. Then, for each level, an auxiliary matrix is created to store the pre-computed values of blocks in that level. Since we do this calculation in order of decreasing level, the total computation cost is only $O(n^2)$. The total space required for auxiliary matrix is $O((\frac{n}{a})^2)$. In practice, a is set to 16, thus, the total space required should be less than $1/100$ space of matrix.

Composing algorithm:

1. Find the largest auxiliary block that completely fits in a queried block. If there are more than one such blocks (of the same size), take them all. If no auxiliary block found, use blocks of size 1×1 instead.

2. If the blocks found in step 1 cover the whole queried block, combine their pre-calculated values and return the result.

3. Divide the remaining area into 4 sub-blocks, and recursively calculating f on these sub-blocks. Combined results of the recursive calls with the pre-calculated values of block found in step 1 and return the result.

Chapter 3
2D Visualization

Web technology

Nowadays, the Web is a standard in the content industry and is considered the best way to deliver information. Anything published on the web can be instantly accessed by billions of Internet users around the world. Therefore, there are more and more services, businesses built to leverage this technology such as office tools, games, mails, personal information managers and they are gradually replacing the role of traditional desktop software. Along with the Web's increasing popularity, web applications are also becoming much more sophisticated with the support of various technologies such as Javascript, Flash, PHP and AJAX. This sweeping development of web technology is the primary inspiration for our project.

In this chapter, we will present a 2D Visualization on the Web which enables users to view their matrices in 2D and navigate it by panning and zooming.

Web 2D Design

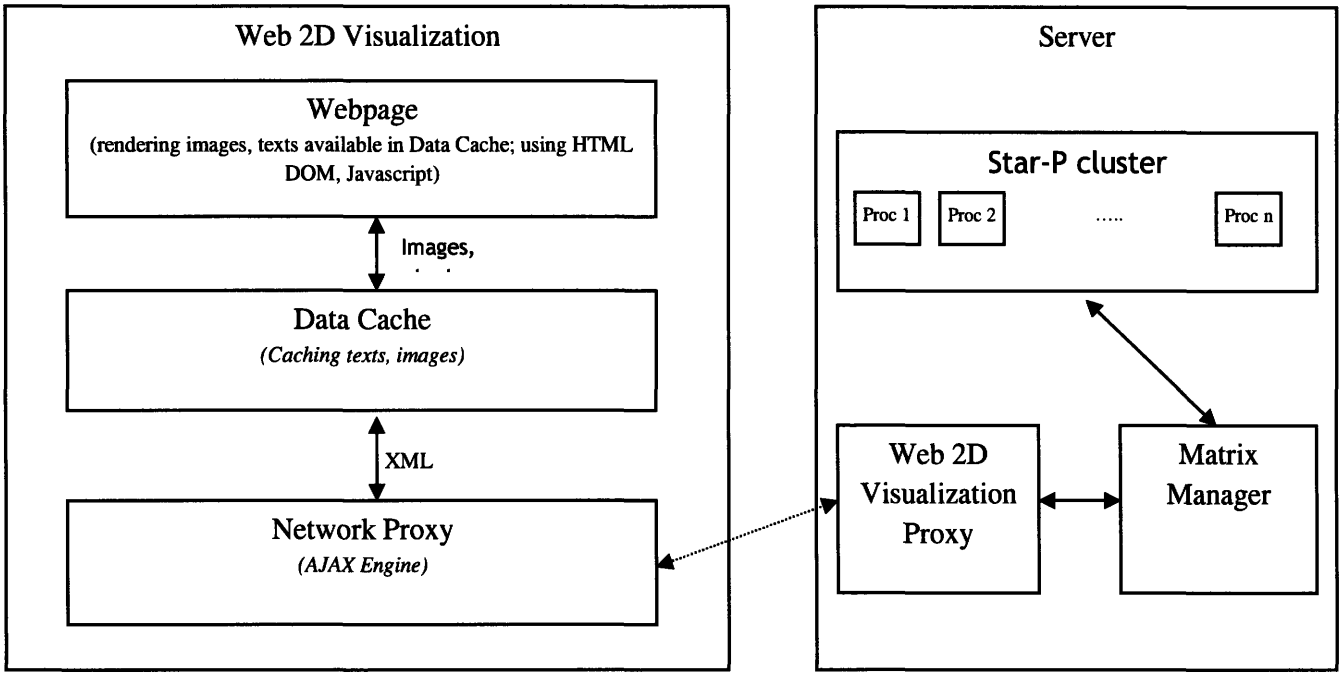


Figure 3 Clip and cache window

Webpage: The Webpage is the location where data is rendered. HTML DOM and Javascript are used for components layout, and processing user control signals such as zooming, drag-and-drop etc.

Data Cache: The Data Cache is responsible for caching texts and Images, making them always available to the webpage. In order to guarantee such availability Data Cache uses pre-caching technique (detail will be shown below).

Network Proxy: Network Proxy is actually an AJAX Engine which is responsible for silently downloading the request data for the Data Cache. The data format used is XML for compatibility.

Pre-caching

The basic idea of pre-caching is trying to make sure that important data is always cached before requested by user.

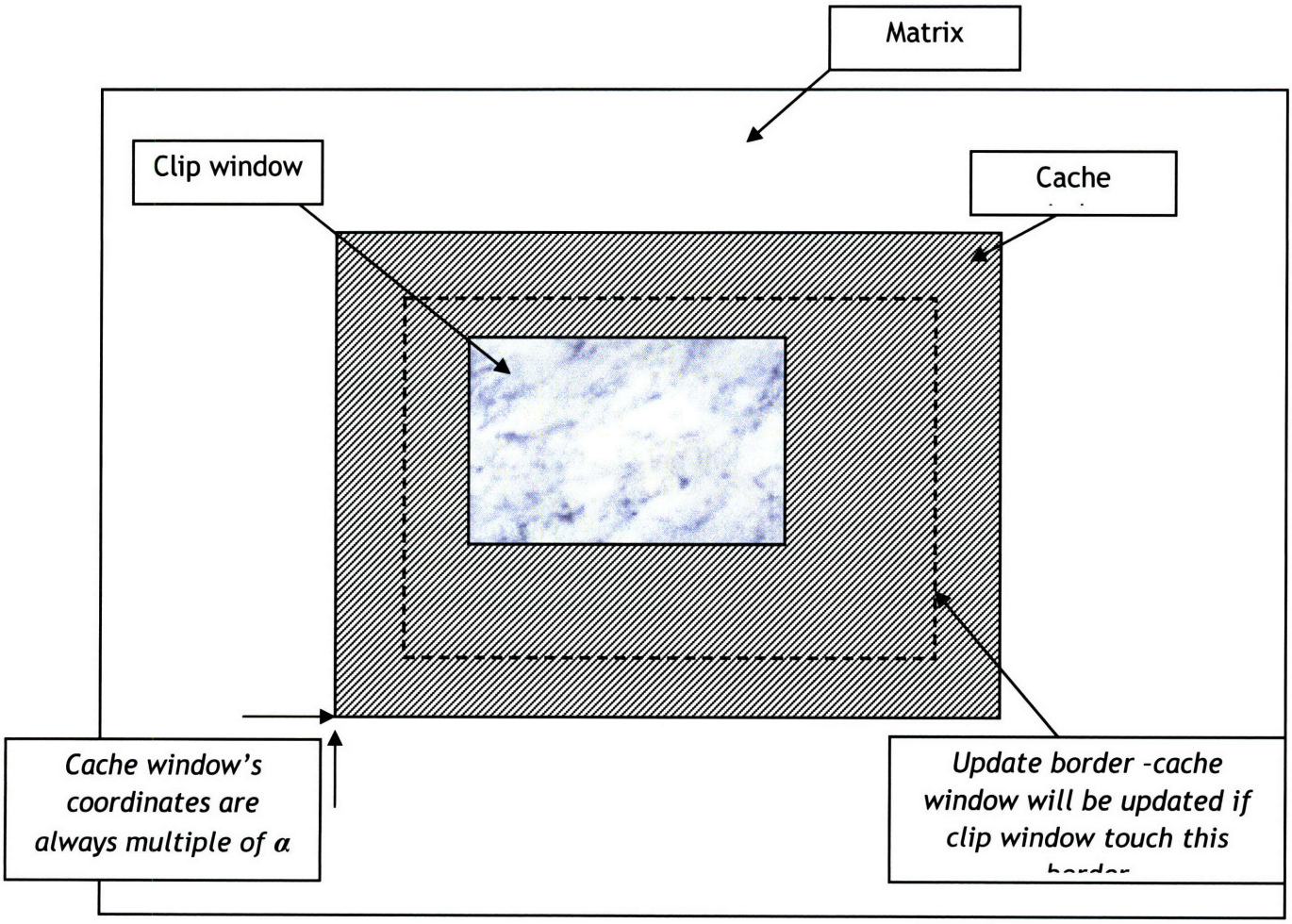


Figure 4 Clip and cache window

In the client side, the user can view matrices by any web browser. This is a very similar view to the map in any online mapping system such as Google Maps, Yahoo Maps or Microsoft Live Maps. Our matrices visualization will be presented inside a rectangular window called a *clip window*.

As we don't want to download the entire matrix (web browsers don't have enough memory for such huge data), we only cache a portion of matrix that contains data in the clip window. This cache is also in rectangular shape and is called *cache window*. The cache window is normally about 4 times larger than clip window (double in each side) so that clip window can fit completely inside it. When the clip window is moved by user, cache window is also moved along to accommodate it. However, since each cache window move is equivalent to a cache update, continuously moving cache window would amplify latency effect and degrade our interactive behavior. Therefore, various techniques have been used to minimizes such moves.

The first technique is to restrict the cache window coordinates to be multiple of α (whose value depends on zoom level) so that the cache will only be updated when clip window is moving far enough (theoretically at least $\alpha/2$). Another technique is to define an update border near each side of cache window. The cache window is only updated when clip window touches this border. The distance of the update border to cache window's side is set based on our caching strategy and networks condition. The smaller this distance, the fewer cache updates needed. However if this distance is too small, latency effect might be experienced as cache window does not have enough time to update when clip window is moving out of it.

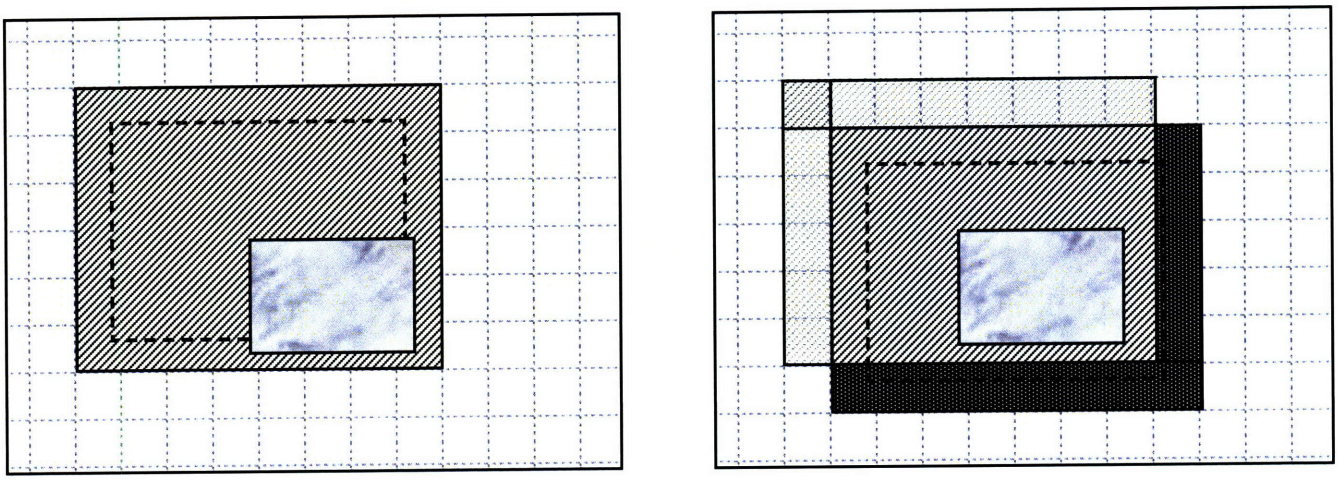


Figure 5 Update border

Finally, we observe that when the cache window is updated, not all of its data is replaced. Normally, just a portion of the cache window is modified on a cache update. Figure 5 describes a usual cache update situation in which the shaded area is added into the cache and the transparent area is removed (if cache window is designed as a 2-dimensional cyclic array, those two areas are actually the same part in that array). Therefore, by only updating those parts that are actually changed, a fair amount of bandwidth and computations has been saved.

Using Web 2D Visualization

It's quite simple to use the Web 2D Visualization since all that we need is a web browser and an Internet connection. Then, the user just needs to login the specified internet address to start visualizing. The remaining process is very much similar to navigating maps online.

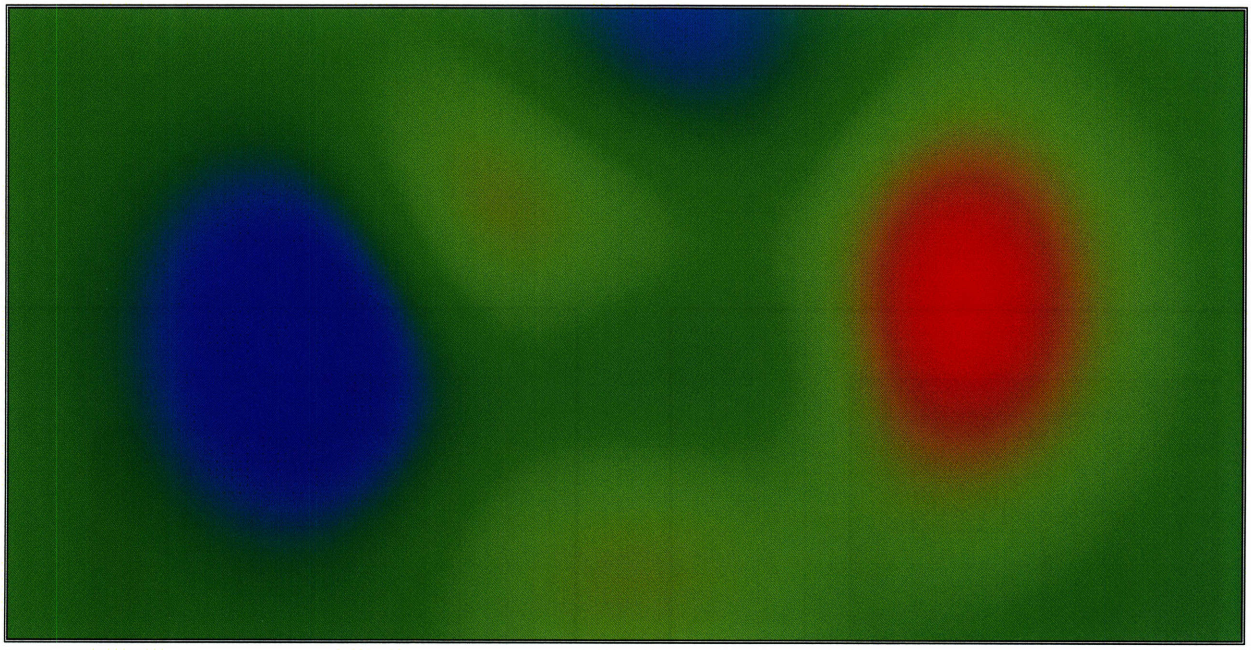


Figure 6 Image view mode

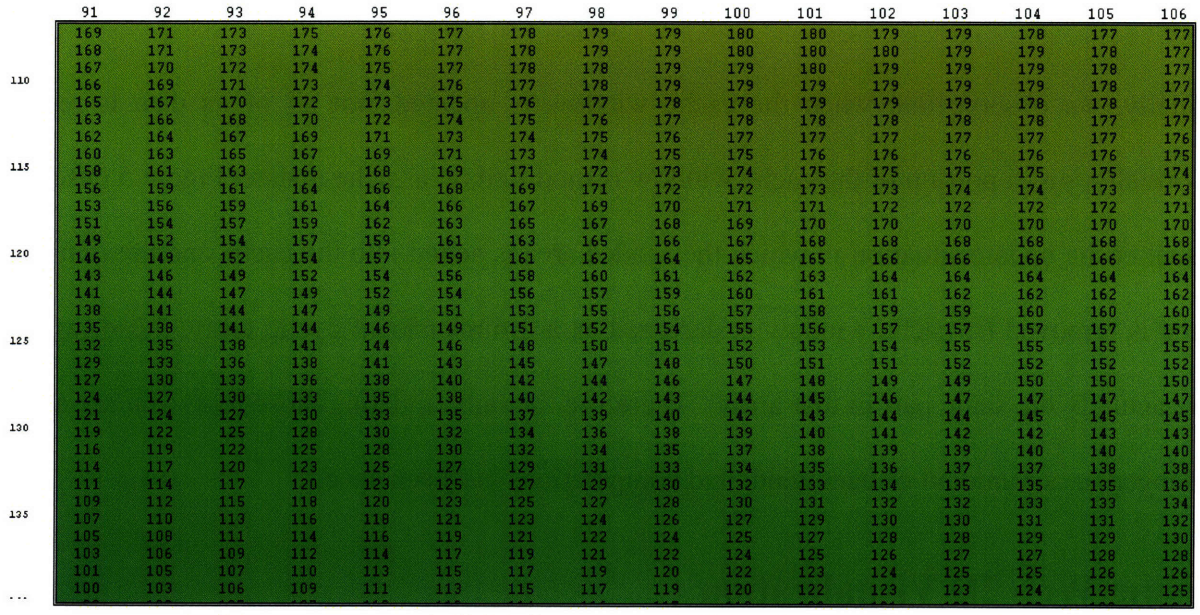


Figure 7 Hybrid view mode

View modes

Depending on the zoom level, there are two view modes available on Web 2D Visualization: Image mode (Figure 6) and Hybrid mode (Figure 7) which is the combination of the image and

numerical values. Image mode is available for all zoom levels while the Hybrid mode is only available in the 3 highest zoom levels (due to the size of text to represent numerical data). Image mode is normally useful for high level study of matrices while the Hybrid mode is more helpful when user wants to work on detail such as in debugging or testing.

Limitation of Web 2D Visualization

Although the Web 2D Visualization is fairly simple and does not require any installation or prior training, its inability to modify or create data makes it dependent on other Star-P applications such as MATLAB Star-P. If these applications are not available, this web application becomes unusable. In the future, we hope to make additional improvements by creating a hook on the website that allows web users to have more control of the data.

Chapter 4
MATLAB Star-P 3D Visualization

The most significant difference between 2D and 3D Visualization interactive applications is the ability to change the view angle. In a 2D world, the user is only allowed to view objects from a fixed angle in a straight down direction, as the previous chapter discussed. Because of this restriction, the whole matrix visualization can be considered very big image. Therefore it is possible to pre-render such image at different zoom levels and send interested parts of it to client upon demand. This is the reason why a light-weight web browser is sufficient to implement a 2D Visualization Client.

In a 3D world, the view angle is changeable. As there is no fixed image in the whole visualization, we can obtain *unlimited* images by looking at the matrix from different view angle and position, and it is impossible to pre-render all of them. Moreover, in 3D world, there are infinite number of points and view angles from which the whole matrix is visible. In such cases, we have to guarantee to have cached enough information to visualize the whole matrix [24]. Otherwise, our application would be at least suffered from latency effect. Obviously, downloading and caching a huge matrix onto user's workstation is extremely expensive and is not considered as a viable solution.

In this chapter, we will present a solution for 3D Remote Visualization for matrix, which not only gives good quality visualization but also guarantees to work at interactive frame rate under unstable network conditions.

Design

As stated in chapter 2, the 3D visualization application is designed in client-server fashion: the Java 3D Viewer Client and the 3D Visualization Server run. The architecture of these two components is described by Figure 8.

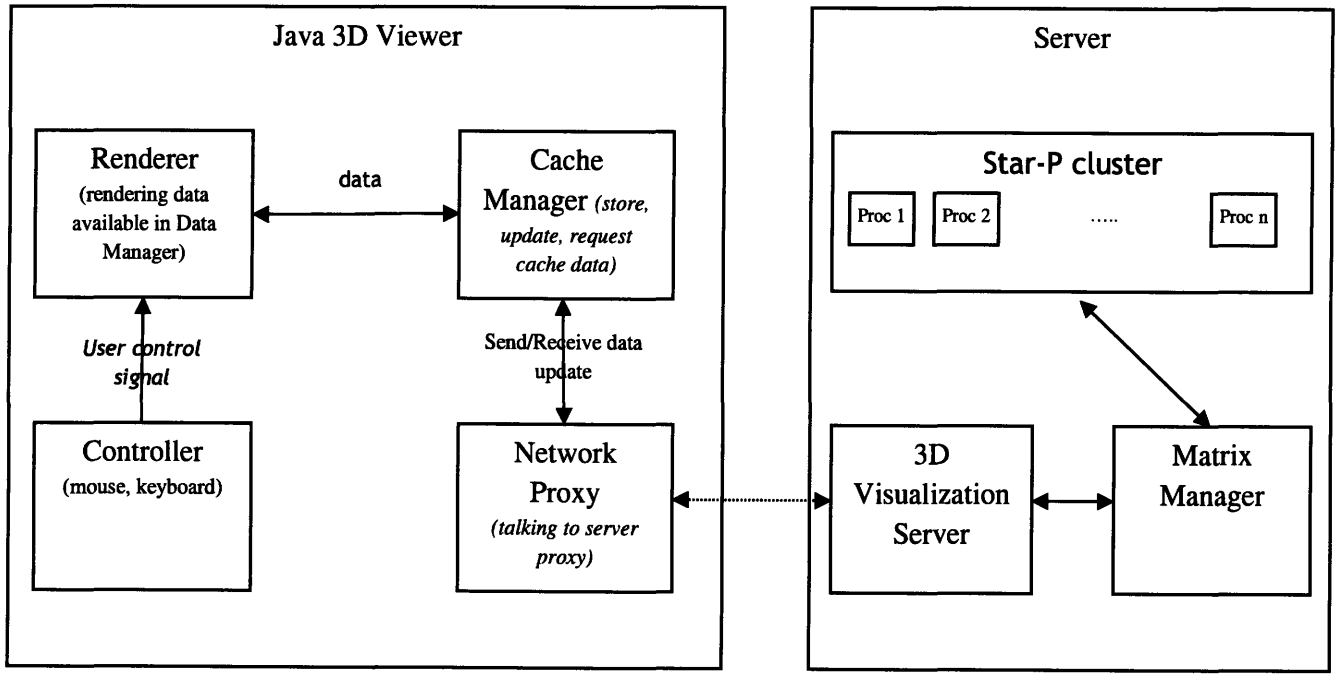


Figure 8 3D Viewer architecture

Java 3D Viewer:

The 3D Viewer consists of 4 different components: Controller, Renderer, Cache Manager and Network Proxy.

Controller: In Java, this component is a listener object that listens to mouse and keyboard events. When an interested event is triggered, Controller should notice the Renderer to change its behavior such as camera position, angle or rendering mesh type...

Cache manager: Cache manager is responsible for storing and managing visualizing data. Moreover, it also helps to keep the Renderer from worrying about the Server and communication problems such as latency. By using various techniques like pre-caching or Geometry Clipmaps, it guarantees to be always available for Renderer's requests and therefore, keep the Renderer run interactively with user.

Renderer: As its name suggested, this component is responsible for drawing objects onto user's desktop screen. Renderer uses OpenGL for 3D drawing. Together, Renderer and Cache Manager is the heart of the viewer system. Cache Manager takes care of the data and Renderer is responsible of how to represent them as best as it can.

Network Proxy: This component is only responsible for implementing the protocol to talk with server. It accepts data update requests from the Cache Manager, forwards them to server and vice versa.

The architecture of three components Cache Manager, Controller and Renderer is analogous to the standard MVC architecture for GUI application. Network Proxy is incorporated into our design to help separate network from visualization. This design separation is not only easier to implement but also more flexible on the protocol used to talk with server. If we would like to change protocol, we can just replace this Network Proxy component and keep the rest of architecture unchanged.

The diagram in Figure 9 shows how our Client-server structure works with MATLAB Star-P. From MATLAB Star-P environment, when 'visualize' function is triggered, it passes control to the local Viewer client. The Viewer then starts the Server remotely by using ppclient provided by MATLAB Star-P. When both the Client (Viewer) and the Server are up and running, they start talking to each other, transferring data and rendering.

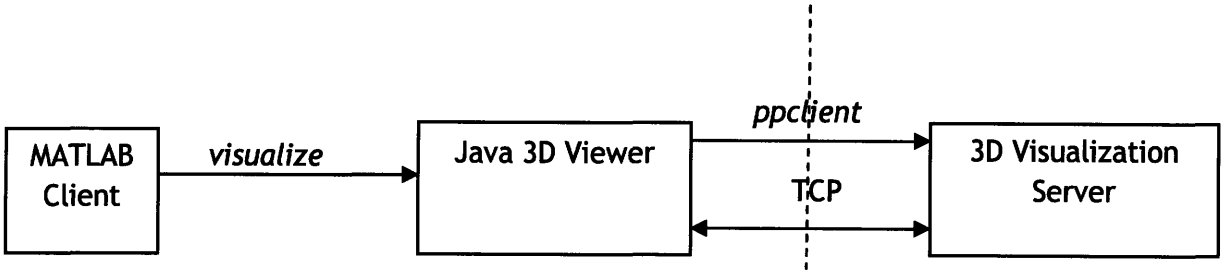


Figure 9 3D Viewer calls diagram

Rendering algorithm:

As stated in the chapter 2, our main concerns in designing this system are communication (network speed, bandwidth, latency) and server computation cost for scalability. Therefore, the 3D rendering algorithm is very important because it decides what information should be provided by the server and which data should be transferred through network. A good rendering

algorithm for this particular purpose is the one that minimizes server computation overhead and query processing, network consumption and number of updates.

Based on those objectives, a number of rendering algorithms are investigated, evaluated and compared to others to choose to most suitable one.

In structure, pre-rendered imagery [18] and VNC, rendering job is implemented on the server and then images, and movies are sent to clients for visualization. The advantage of this approach is simplicity and sensitivity to output. However, they either suffer from the latency effect (VNC) or real time (pre-rendered imagery). They also require a lot of server computation for rendering.

Another approach is to send geometry data to client and let it do rendering job. This approach not only saves server computation for rendering but also is latency-tolerant. Streaming mesh introduced by Isenburg *et al.* [7] reorders vertices and triangles to optimize memory access and speed up rendering algorithm substantially. However, as the geometry data is proportional to the size of matrices, it is impossible to transfer it in raw format; data simplification is required to reduce network consumption. Progressive mesh [15] is a standard simplification method which merges *close vertices* to reduce mesh complexity. This algorithm can be tuned to make sure that the mesh size is transferable under any kind of network. However, since this algorithm is designed for general purposes, it is not optimized for our specific matrix structure; it is complicated and consumes server power. In addition, it is not *locality consistent,* (i.e. visualizing data for neighboring viewpoints don't share much information in common), and thus, the number of updates due to navigation can be large and unmanageable. Another method (Lindstrom *et al.* [3][6][12], Pajarola [16], Atlan *et al.* [19], Rottger *et al.* [13], AMR [2] and ROAM [11][14]) is

using triangle hierarchies to take care of vertices of different level of details. Based on this tree and position of the viewpoint, vertices at appropriate level of detail are selected for rendering. These algorithms ensure a fast render process with low computation cost. However, data structures used in these methods are very complicated and hard to maintain. In addition, they don't have locality consistence property; therefore, cache updates can still be high. To avoid the complicated triangle hierarchies' structure, Joachim *et al.* [26] partitions matrix mesh into tile blocks of different resolutions and chooses appropriate tiles for rendering based on view distance. Although this method has simpler data structure than triangle hierarchies, it has issues in dealing with cracks between 2 tiles.

Visapult uses a hybrid approach; matrix mesh is represented by both pre-rendered images and simple geometry data. The method has proved to be very efficient for remote visualizing of 3D Object. However, this system requires a highly graphic power server where all rendering work is done. In addition, this algorithm is very complicated to implement.

In our application, we decide to use Geometry Clipmaps [20][22] (GC) which falls into the category of algorithms that send geometry data to user. It is simple, fast, requires low server computation overhead, and almost minimizes network consumption and cache updates.

Geometry Clipmaps

Basically, GC is the generalization of clip and cache windows in our 2D version on web. In this algorithm, the *clipmaps* is analogous to the cache windows and *level-of-detail* is equivalent to zoom level. However, unlike 2D version in which only one zoom level is rendered at a time, in GC, many windows of different levels are rendered at once. Large but coarser windows will be

used to render objects far from view point and small but finer windows will be used to render close objects in detail. To achieve this illusion, the only requirement is that all cache windows be centered at the viewpoint which is the same as in 2D Visualization. Of course, to get a high quality of visualization, cache windows of larger size than 2D Visualization is required, but managing them is almost the same.

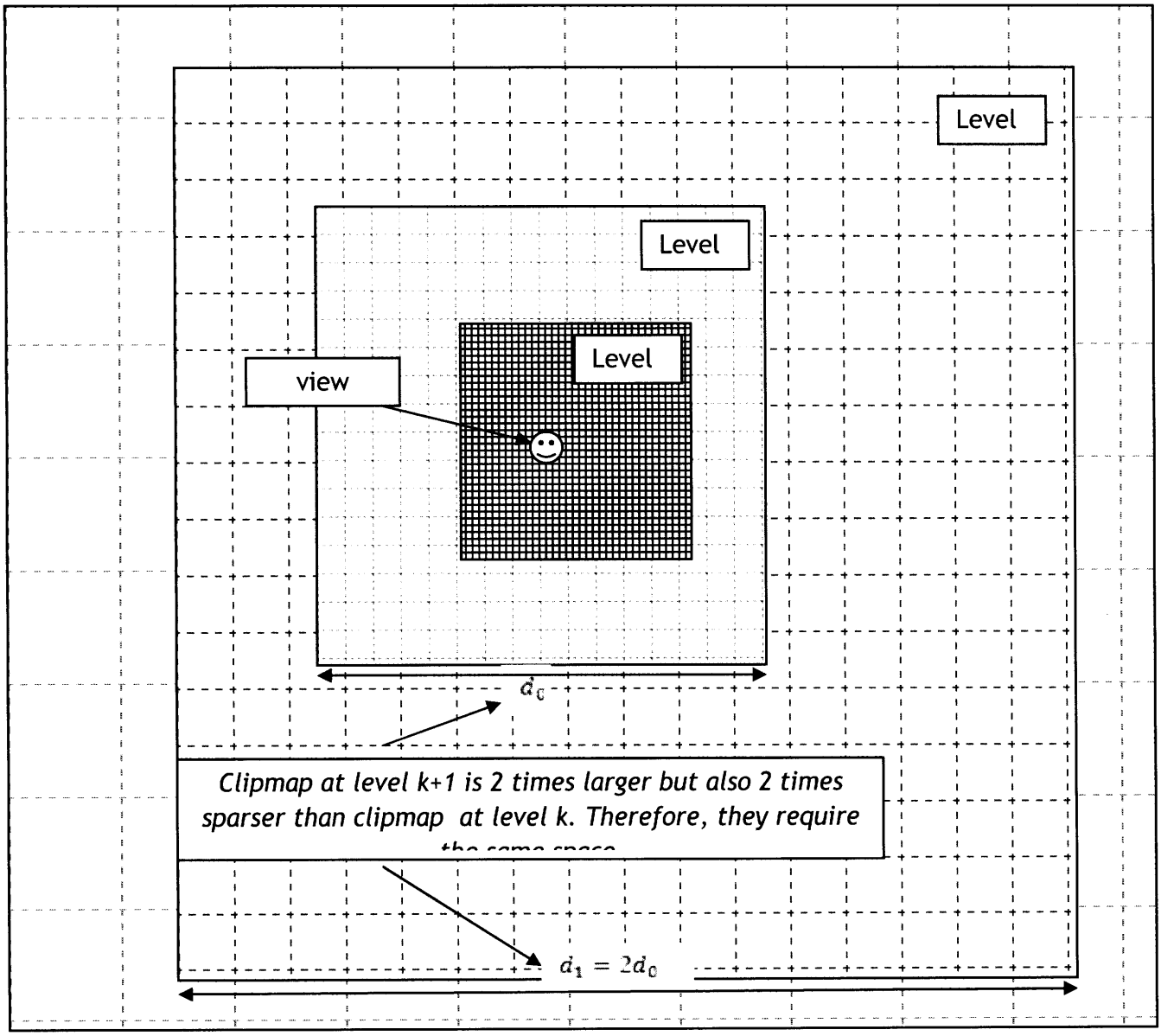


Figure 10 A Geometry Clipmap configuration

Figure 10 shows a configuration of the GC. The clipmaps are indexed in finer-coarser order (clipmap at level 0 is the finest one and cover the least area). Clipmaps of higher level are coarse but the area they cover is also larger. Clipmap at the highest level is the smallest one that covers the whole matrix.

In our algorithm, we set area of clipmap at level k+1 to be 4 times larger than clipmap at level k but also 4 times coarser. Therefore, the space required for clipmaps at different levels are the same. It is apparent from the figure that the clipmaps are not accurately centered at the view-point. This slight displacement is due to the grid alignment (discussed before in chapter 3) of the cache windows to reduce cache updates. Since all clipmaps are supposed to center at the view point, any clipmap of level k should be totally enclosed by clipmaps of higher level. Therefore, even in case clipmap of level k is not available (due to network latency, server computation time) clipmap of level k+1 can be used to render its area. When level k clipmap becomes available, its data will be used to replace data taken from level k+1. Although there should be a visualization difference between these 2 clipmaps and artifact might occur after replacement, in practice, it is rarely noticeable.

Network performance:

Compare to 2D Visualization:

In our implementation of GC, we notice that if the view-point's elevation is high, clipmaps at low level (0, 1, 2 ...) may be too small and indistinguishable. Therefore, removing them from rendering list does not hurt visualization quality much but saves a lot of rendering cost. In fact, for a given view-point elevation, we set the lowest level clipmap to be rendered as:

$$minlevel = \left\lceil \log_2\left(\frac{2z\tan\frac{\theta}{2}}{2d_0}\right)\right\rceil$$

where

$$z = view\ point\ elevation$$

$$\theta = view\ angle$$

$$d_0 = size\ of\ clipmap\ level\ 0$$

Essentially, *minlevel* indicates the lowest level whose area is at least 4 times larger than the visible square when the view-direction is straight down (bird-eye view).

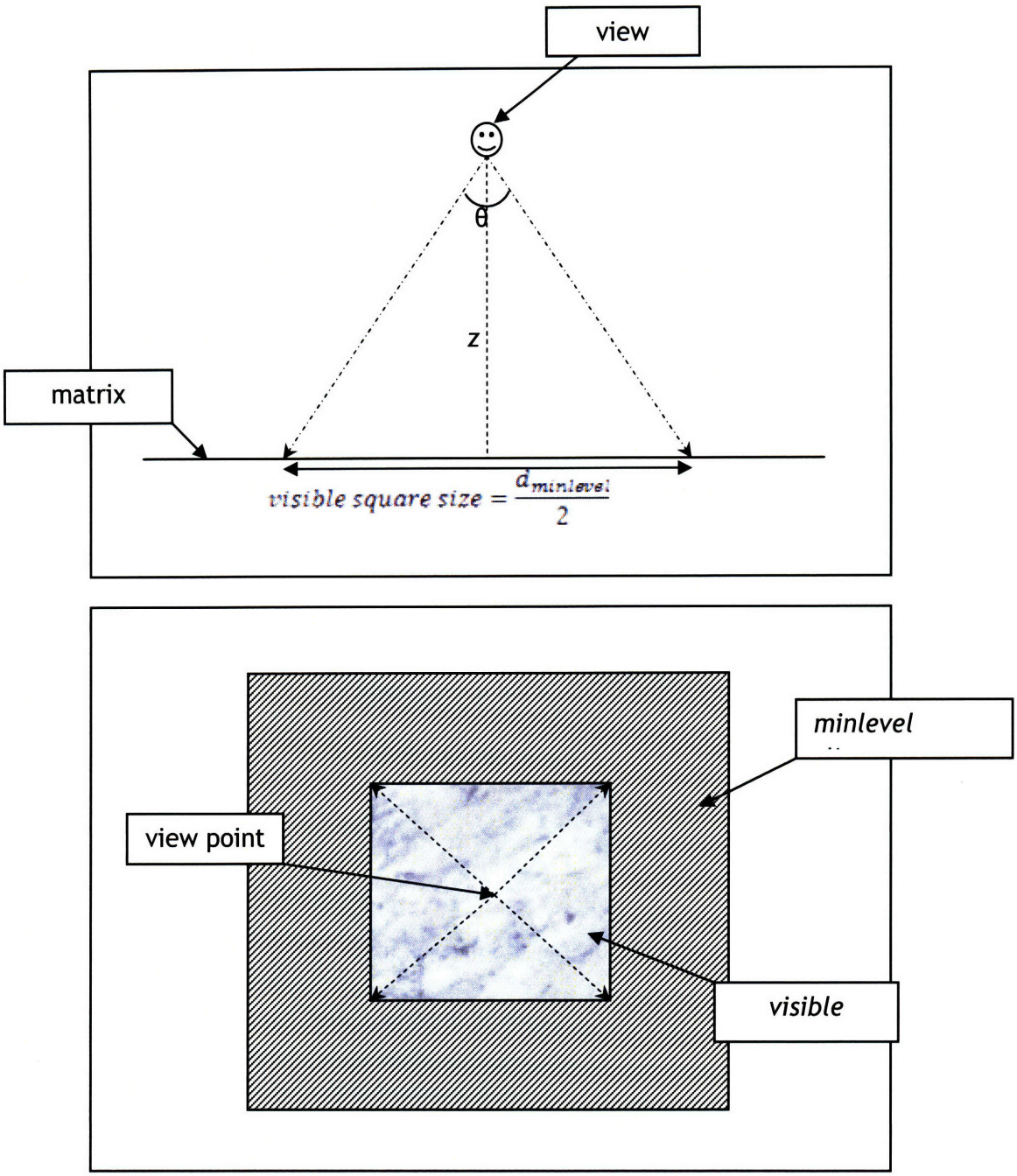


Figure 11 *minlevel* clipmap and visible window

It can be noticed from Figure 11, visible square and *minlevel* clipmap are very similar to the cache and clip windows in chapter 3. Therefore, if we also set the panning rate to be comparable to the panning rate in Web 2D Visualization, the cache behavior of our 3D Viewer

should be similar to the 2D case. More precisely, they are only different in a constant factor. Therefore, our 3D Viewer should have the similar network performance as in 2D.

Although, the above argument does not prove any absolute bound on the network performance, it does give a relative bound that roughly speaking, if the Web 2D Visualization works, the 3D Viewer should also work. In practice, both of them run smoothly under typical high speed internet connection.

Server computation overhead and query cost:

Since no rendering are required on the server, computation overhead and query cost in GC are both low. In addition, the block-wise independent server query type of GC allows us to use the technique shown in Chapter 2 to reduce the server query cost to constant.

Network adapting

As network unreliability is one of our major concerns in building remote applications. Adapting to networking condition not only helps us avoid annoying problems such as lost packet, network congestion, etc. but achieve better visualization quality under good networking condition.

We seek to build the streaming of visualizing data in such a way that whatever information received at client side is sufficient to rendering *something* and if more information arrives, it can be used to achieve better graphic quality. This principle suggests us to order the data in coarser-to-finer style, which fits naturally to our clipmaps structure. Coarser data are clipmaps at higher level while finer data are low level clipmaps. Each time, a group of clipmaps updates are received at the server, those updates for higher level clipmaps will be sent back to clients first.

On the client side, the visualization program renders all information it has received so far. During rendering process, if the level clipmaps are not yet available, all lower level are ignored.

Using Star-P 3D Visualization

Besides its efficiency in rendering a view, our visualization application is also designed to be as much user-friendly as possible. Assuming a distributed matrix ***a*** was created in MATLAB Star-P, for example:

```
>> [X,Y] = meshgrid(-3:(6/400):3*p);
>> a = peaks(X,Y)
a =

      ddense object: 401-by-401p
```

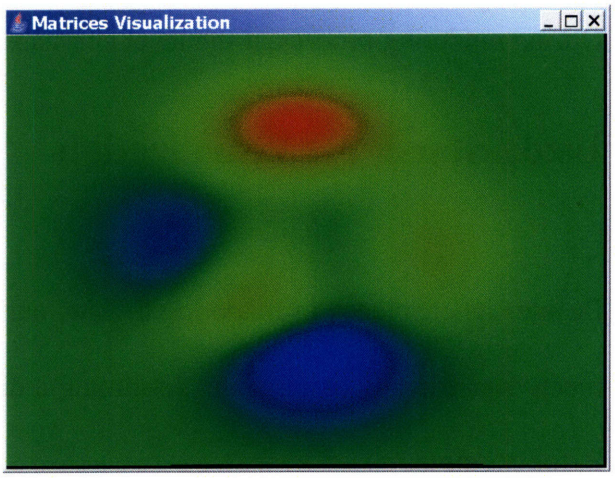
to visualize ***a***:

```
>> visualize('a');
```

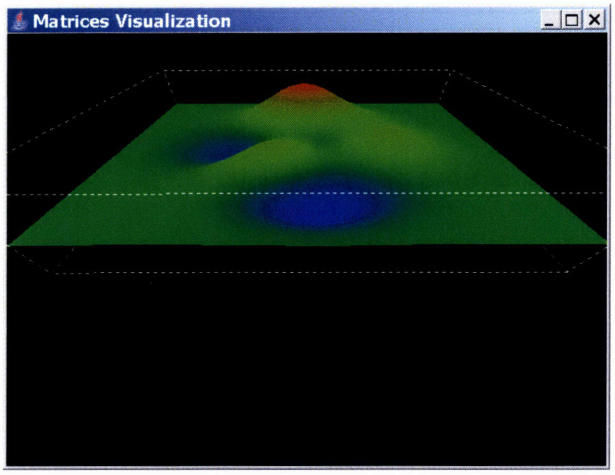
(Note: The single quote is required for technical reasons).

A viewer window will appear in response to the command and the user is freely to navigate (zoom, pan, rotate) his/her matrix in 3D with mouse or keyboard. Figure 12 shows a few screen shots for visualization of our example matrix ***a.***

Peaks matrix in 3D Viewer, from bird-eye view point



From a different view point



A closer look

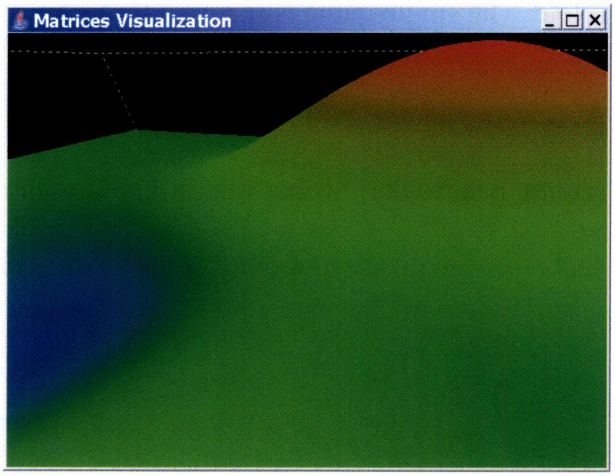


Figure 12 Some screen shots of the 3D Viewers for peaks matrix

Chapter 5
Future work

Because of time constraints, there are several interesting features have not been implemented in our system.

Firstly, as stated in the end of chapter 3, the Web 2D visualization still has limitation in data controlling, making it only useful for presenting one's work to others through the Web. Without the support of other applications such as MATLAB Star-P to create matrices, Web 2D is almost unusable. A solution to this problem is to build a Controller on the Star-P server which accepts both *controlling message* and query message from client. When the incoming message is for querying, it will be forwarded to the Matrix Manager and processed the same as before. If it is a controlling message (e.g matrix creation, addition...), the Controller should be authorized to perform corresponding operation on Star-P server and send the error/status message back to clients. Currently, with the support of Star-P SDK, we believe that it is possible to build such Controller. If we are successful, our Web 2D Visualization will be greatly improved to have both the power of an independent MATLAB client and the popularity of the Web.

Secondly, in the current implementation, only one front end processor on the server is responsible to gathering data and process client requests. As a result, there is only one download/upload channel used between client and server. Although this centralized structured has advantage of simplicity, it is not optimized in server load balancing and network consumption. As one of our next steps in this project, we would like to de-centralize server

process, allow multiple processors to process client request at once and utilize parallel network channels to increase network performance.

Finally, we are also interested in the chance of expanding domain of visualized objects which is currently limited to 2D matrices. Although this seems to be the most natural *next step* for our project, it has many theoretical and practical challenges. As can be noticed in chapter 4, the rendering algorithm was carefully selected and tuned to take advantages of nice structures in 2D matrices class. In other data classes, such structures hardly exist. Consequently, current algorithm can't be generalized to work with other classes and more sophisticated algorithms are needed. Currently, even the existence of such algorithms is still an open question to us.

Bibliography

1. VNC Documentation. AT&T Laboratories, Cambridge, UK, http://www.cl.cam.ac.uk/research/dtg/attarchive/vnc/howitworks.html.

2. **Kähler R., Prohaska S., Hutanu A., Hege H.C.** Visualization of Time-Dependent Remote Adaptive Mesh Refinement Data. 2005, IEEE Visualization.

3. **Lindstrom P., Pascucci V.** Visualization of Large Terrains Made Easy. 2001, IEEE Visualization .

4. VisIt: Parallel Visualization and Graphical Analysis Tool. http://www.llnl.gov/VisIt/home.html.

5. Visapult. http://vis.lbl.gov/Research/visapult2/.

6. **Lindstrom P., Pascucci V.** Terrain Simplification Simplified: A General Framework for View-Dependent Out-of-Core Visualization. 2002, Vols. 8(3): 239-254 , IEEE Trans. Vis. Comput. Graph.

7. **Isenburg M., Lindstrom P.** Streaming Meshes. 2005, IEEE Visualization .

8. **Choy, R., Edelman, A.; Gilbert, J.R.; Shah, V.; Cheng, D.** Star-P: High Productivity Parallel Computing. June 2004, Technical report.

9. Star-P Manual. Interactive Supercomputing.

10. **Shah V., Gilbert ,J. R.** Sparse Matrices in MATLAB Star-P: Design and Implementation. 2004, HiPC .

11. **Duchaineau M. A., Wolinsky M., Sigeti D.E., Miller M. C., Aldrich C., Mineev-Weinstein M.B.** ROAMing terrain: real-time optimally adapting meshes. 1997, IEEE Visualization.

12. **Lindstrom P., Koller D., Ribarsky W., Hodges L.F., Faust N., Turner G.A.** Real-Time, Continuous Level of Detail Rendering of Height Fields. 1996, Vols. 109-118, SIGGRAPH.

13. **Rottger S., Heidrich W., Slusallek P., and Seidel, H.P.** Real-time generation of continuous levels of detail for height fields. Central Europe Conf. on Computer Graphics and Vis., .

14. Real-Time Dynamic Level of Detail Terrain Rendering with ROAM. http://www.gamasutra.com/features/20000403/turner_01.htm.

15. **Hoppe H.** Progressive Meshes. 1996, SIGGRAPH .

16. **Pajarola R.** Overview of quadtree based terrain triangulation and visualization. Jan 2002, Report UCI-ICS TR 02-01,.

17. **Tuchman A.M., Berry M.W. .** Matrix visualization in the design of numerical algorithms. 1990. , ORSA Journal on Computing, 2(1):84-92.

18. **Chen J., Yoon I., and Bethel E. W. .** Interactive, Internet Delivery of Scientific Visualization via Structured, Prerendered Imagery. April 20, 2005, Lawrence Berkeley National Laboratory. Paper LBNL-57528. .

19. **Atlan S., Garland M.** Interactive Multiresolution Editing and Display of Large Terrains. June 2006, Vols. Volume 25, Number 2, 211-223(13), Computer Graphics Forum.

20. **Losasso F., Hoppe H.** Geometry clipmaps: terrain rendering using nested regular grids. 2004, Vols. 23(3): 769-776, ACM Trans. Graph.

21. **Rasmussen M., Newman M., Karypis G.** gCluto Documentation. 2004, http://www-users.cs.umn.edu/~mrasmus/gcluto/doc/gcluto-0.5/index.html.

22. **de Boer W. H.** Fast Terrain Rendering Using Geometrical MipMapping. October 2000, E-mersion Project.

23. **Ware C., Franck G.** Evaluating stereo and motion cues for visualizing information nets in three dimensions. 1996 , ACM Transactions on Graphics (TOG).

24. **Ng C.M, Nguyen C.T, Tran D.N, Tan T.S., Yeow S.W.** Analyzing Pre-fetching in Large-scale Visual Simulation. June 2005, Proceeding of Computer Graphic International Conference (CGI), .

25. **Garrett J.J.** Ajax: A New Approach to Web Applications. February 2005, Adaptive Path .

26. **Joachim Pouderoux, Jean-Eudes Marvie.** Adaptive streaming and rendering of large terrains using strip masks. 2005, GRAPHITE .

27. **Bethel W., Tierney B., Lee J., Gunter D., Lau S.** Using high-speed WANs and network data caches to enable remote and distributed visualization. 2000, High Performance Networking and Computing Conference.