Hardware-Accelerated 3D Visualization of Mass Spectrometry Data

José de Corral* — Waters Corporation

Hanspeter Pfister† — Mitsubishi Electric Research Labs (MERL)

Abstract

We present a system for three-dimensional visualization of complex Liquid Chromatography - Mass Spectrometry (LCMS) data. Every LCMS data point has three attributes: time, mass, and intensity. Instead of the traditional visualization of two-dimensional subsets of the data, we visualize it as a height field or terrain in 3D. Unlike traditional terrains, LCMS data has non-linear sampling and consists mainly of tall needle-like features. We adapt the level-of-detail techniques of *geometry clipmaps* for hardware-accelerated rendering of LCMS data. The data is cached in video memory as a set of nested rectilinear grids centered about the view frustum. We introduce a simple compression scheme and dynamically stream data from the CPU to the GPU as the viewpoint moves. Our system allows interactive investigation of complex LCMS data with close to one billion data points at up to 130 frames per second, depending on the view conditions.

CR Categories: I.3.7 [Computing Methodologies]: Computer Graphics—Three-Dimensional Graphics and Realism; I.3.2 [Computing Methodologies]: Computer Graphics—Graphics Systems

Keywords: Mass Spectrometry, Terrain Rendering, GPU Rendering

1 Introduction

Liquid Chromatography - Mass Spectrometry (LCMS) is an analytical technique resulting from a combination of Liquid Chromatography (LC) and Mass Spectrometry (MS). It is widely used in the life sciences and in drug discovery for applications such as protein analysis. LCMS data has three components: time, mass, and relative intensity. Typically, two-dimensional subsets of the data are processed using numerical methods and then visualized in 2D.

In recent years, the demand for faster and more accurate analysis results have resulted in an increase in the size and complexity of data obtained from LCMS instruments. To keep up with this pace, analytical chemists use software tools to help them interpret the data. The system presented in this paper is one of these tools, allowing the users to interactively visualize LCMS data in 3D as shown in Figure 1, where relative mass detector intensity is plotted with respect to time and mass-to-charge ratio.

A three-dimensional visualization allows LCMS practitioners to have an overview of the entire dataset as well as detailed close-ups. It can help them to identify features in the data that simultaneously vary with mass and time. This is hard to do using just 2D plots and normally requires some prior knowledge about the sample. The 3D visualization also permits quick identification of peak clusters that are near the data noise level, confirmation of the presence of a peak predicted by numerical methods, and other interactive exploration of the data. Consulted LCMS users were unaware that this type

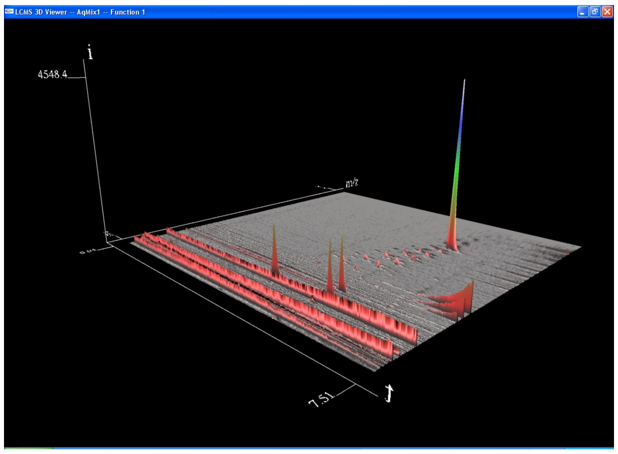

Figure 1: *Three-dimensional visualization of LCMS data.*

of 3D visualization was possible. They encouraged the authors to develop this system, as they envisioned its great potential.

Modern LCMS datasets range from tens of millions to over a billion data points, making interactive 3D visualizations a non-trivial problem. Also, LCMS data spacing is non-uniform and varies greatly along each axis. Our system renders large LCMS datasets in 3D on modern graphics processing units (GPUs). We have adapted the level-of-detail (LOD) techniques of *geometry clipmaps* [15, 2] that were developed to render large terrain data at video frame rates. Geometry clipmaps are based on a pyramid of nested rectilinear grid rectangles at power of two resolution that are re-filled with new data as the view changes. We use the same conceptual LOD approach, but use different LOD levels, data compression, and storage formats. Our system is capable of interactively exploring LCMS datasets with close to one billion data points (355,329 mass points $\times$ 2,669 time steps).

2 Previous Work

Typically, LCMS data is viewed in 2D. There are several tools available for this, including plot programs that are part of the Matlab bioinformatics toolbox. Matlab and some recent software tools [12] also allow a general or partial view of LCMS datasets in 3D. These tools are limited by either the size of data they can display or their rendering performance. We are not aware of any other attempt to visualize large LCMS datasets interactively with LOD techniques on modern GPUs.

LCMS data visualization in 3D has similarities with terrain rendering, as both use elevation maps. In recent years, terrain rendering has received a lot of attention. The focus of most of this work has been on real-time terrain visualization with LOD techniques (e.g., [6, 4, 5, 11, 13]). Others pursue a drastic reduction in the number of vertices necessary to render the terrain image, resulting in irregular grids for terrain representation (e.g., [14, 8, 3, 7]).

LCMS data visualization in 3D has also substantial differences with terrain rendering that make most of the previous terrain ren-

*e-mail: jose.de.corral@waters.com

†e-mail: pfister@merl.com

dering work not directly applicable. Terrain is in general relatively smooth with no sharp features. On the other hand, LCMS data is full of spike-like features embedded in flat sections. For highest performance, terrain grids are sampled at regular intervals along both axes, making tessellation of the model easy [16]. The sampling rate of LCMS data is different along each axis. It is nonlinear, and there are typically about one hundred times more samples along the mass axis than the time axis. Figure 2 illustrates this issue. A typical terrain sampling grid is shown on the left, and a typical LCMS data grid on the right. The sampling rate is higher for low mass values than for high mass values and is broken occasionally with bursts of samples. Time sampling is normally regular, except that some samples might be missing when the instrument interleaves a mass scan for other purposes.

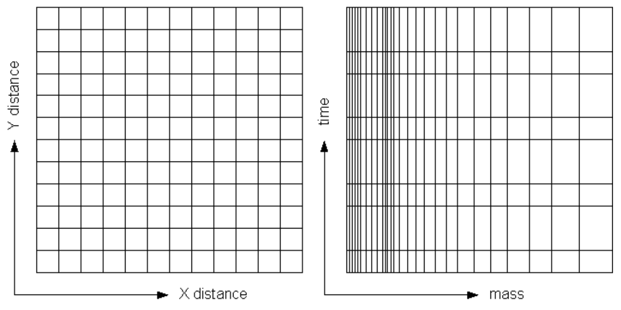

Figure 2: *Terrain and LCMS data sampling comparison.*

Initially, we considered an irregular grid approach [5, 11, 8] because LCMS data has contiguous regions of zero intensity. This is promising for vertex reduction techniques that result in irregular grids of vertices. However, we realized that some LCMS datasets have more noise and no regions of zero intensity, which reduces the benefit of this technique in those cases. In addition, every LCMS dataset would have a different vertex distribution, making it difficult to overlay and compare two LCMS datasets of the same size. Besides, rendering an irregular grid is usually slower and more difficult. Therefore, we decided that a rectilinear grid approach was a better solution. Although LCMS data is not evenly spaced along the time or mass axes, it is rectilinear in the sense that it is made of rows and columns of data points.

Among the terrain-rendering approaches we studied, we found the work by Losasso and Hoppe [15] to be the best fit for our problem, although there are substantial differences between the two solutions. Losasso and Hoppe use a lossy compression algorithm, which we cannot use for reasons mentioned later. They use a pyramid structure of nested regular grid rectangles centered about the viewpoint (i.e., viewer position), with each rectangle representing the data at a power of two resolution, and equal resolution across the two horizontal axes. While we use a similar pyramid structure, ours is centered about the view frustum, and the sampling resolution along each axis is very different.

Losasso and Hoppe address visual continuity between levels of the pyramid in their work, but we found this not essential for our application. The spike-like nature of the LCMS data makes it difficult to see these transitions except in relatively flat regions of the data. Nevertheless, this is a subject for future work. Similar to Losasso and Hoppe, we store the entire dataset in main memory in compressed form, while the geometry that is being rendered is stored in video memory using vertex buffers. However, our method to perform the video memory updates from main memory is different. In both cases, viewer distance determines which pyramid levels are visible. Also, due to the toroidal addressing, Losasso and Hoppe recompute vertex indices each frame. We found a method to avoid this.

3 Overview of LCMS

Liquid Chromatography (LC), also commonly known as High Performance Liquid Chromatography (HPLC), is an analytical technique used to separate the chemical components of a sample mixture. The sample is dissolved in a solvent and pushed through a liquid chromatograph (Figure 3) that performs the separation into chemical components [17, 18]. A high-pressure pump pushes the solvent from a container into the rest of the elements at a steady flow. The sample injector inserts the sample solution into the solvent stream without disrupting the solvent flow. The device that performs the actual separation of the sample components is called a separation column or simply column. The column is a piece of tubing packed with material that allows the solvent to go through but has some physical or chemical affinity with the components of the sample. The components with strong affinity will be retained inside the column longer than those with less affinity. Therefore, the different components of the sample will be washed out from the column at different times. Finally, a detector device gives a response dependent on some physical property of the components, such as UV light transmission, index of refraction, electrical conductivity, etc. The response of the detector, therefore, varies with time as the different sample components emerge from the column. The duration of a LC analysis, also known as a run, can be anywhere from a few minutes to a couple of hours or more, depending on the type of sample.

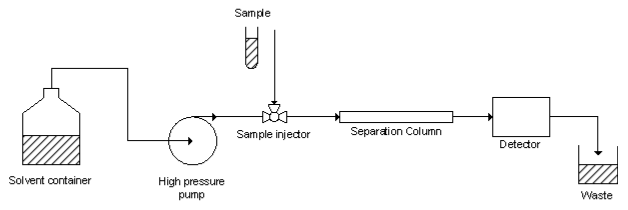

Figure 3: *Block diagram of a liquid chromatograph.*

The LC instrument provides data that varies with time. Most LC detectors provide a single data point per unit time, resulting in a 2D plot known as a chromatogram. Typically, the LC detector output stays flat and changes only (normally increasing) when a sample compound reaches it. This creates in the 2D plot what are known as chromatographic peaks. Figure 4 shows an example of a LC chromatogram from an Ultraviolet (UV) detector showing several peaks with their amplitude in absorbance units (AU).

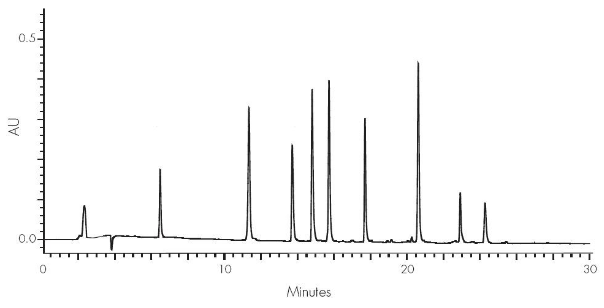

Figure 4: *Example of a LC chromatogram.*

Mass Spectrometry (MS), also known as Mass Spec, is a very powerful analytical spectroscopic technique to determine the mass of molecules [1, 10, 17]. A mass spectrometer has three major parts: the ionization source, the analyzer, and the detector.

The sample is introduced in the ionization source, where the molecules of the sample are fragmented and ionized. The analyzer is a device that uses electrical or magnetic fields, or both, to accelerate the ions and move them from the ionization source to the detector. The electrical or magnetic fields are chosen such that most of the ions hit the walls of the analyzer, and only those with a specific mass-to-charge ratio manage to reach the detector. Changing the electric or magnetic fields continuously allows different ions to reach the detector at different times, creating what is known as a mass scan. A mass scan normally takes less than a second, and covers a wide range of mass-to-charge ratios, typically from 50 to 2000 Atomic Mass Units (AMU) per charge. Another type of analyzer, known as Time of Flight (TOF), sets the electrical field such that none of the ions hit the analyzer walls and all ions reach the detector, although they do it at different times. Finally, the detector gives an intensity response proportional to the number of ions that reach it at any given time.

The mass spectrometer generates an array of data points of relative intensity that vary with mass (MS scan). An MS scan 2D plot shows peaks just like a LC chromatogram, although the peaks are sharper than those of a LC chromatogram and normally more abundant. Figure 5 shows an example of an MS scan.

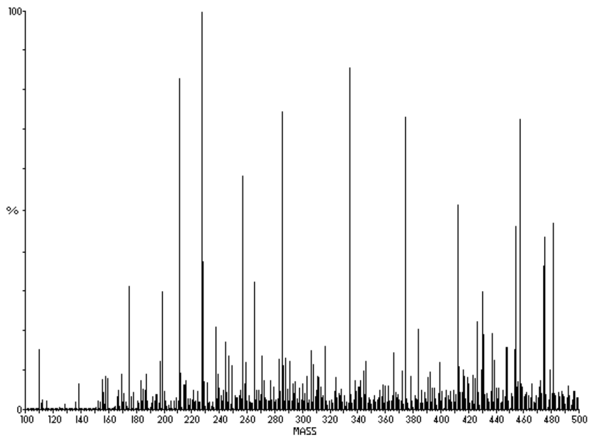

Figure 5: *Example of an MS scan.*

These two analytical techniques, LC and MS, are used independently in many applications . However, they can also be combined such that the output of the LC column (Figure 3) is sent to the ionization source of the mass spectrometer. In other words, the mass spectrometer becomes the LC detector. This is the hybrid technique known as LCMS. The data produced by an LCMS system is a 3D surface plot with the three axes being time, mass-to-charge ratio (the independent variables), and intensity (the dependent variable). Considering that an MS scan may contain anywhere from a few thousands to almost half a million data points, and that the LC chromatogram may have from a few hundred data points to a few thousand, it is easy to see that the LCMS technique creates a huge amount of data.

An LCMS practitioner uses different methods to tackle this massive amount of data. For example, in virtually all LCMS datasets there are areas of no interest. If the user knows something about the sample being analyzed, he or she may focus on specific areas. Another common approach is to process the data using numerical methods on 2D subsets of the data, specifically on individual scans. Normally, a scan produced at a particular time of the LC run is compared with another scan containing known information. The other scan may come from a library of known compounds, in which case the user is looking for compound identification [1]. Alternatively, the second scan may come from a different analysis by the user, in which case he or she is looking for scan differences that may reveal certain drug interactions on a subject.

4 System Overview

Our intent is to display in 3D an entire LCMS dataset or sections of it at any desired resolution. This visualization may be used as a tool to gain insight into a sample analysis, which otherwise may be difficult to grasp using traditional methods. The user should be able to observe the 3D plot from far away to get a general view, or zoom up to the full resolution of the data. One fundamental requirement in this application is that the compression method used to store the data in memory is lossless. Given the important and precise relation that exists between the coordinates of different peaks, the data should be preserved without degradation. Also, since the user may request these coordinates with a mouse click, their exact value should be accessible.

The system uses a simple, efficient, lossless compression algorithm to store an entire LCMS dataset in main memory. The user has the option to use the raw data, or to reduce the data specifying a fixed sampling interval along the mass axis. The data in main memory is used to stream data into video memory for rendering. To render the image, the system uses a LOD approach based on a pyramid of nested rectangles (PlotMaps) centered about the view frustum. The PlotMaps are stored as vertex buffers in video memory and are refilled with data from main memory as the viewpoint moves. To allow small viewpoint movements without the need for memory updates, each PlotMap has in video memory twice the size of data required for rendering. When an update is necessary, we use toroidal addressing [15] to update the PlotMap with the new data only.

The system provides independent scale adjustments for each of the three axes to help with the interpretation of the data. Improperly set scales can alter the aspect ratio of the data enough to make it difficult, or even impossible, to see any relation between the peaks. We use ordinary Phong shading and color the surface based on height. The system also provides a picking mechanism to help the user identify a peak by its coordinates (mass, time, intensity).

5 Data Format and Compression

The raw LCMS data from a single analysis comes in different files grouped in a directory. The instrument interleaves mass scans obtained with different instrument conditions. Each file contains the scans acquired under a particular instrument condition. Inside each file, the scans are stored in time-ascending order. Each scan is marked with the time of acquisition and consists of a sequence of mass value and intensity value pairs sorted in ascending order of mass value.

In a typical mass scan there are many mass values with associated zero intensity. None of these mass/intensity pairs are saved to the file, except for those that are adjacent to a non-zero intensity value. This lossless compression technique reduces the size of the data file, but can be improved further. Our data structure to store the LCMS data in main memory eliminates the remaining zero intensity pairs. It also replaces the mass/intensity value pairs with mass index/intensity value pairs. In a typical large LCMS file the reduction in size with our method is about 35%.

Our mass scan configuration maps a mass index to an intensity value. The intensities data structure is then made of an array of these scans, one for each mass scan in the data. Figure 6 show a diagram of the resulting data structure in main memory. The figure shows a mass array with *m* elements of distinct mass values, a time array with *n* elements, and *n* scans, one per element of the time

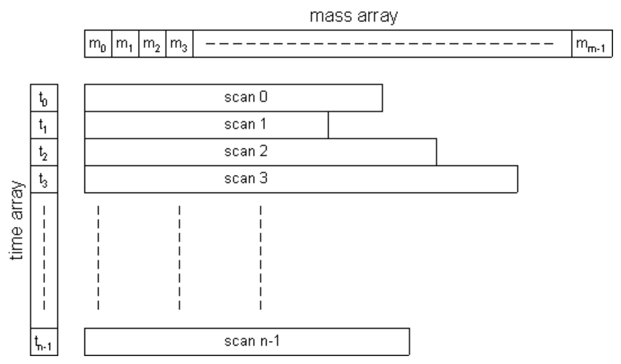
mass array
m_0 m_1 m_2 m_3 m_{m-1}
time array
t_0 t_1 t_2 t_3 t_{n-1}
scan 0
scan 1
scan 2
scan 3
scan n-1

Figure 6: *Data structure in main memory.*

array. The number of elements for each scan is different (typically) and it is smaller than *m*.

Given a time index *t* and a mass index *m* we get the time, mass, and intensity values from the data structure as follows. We use the time index *t* to read the time value from the time array and to address one of the mass scans. We use the mass index *m* to read the mass value from the mass array and to read the intensity value from the addressed scan at the same *m* index. If index *m* is not present in the scan, the intensity is zero.

We use this data structure to fill the PlotMaps with data of the appropriate subsampling level. A distinct characteristic in this application is that the data reduction may be different in the mass versus the time axis because the PlotMap resolution is different along each axis. Since each PlotMap needs the data at a different power of two resolution depending on its LOD level, we subsample the data appropriately while filling the PlotMaps. However, we cannot use straightforward subsampling because the high frequency of the data would result in severe and unacceptable aliasing. Similarly, filtering the data would produce visible intensity value (height) transitions between PlotMaps. Instead, during subsampling we keep track of the maximum intensity value of all raw data points and assign the maximum intensity value found to the reduced sample. The time and mass values are subsampled normally, though.

6 Level-Of-Detail Hierarchy

We use a LOD structure consisting of a pyramid of nested PlotMaps at power of two resolutions, just like Losasso and Hoppe. Each PlotMap represents a rectangular section of the data, not necessarily square as in the case of geometry clipmaps. The tessellation value is the number of data points (minus one) along each side of the PlotMap rectangle, and it is normally different along each axis (time and mass). In addition, the system has a maximum tessellation value for each of the axes that will not be exceeded. An important difference with Losasso and Hoppe is that the PlotMap points are not evenly spaced as in the case of regular terrain grids.

When a new data file is read, the system examines the size of the data and decides how many LOD levels are required to cover the data with the LOD structure. In particular, it checks how many data points there are across the mass axis and across the time axis, and computes the mass and time tessellation values. The system then defines a different number of LOD levels for the mass axis and for the time axis.

For each axis, given the tessellation value, the number of LOD levels are the number of levels necessary such that the highest resolution PlotMap covers a section of the data at full resolution, each of the next lower resolution PlotMaps covers a section of the data double in size but at half the resolution than the previous PlotMap, and the lowest resolution PlotMap covers the entire data. The tessellation value along each axis, which remain the same for PlotMaps at all levels, is chosen to be smaller than the defined maximum, and is forced to be an even number so that the points at adjacent levels align.

For example, assume that the maximum mass tessellation value is 200 and the number of points across the mass axis for a particular file is 50,000. We first compute the maximum even number under 200 that multiplied by a power of two results in the closest number under 50,000. In this case, $194 \times 2^8 = 49,664$. The exponent is the number of LOD levels minus one. Thus, the mass tessellation value is 194 and the number of mass levels is 9. 49,664 is the number of raw data points actually covered by the lowest resolution PlotMap, 194 is the number of points of the PlotMap across the mass axis, and $2^8 = 256$ is the number of consecutive raw data points we have to read from the data structure to get to each of the subsampled 194 points. For the next higher level PlotMap results $194 \times 2^7 = 24,832$, and similarly for higher level PlotMaps. The same method is used for the time tessellation and number of time LOD levels.

Note that the lowest resolution PlotMap does not cover the entire 50,000 mass points. The number of points that are not covered is always a small percentage of the data. This is generally not an issue for LCMS data because the users set the instruments to acquire data well before and after the region of interest. However, an alternate tessellation method that overcomes this limitation involves choosing a larger tessellation value such that the number of data points covered is greater than the number of points in the file. The system then displays zero intensity value for data points that are beyond the data range. This method, however, requires more video memory as each PlotMap is larger.

Given that typically there are more data points across the mass axis than the time axis, there are almost always more mass LOD levels than time LOD levels. The system creates as many PlotMaps as the largest of the two LOD levels. Figure 7 shows an example with nine PlotMaps that have nine mass LOD levels and four time LOD levels. The correspondence between LOD level, mass axis

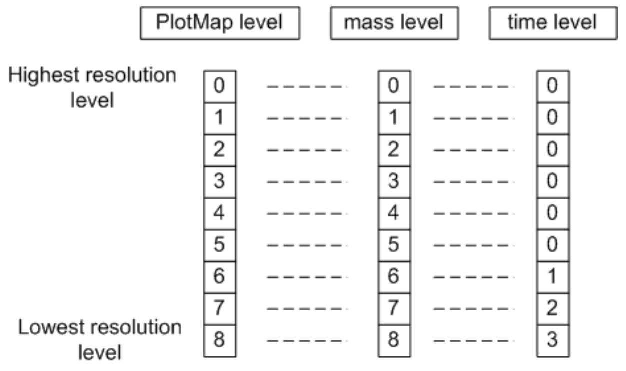

Figure 7: *PlotMap LOD levels (method A).*

LOD level, and time axis LOD level is shown using dashed lines in the figure. To maintain the maximum time resolution in as many PlotMaps as possible, time level zero is repeated for LOD levels zero through five. These PlotMaps were generated from a data file with 53,006 points in the mass axis, 322 points in the time axis, a maximum mass tessellation of 400, and a maximum time tessellation of 50. The computed tessellation was 206 in the mass axis and 40 in the time axis. In this example, the maximum time tessellation was purposely set too low to force four time levels to illustrate the level distribution.

Using a flat grid and different colors, Figure 8 shows a vertical view of the data area covered by each PlotMap with this method. The user-defined mass and time scalings (see Section 8)

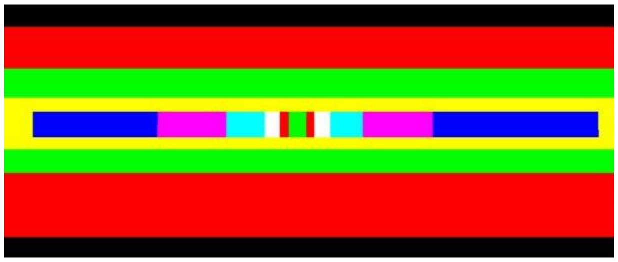

Figure 8: *PlotMaps LOD with resolution mismatch (method A).*

were set to render PlotMap level zero (small green rectangle) approximately square. Figure 8 shows that this method creates an exaggerated sampling mismatch between PlotMaps along the mass axis. PlotMap levels zero though five have PlotMap level six (in yellow) as neighbors along the horizontal (mass) axis. The differences in resolution are extremely noticeable during rendering.

Instead of repeating the highest resolution time level, we repeat the lowest resolution (level three) to solve this problem (Figure 9). Figure 10 shows a vertical view of the data area covered by each

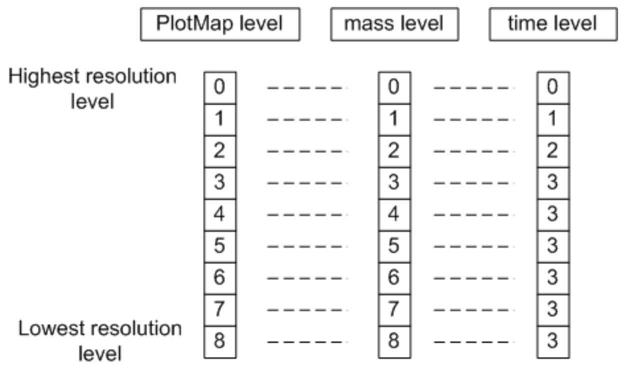

Figure 9: *PlotMap LOD levels (method B).*

PlotMap with this method, but only PlotMaps zero through six are shown in this case. Again, the mass and time user-defined scal-

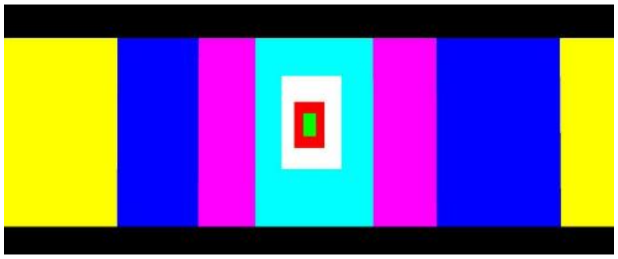

Figure 10: *PlotMaps LOD without mismatch (method B).*

ings were set to render PlotMap level zero (in green) approximately square. Note that all adjacent PlotMaps are only one LOD level apart. The PlotMap data is stored in an array organized as rows along the mass axis and columns along the time axis. The number of columns is the mass tessellation value, and the number of rows is the time tessellation value. The PlotMap data array is stored in video memory using Vertex Buffer Objects (VBO) [9].

7 Data Streaming

We define three regions within each PlotMap: the memory area, the draw area, and the hole area (Figure 11). The regions typically

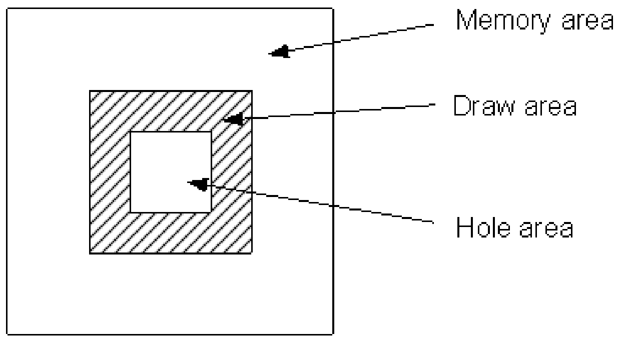

Figure 11: *PlotMap regions.*

are rectangular. Their relative size in each direction is shown in the figure. The draw area can be anywhere within the memory area, and the hole area can be anywhere within the draw area.

The draw area is the region that is actually shown on the screen, the hole area is the region occupied by the PlotMap of the next LOD level, and the memory area is the region that the PlotMap keeps in video memory (i.e., the entire PlotMap data). The memory area is set to be twice the size of the draw area in each direction. This allows the PlotMap to draw without delay if the draw area moves slightly within the memory area. The hole area is by definition half the size of the draw area in each direction, because adjacent PlotMap levels vary by a factor of two.

The position of the PlotMap draw area within the entire data moves with the view frustum. If the movement is small, the PlotMap draw area moves within the memory area and the drawing is immediate. However, if the draw area moves too close to or beyond the memory area border, the PlotMap moves its memory area such that it becomes centered with the new draw area (Figure 12).

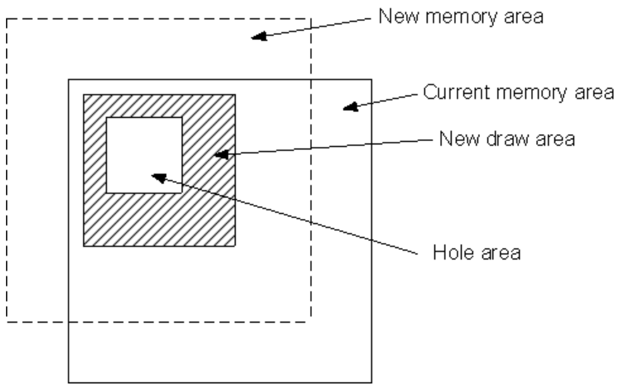

Figure 12: *PlotMap memory area move.*

To move the memory area, the PlotMap updates it with new data from the data structure in main memory. However, the PlotMap does not refill the memory area entirely; it only updates the new data that is necessary. The PlotMap uses toroidal addressing to store the data in the PlotMap data array, and a pair of toroidal offset indices to keep track of where the memory area origin is within the array.

As stated earlier, the PlotMap data array is organized as rows along the mass axis and columns along the time axis. When the array is loaded for the first time, the rows are in time-sequential order, each row is in mass-sequential order, the memory area origin coincides with the array origin, and the toroidal offsets are zero.

Figure 13 shows an example of a PlotMap data array with a mass tessellation of 200 and a time tessellation of 50 that has been updated with new data. The figure shows a mass toroidal offset of

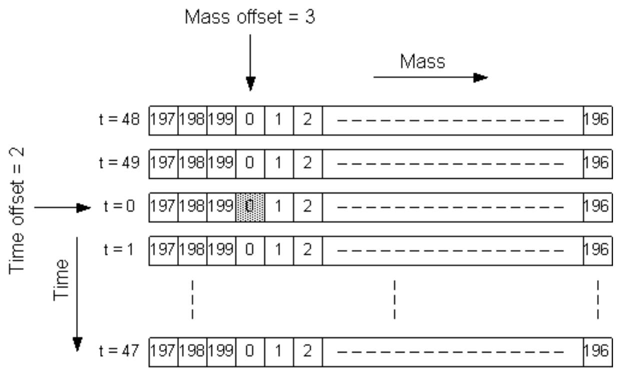

Figure 13: *Updated state of a PlotMap data array.*

three and a time toroidal offset of two. The memory area origin (the shaded square) and the array origin do not coincide anymore.

To draw the data in the PlotMap draw area, we use triangle strips parallel to the mass axis with one triangle strip per time value. To avoid drawing the hole area, we partition the draw area in four sectors and draws each one of them separately.

The PlotMap needs to locate the draw area data origin within the PlotMap data array, and dereference the toroidal addressing. Dereferencing the toroidal addressing along the time axis determines which triangle strip inside the array is going to be drawn, and dereferencing the toroidal addressing in the mass axis determines where within the triangle strip the drawing should start.

We use different methods to dereference the time and the mass axes. For the time axis, we add the toroidal time offset to the position of the triangle strip and apply the modulo operand to get the actual position inside the data array. For the mass axis, we use the indices that are used to draw the triangle strips. Normally the indices need to be re-computed as the toroidal offset changes. We use a simple approach to avoid this re-computation (Figure 14). The

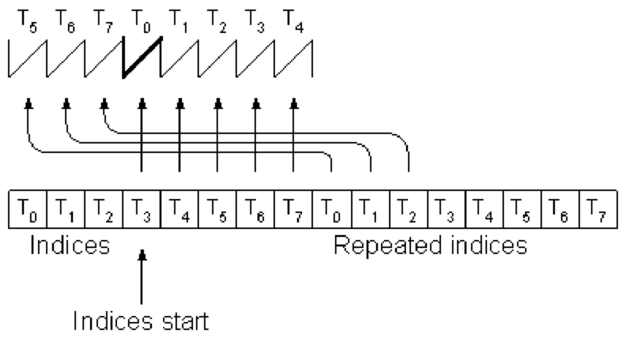

Figure 14: *Repeated triangle strip indices.*

size of the array of indices is doubled and the same index sequence is repeated. Now, instead of starting the indices from the beginning of the array, we start at the location that corresponds to the toroidal mass offset. The triangles from that point to the end of the strip are produced by the indices in the first part of the array of indices, and the rest of the triangles are produced by the indices in the second part of the array of indices.

8 Rendering

The system controls the LOD on every view frustum move and communicates to each of the PlotMaps the location of the draw area, the location of the hole area, if it should have a hole, and if it should be visible. As mentioned earlier, each PlotMap is responsible for the location of its memory area based on the requested draw area and determines if it needs a memory update to honor the new draw area request.

The lowest resolution PlotMap does not need memory updates because by definition its draw area covers the entire data. The second lowest resolution PlotMap does not need memory updates either because its memory area covers the entire data. Recall that the memory area is twice the size of the draw area in each direction. Consequently, the two lowest resolution PlotMaps are always ready for immediate drawing.

The LOD pyramid is centered at the point of the view frustum intersection with the mass-time plane that has the shortest distance to the camera eye position. The width of the frustum intersection at that point is used to determine which is the highest resolution PlotMap that should be visible.

If a PlotMap determines that an update is necessary, it declares itself as unloaded and blocks drawing. The system uses a background task to perform the updates. In the meantime, new drawing request may be arriving. In this case, the system checks which PlotMaps are available for drawing (not declared as unloaded) among those declared as visible, and performs the drawing with them. At least we know that the two lowest resolution PlotMaps are always available for drawing. The background task performs the PlotMap updates and triggers a re-draw after each PlotMap is updated. Because the PlotMaps are processed from the lowest level upwards, this automatically results in progressive refinement of the mesh.

The system translates and scales the entire 3D plot from object space to world space such that it is enclosed in a unit-cube bounding box. This transformations helps to handle the camera without having to deal with the size of the data or the data aspect ratio in object space. This default transformation, however, may distort the aspect ratio of the data to a point where the user has difficulty recognizing the features of the data that he or she is expecting. To solve this issue, we allow the user to enter a custom scaling for each of the three axes.

Each PlotMap uses a fragment program and pixel buffers to compute the normals on every update and stores them in Pixel Buffer Objects (PBO). Phong lighting is done per fragment for smoother shading, and the lighting computations are done in world space rather than in object space due to the strong scaling of the time and mass axes. The color of each point in the 3D plot is a function of the intensity value at that point. This function is stored in a 1D texture that the fragment program uses in conjunction with the lighting computations.

We provide the user with the ability to interactively pick the object space coordinates of any point of the 3D plot on the screen. We use the depth buffer and inverse transformations to compute these coordinates.

9 Results

For our tests we used a PC with a 3.4GHz Pentium IV CPU, 2 GB of main memory, and an NVIDIA 6800 Ultra 256MB graphics card. We used different types of LCMS data at the extremes of the typical size range. The small datasets have about 15 million points with file sizes in the order of 1 MB, and the large datasets have about 1 billion points with file sizes between 750 MB and 1.1 GB.

Our data compression method to store the data in main memory reduces the data by 35% with respect to the file size. This reduction varies, of course, with the data contents. Due to the overhead of the mass and time arrays, the compression ratio becomes smaller for smaller files and finally becomes negative for the smallest files. We compared our compression method to the well known zip compression algorithm. Table 1 summarizes the results. The results

Method	Data Size	Compress	Decompress
Our	682.5 MB	3 min 35 sec	23 sec
Zip	497.7 MB	13 min 40 sec	1 min 14 sec

Table 1: *Compression comparison. Original file size: 1,073.7 MB*

show that zip compression is more effective but took much longer time to compress and decompress, which is not acceptable for our application.

Figures 1 (LCMS 0) and 15 (LCMS 1 through LCMS 4) show sample views of different datasets. The window size was 640×480 in all cases. Table 2 lists the relevant parameters for each dataset. The number of vertices reported is the actual number of distinct vertices in the draw area of the currently visible PlotMaps that are sent to the GPU.

The performance of our system varies between 4 and 130 fps. On average we render about 55 million triangles per second. Because we are currently not doing frustum culling, the number of processed vertices increases as the number of visible PlotMaps increases. Therefore, the performance is in inverse proportion to the size of the data and to the proximity of the view. Large datasets need more PlotMap levels, and when the data is viewed close up all PlotMaps may be visible. The size of the display window has only a minor impact on the performance because we are doing lighting computations in the fragment program.

The time to update each PlotMap from main memory is inversely proportional to the PlotMap resolution. This is due to the fact that all PlotMaps fill their data from the same data structure in main memory, and lower resolution PlotMaps need to read proportionally large areas of the data. The time to load the entire highest resolution PlotMap for a pyramid with 8 levels with a mass tessellation of 414 and a time tessellation of 322 is 190 milliseconds.

10 Conclusions and Future Work

We have presented an approach to interactively visualize LCMS datasets in 3D using modern graphics hardware. The response from a small group of LCMS practitioners has been very positive and enthusiastic. A user mentioned that the system is useful because he was able to confirm the existence of small peaks that were predicted by some numerical tool. Another user said that the system will be very useful to detect sequences of peaks that are related by mass and time simultaneously, which is difficult to see using 2D plots. The detection of peak clusters near the noise level was mentioned by another user as one of the advantages, especially since quantitative peak information can be easily obtained using a click with the mouse. Another user mentioned that he was surprised by the density of the data and how he could appreciate the signal-to-noise ratio of the data using the system.

There are two immediate enhancements that will increase the rendering speed and overall responsiveness of the system. One is view-frustum culling. With view-frustum culling, as implemented by Losasso and Hoppe [15], the number of vertices sent to the GPU is greatly reduced with a corresponding increase in performance. The other improvement involves the data layout in main memory. Instead of having just one data structure from which all LOD levels fill their data, we could use one data structure per LOD level. This will substantially improve memory updates, especially for the lower resolution LOD levels.

T-junction removal is another opportunity for future work. Despite the fact that the irregular nature of the LCMS data is masking this effect, it would be beneficial in certain cases. Another feature desired by some users is the ability to instruct the system to go to a particular point with given coordinates. This would be useful to judge the relations between peaks that are quite distant to each other. A minor enhancement is a peak information feature that provides quantitative peak apex data rather than information about the clicked point location.

11 Acknowledgements

We would like to thank Marc Gorenstein and Chris Stumpf from Waters Corporation for providing us with the LCMS data samples and for their enthusiastic comments and Jennifer Roderick Pfister for proofreading the paper.

References

[1] R. E. Ardrey. *Liquid Chromatography Mass Spectrometry: An Introduction.* John Wiley & Sons, West Sussex, England, 2003.

[2] A. Asirvatham and H. Hoppe. *GPU Gems 2*, chapter Terrain rendering using GPU-based geometry clipmaps. Addison-Wesley, March 2005.

[3] A. Bogomjakov and C. Gotsman. Gpu-assisted z-field simplification. In *International Symposium on 3D Data Processing, Visualization and Transmission*, pages 673–679, 2004.

[4] P. Cignoni, E. Puppo, and R. Scopigno. Representation and visualization of terrain surfaces at variable resolution. *The Visual Computer*, 13(5):199–217, 1997.

[5] D. Cohen-Or and Y. Levanoni. Temporal continuity of levels of detail in delaunay triangulated terrain. In *IEEE Visualization*, pages 37–42, 1996.

[6] W. H. de Boer. Fast terrain rendering using geometrical mipmapping. FlipCode, http://www.flipcode.com/articles/article_geomipmaps.pdf, 2000. Cited March, 2005.

[7] M. Duchaineau, M. Wolinski, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. Roaming terrain: Real-time, optimally adapting meshes. In *Visualization '97*, pages 81–88, October 1997.

[8] J. El-Sana and A. Varshney. Generalized view-dependent simplification. In *Eurographics*, pages 83–94, 1999.

[9] R. Fernando. Using vertex buffer objects (VBOs). http://developer.nvidia.com/. Cited March, 2005.

[10] P. Gates and P. Skelton. Cambridge university mass spectrometry WWW server. http://www-methods.ch.cam.ac.uk/meth/ms/. Cited August, 2005.

[11] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization*, pages 35–42, 1998.

[12] Waters Informatics. Masslynx mass spectrometry software. http://www.watersinformatics.net/. Cited August, 2005.

[13] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time, continuous level of detail rendering of height fields. In *ACM SIGGRAPH*, pages 109–118, 1996.

[14] P. Lindstrom and V. Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE TVCG*, 8(3):239–254, 2002.

[15] F. Losasso and H. Hoppe. Geometry clipmaps: Terrain rendering using nested regular grids. *ACM Transactions on Graphics*, 24(3):769–776, 2004.

[16] M. McGuire and P. G. Sibley. A heightfield on an isometric grid. Poster presentation, SIGGRAPH 2004, July 2004.

[17] W. M. A. Niessen. *Liquid Chromatography Mass Spectrometry*. Marcel Dekker, New York, 1999.

[18] R. P. W. Scott. *Liquid Chromatography for the Analyst.* Marcel Dekker, New York, 1994.

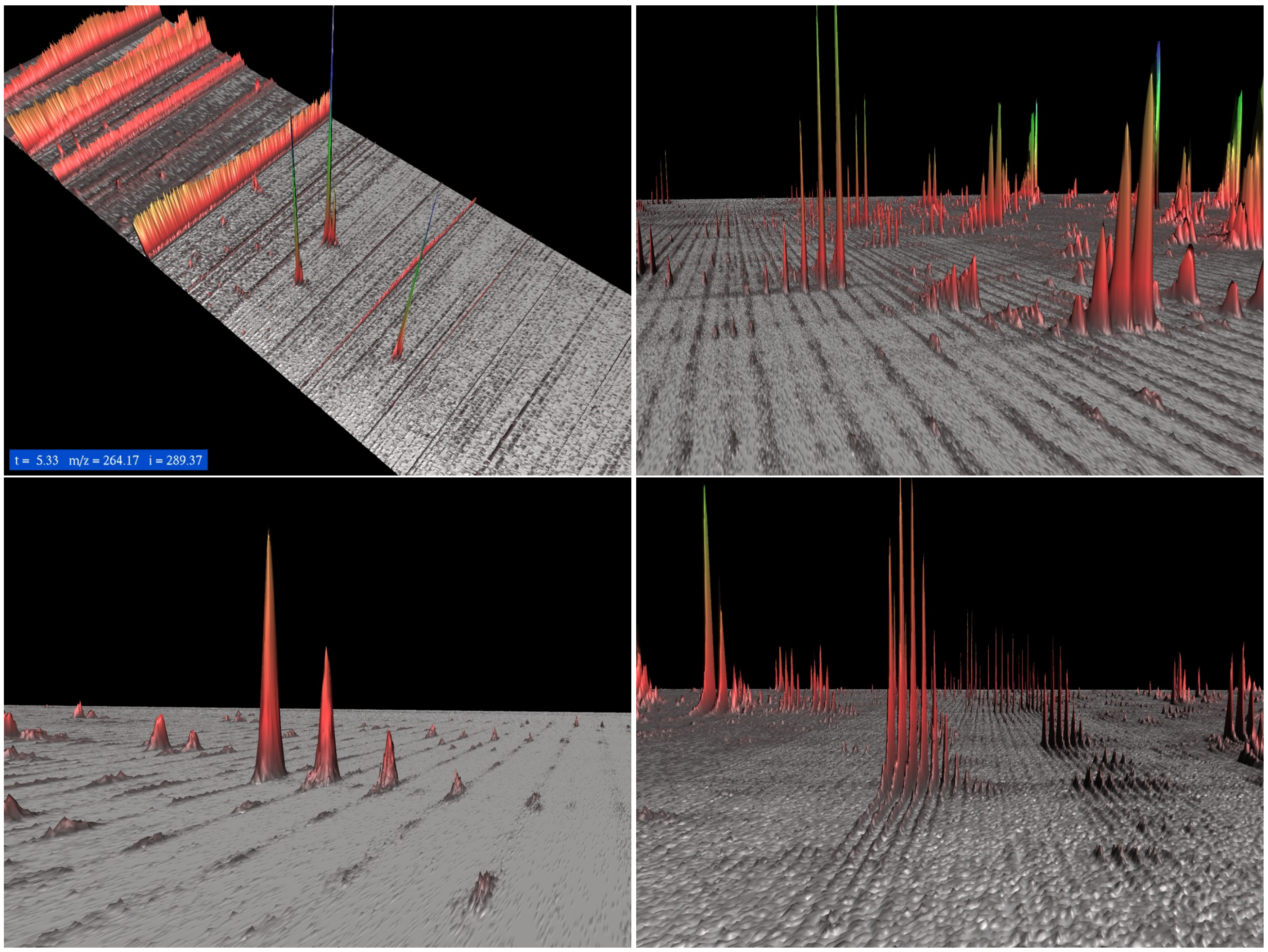

Figure 15: *3D LCMS visualizations for datasets LCMS 1 through LCMS 4 (clockwise from upper left). Dataset LCMS 0 is shown in Figure 1.*

Data	File Size	Data Points		Tessellation		LOD Levels	Vertices	Avg. fps
		Mass	Time	Mass	Time			
LCMS 0	0.9 MB	52,993	322	414	322	8	134,045	134
LCMS 1	0.9 MB	51,713	322	404	322	8	327,037	62.5
LCMS 2	757.6 MB	355,329	2,669	694	1,334	10	5,798,901	5
LCMS 3	757.6 MB	354,305	2,669	346	1,334	11	2,895,276	14.5
LCMS 4	560 MB	355,329	1,567	694	1,567	10	8,445,640	3.5

Table 2: *Performance comparison for various datasets. Numbers under mass and time are number of data points and tessellation, respectively.*