Interactive Visualization of Particle Beams for Accelerator Design

Brett Wilson[1], Kwan-Liu Ma[1], Ji Qiang[2], and Robert Ryne[2]

[1] Department of Computer Science, University of California,
One Shields Avenue, Davis, CA 95616
[wilson, ma]@cs.ucdavis.edu
[2] Lawrence Berkeley National Laboratory,
One Cyclotron Road, Berkeley, CA 94720
[jQiang, RDRyne]@lbl.gov

From: P.M.A. Sloot et al (Eds.): ICCS 2002, LNCS 2331, pp. 352–361, 2002.

Abstract. We describe a hybrid data-representation and rendering technique for visualizing large-scale particle data generated from numerical modeling of beam dynamics. The basis of the technique is mixing volume rendering and point rendering according to particle density distribution, visibility, and the user's instruction. A hierarchical representation of the data is created on a parallel computer, allowing real-time partitioning into high-density areas for volume rendering, and low-density areas for point rendering. This allows the beam to be interactively visualized while preserving the fine structure usually visible only with slow point-based rendering techniques.

1 Introduction

Particle accelerators are playing an increasingly important role in basic and applied sciences, such as high-energy physics, nuclear physics, materials science, biological science, and fusion energy. The design of next-generation accelerators requires high-resolution numerical modeling capabilities to reduce cost and technological risks, and to improve accelerator efficiency, performance, and reliability. While the use of massively-parallel supercomputers allows scientists to routinely perform simulations with hundreds of millions of particles [2], the resulting data typically requires terabytes of storage space, and overwhelms traditional data analysis and visualization tools.

The goal of beam dynamics simulations is to understand the beam's evolution inside the accelerator and, through that understanding, to design systems that meet certain performance requirements. These requirements may include, for example, minimizing beam loss, minimizing emittance growth, avoiding resonance phenomena that could lead to instabilities, etc. The most widely used method for modeling beam dynamics in accelerators involves numerical simulation using particles. In three-dimensional simulations each particle is represented by a six-vector in phase space, where each six-vector consists of three coordinates (X, Y, Z) and three momenta (P_x, P_y, P_z). The coordinates and momenta are updated as the particles are advanced through the components of the accelerator, each of which provides electromagnetic forces that guide and

focus the particle beam. Furthermore, in high intensity accelerators, the beam's own self-fields are important. High intensity beams often exhibit a prounounced beam halo that is evidenced by a low density region of charge far from the beam core. The halo is responsible for beam loss as stray particles strike the beam pipe, and may lead to radioactiviation of the accelerator components.

2 Particle Visualization

In the past, researchers visualized simulated particle data by either viewing the particles directly, or by converting the particles to volumetric data representing particle density [4]. Each of these techniques has disadvantages. Direct particle renderings takes too long for interactive exploration of large datasets. Benchmarks have shown that it takes approximately 50 seconds to render 300 million points on a Silicon Graphics Infinite-Reality engine, and PC workstations are unable to even hold this much data in their main memory.

Volume rendering can provide interactive framerates, even for PC-based workstations with commercial graphics cards. In this type of rendering, the range covered by the data is evenly divided into voxels, and each voxel value is assigned a density based on the number of points that fall inside of it. This data is then converted into an 8-bit paletted texture and rendered on the screen as a series of closely-spaced parallel texture-mapped planes. Taken together, these planes give the illusion of volume. A further advantage of volume-based rendering is that it allows realtime modification of a transfer function that maps density to color and opacity, since only the palette for the texture needs to be updated [5]. However, there are also limitations. In order to fit in a workstation's graphics memory, the resolution is typically limited to 256^3 (512^3 for large systems). This, as well as the low range of possible density values (256), can result in artifacts, and can hide fine structures, especially in the low-density halo region of the beam.

Ideally, a visualization tool would be able to interactively visualize the beam halo of a large simulation at very high resolutions. It would also provide realtime modification of the transfer function, and run on high-end PCs rather than a supercomputer. This tool would be used to quickly browse the data, or to locate regions of interest for further study. These regions could be rendered offline at even higher quality using a parallel supercomputer.

To address these needs, our system uses a combined particle- and volume-based rendering approach. The low-density beam halo is represented by directly rendering its constituent particles. This preserves all fine structures of the data, especially the lowest-density regions consisting of only one or two particles that would be invisible using a volumetric approach. The high-density beam core is represented by a low-resolution volumetric rendering. This area is of lesser importance, and is dense enough so that individual particles do not have a significant effect on the rendering. The volume-rendered area provides context for the particle rendering, and, with the right parameters, is not even perceived as a separate rendering style.

3 Data Representation

To prepare for rendering, a multi-resolution, hierarchical representation is generated from the original, unstructured, point data. The representation currently implemented is an octree, which is generated on a distributed-memory parallel computer, such as the PC cluster shown in Figure 1. This pre-processing step is performed once for each plot type desired (since there are six values per point, many different plots can be generated from each dataset). This data is later loaded by a viewing program for interactive visualization.

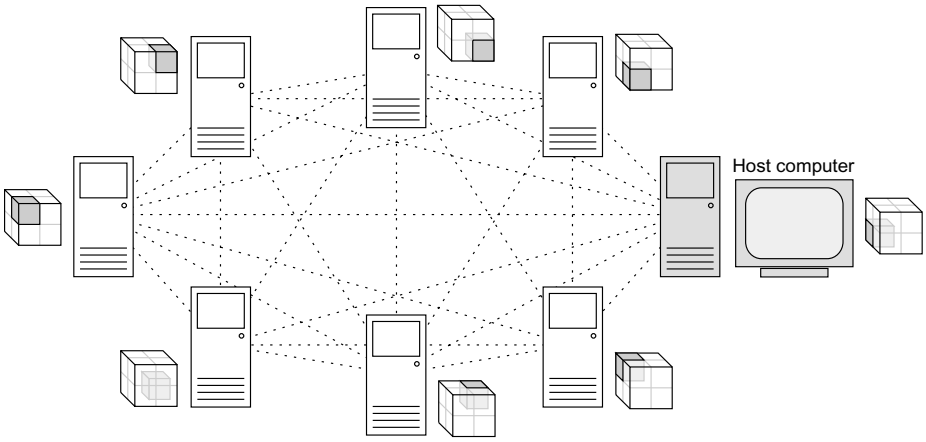

Fig. 1. The data is distributed to a parallel computer, such as a PC cluster, each processor of which is responsible for one octant of the data. After being read, the points are forwarded to the appropriate processor, which creates the octree for that section of data. Viewing is performed on one of the nodes with a graphics card.

The hierarchical data consists of two parts: the octree data, and the point data. At each octree node, we store the density of points in the node, and the minimum density of all sub-nodes. At the leaf octree nodes (the nodes at the finest level of subdivision), we store the index into the point data of the node's constituent points. The leaf nodes should be small enough so that the boundary between point-rendered nodes and volume-rendered nodes appears smooth on the screen. Simultaneously, the nodes need to be big enough to contain enough points to accurately calculate point density.

Since the size of the point data is several times the available memory on the workstation used for interaction, not all of the points can be loaded at once by the viewing program. Having to load points from disk to display each frame would result in a loss of interactivity. Instead, we take advantage of the fact that only low-density regions are rendered using the point-based method. High-density regions, consisting of the majority of points in the dataset, are only volume rendered, and the point data is never needed. Therefore, the points belonging to lower-density nodes are stored separately from the rest of the points in the volume. The preview program pre-loads these points from disk

when it loads the data. It can then generate images entirely from in-core data as long as the display threshold for points does not exceed that chosen by the partitioning program. For this reason, the partitioning program generates approximately as much pre-loaded data as there is memory available on the viewing computer.

4 User Interaction

The preview program is used to view the partitioned data generated by the parallel computer. As shown in Figure 2, it displays the rendering of the selected volume in the main portion of the window, where it can be manipulated using the mouse. Controls for selecting the transfer functions for the point-based rendering and the volume-based rendering are located on the right panel.

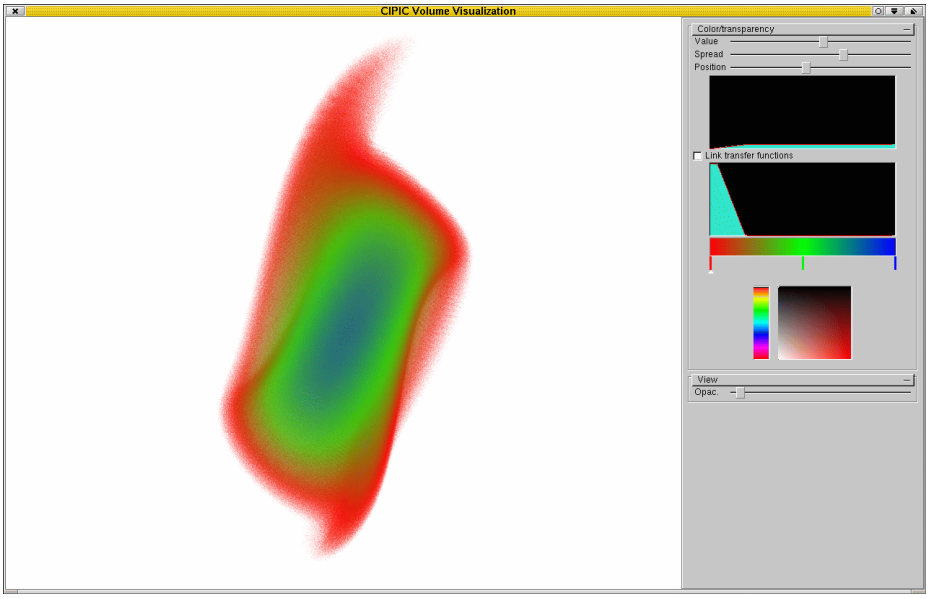

Fig. 2. The user interface, showing the volume transfer function (black box in the top right of the window) and the point transfer function (below it) with the phase plot (x, P_x, z) of frame 170 loaded. This image consists of 2.7 million points and displays in about 1 second on a GeForce 3.

The volume transfer function maps point density to color and opacity for the volume-rendered portion of the image. Typically, a step function is used to map low-density regions to 0 (fully transparent) and higher density regions to some low constant so that one can see inside the volume. The program also allows a ramp to transition between the high and low values, so the boundary of the volume-rendered region is less visible.

The point transfer function maps density to number of points rendered for the point-rendered portion of the image. Below a certain threshold density, the data is drawn

rendered as points; above that threshold, no points are drawn. Intermediate values are mapped to the fraction of points drawn. When the transfer function's value is at 0.75 for some density, for example, it means that three out of every four points are drawn for areas of that density. This allows the user to see fewer points if too many points are obscuring important features, or to make rendering faster. It also allows a smooth transition between point-rendered portions of the image and non-point-rendered portions. Point opacity is given as a separate control, a feature that can be useful when many points are being drawn.

By default, the two transfer functions are inverses of each other. Changing one results in an equal and opposite change in the other. This way, there is always an even transition between volume- and point-rendered regions of the image. In many cases, this transition is not even visible. The user can unlink the functions, if desired, to provide more or less overlap between the regions.

5 Rendering

The octree data structure allows efficient extraction of the information necessary to draw both the volumetric- and point-rendered portions of the image. Volumetric data is extracted directly from the density values of all nodes at a given level of the octree. Most graphics cards require textures to be multiples of powers of two, and the octree contains all of these resolutions pre-computed up to the maximum resolution of the octree, so extraction is very fast. These density values are converted into 8-bit color indices and loaded into textures. One texture is created for each plane along each axis of the volume [1]. So, a 64^3 texture would require $64 \times 3 = 192$ two-dimensional textures at a resolution of 64×64.

To draw the volume, a palette is loaded that is based on the transfer function the user specified for the volumetric portion of the rendering. This palette maps each 8-bit density value of the texture to a color and an opacity; regions too sparse to be displayed for the given transfer functions are simply given zero opacity values. Then, a series of planes is drawn, back-to-front, along the axis most perpendicular to the view plane, each mapped with the corresponding texture. The accumulation of these planes gives the impression of a volume rendering. While often the highest possible resolution supported by the hardware is used for rendering, we found that relatively low resolutions can be used in this application. This is because the core of the beam is typically diffuse, rendered mostly transparent, and is obscured by points. All images in this paper were produced using a volume resolution of 64^3.

In contrast to the volume rendering, in which only the palette is changed in response to user input, point rendering requires that the appropriate points from the dataset be selected each time a frame is rendered. Therefore, we want to quickly eliminate regions that are too dense to require point rendering. When displaying a frame, we first calculate the maximum density a node must have to be visible in the point rendering, based on the transfer function given by the user. Since each octree node contains the minimum density of any of its sub-nodes, only octree paths leading to renderable leaf nodes must be traversed; octree nodes leading only to dense regions in the middle of the beam need never be expanded.

Once the program decides that a leaf node must be rendered, it uses the point transfer function to estimate the fraction of points to draw. Often, this value is one, but may be less than one depending on the transfer function specified by the user. It then processes the list of points, drawing every n-th one. The first point drawn is selected to be a random index between 0 and n. This eliminates possible visual artifacts resulting in the selection of a predictable subset of points from data that may have structure in the order it was originally written to disk. Figure 3 illustrates the two regions of the volume regarding to the image generation process.

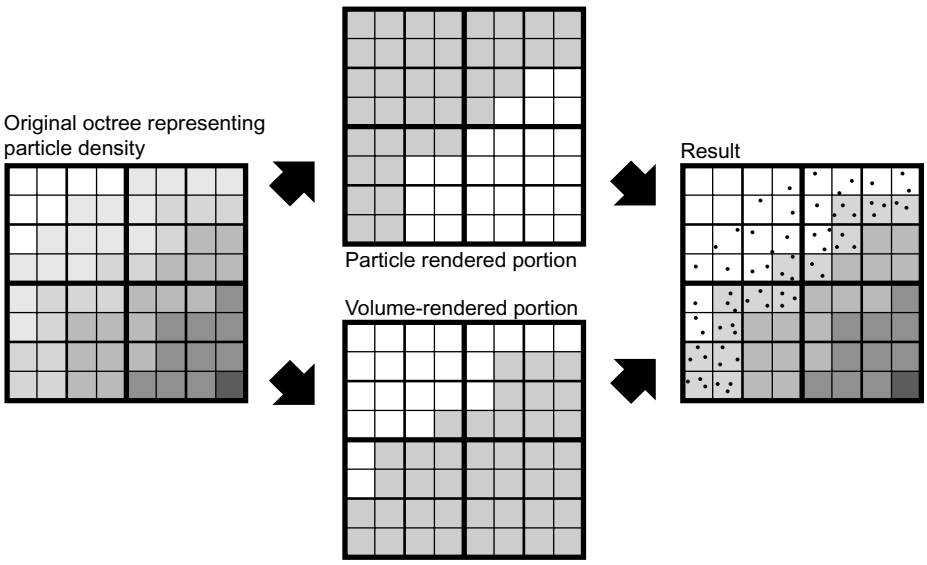

Fig. 3. The image is created by classifying each octree node as belonging to a volume-rendered region or a point-rendered region, depending on the transfer functions for each region (the regions can overlap, as in this example). The combination of the two regions defines the output image.

6 Results

The system was tested using the results from a self-consistent simulation of charged particle dynamics in an alternating-focused transport channel. The simulation, which was based on an actual experiment, was done using 100 million particles. Each particle was given the same charge-to-mass ratio as a real particle. The particles moving inside the channel were modeled, including the effects of external fields from magnetic quadrupoles and self-fields associated with the beam's space charge. The three-dimensional mean-field space-charge forces were calculated at each time step by solving the Poisson equation using the charge density from the particle distribution. The initial particle distribution was generated by sampling a 6D waterbag distribution (i.e.

a uniformly filled ellipsoid in 6D phase space). At the start of the simulation, the distribution was distorted to account for third-order nonlinear effects associated with the transport system upstream of the starting point of the simulation. In the simulation, as in the experiment, quadrupole settings at the start of the beamline were adjusted so as to generate a mismatched beam with a prounounced halo. The output of the simulation consisted of 360 frames of particle phase space data, where each frame contained phase space information at one time step.

Several frames of this data were moved onto a PC cluster for partitioning, although the data could have been partitioned on the large IBM SP that was used to generate the data. We used eight PCs, each was a 1.33 GHz AMD Athlon with one GB of main memory. A typical partitioning step took a few minutes, with most of the time being spent on disk I/O. The resulting data was visualized on one of the cluster computers equipped with an nVidia GeForce 3.

Figure 4 shows a comparison of a standard volumetric rendering, and a mixed point and volumetric rendering of the same object. The mixed rendering is able to more clearly resolve the horizontal stratifications in the right arm, and also reveals thin horizontal stratifications in the left arm not visible in the volume rendering from this angle.

Figure 5 shows how the view program can be used to refine the rendering from a low-quality, highly interactive view, to a higher-quality less interactive view.

7 Conclusions

The mixed point- and volume-based rendering method proved far better at resolving fine structure and low-density regions than volume rendering or point rendering alone. Volume rendering lacks the spatial resolution and the dynamic range to resolve regions with very low density, areas which may be of significant interest to researchers. Point-based rendering alone lacks the interactive speed and the ability to run on a desktop workstation that the hybrid approach provides.

Point-based rendering for low-density areas also provides more room for future enhancements. Because points are drawn dynamically, they could be drawn (in terms of color or opacity) based on some dynamically calculated property that the researcher is interested in, such as temperature or emittance. Volume-based rendering, because it is limited to pre-calculated data, can not allow dynamic changes like these.

8 Further Work

We plan to implement this hybrid particle data visualization method using the massively parallel computers and high-performance storage facility available at the Lawrence Berkeley National Laboratory. Through a desktop graphics PC and high-speed networks, accelerator scientists would be able to conduct interactive exploration of the highest resolution particle data stored.

As we begin to study the high resolution data (up to 1 billion points), the cost of volume rendering is not negligible any more. 3D texture volume rendering [6] will be thus used which offers better image quality with a much lower storage requirement.

Fig. 4. Comparison of a volume rendering (top) and a mixed volume/point rendering (bottom) of the phase plot (x, P_x, y) of frame 170. The volume rendering has a resolution of 256^3. The mixed rendering has a volumetric resolution of 64^3, 2 million points, and displays at about 3 frames per second. The mixed rendering provides more detail than the volume rendering, especially in the lower-left arm.

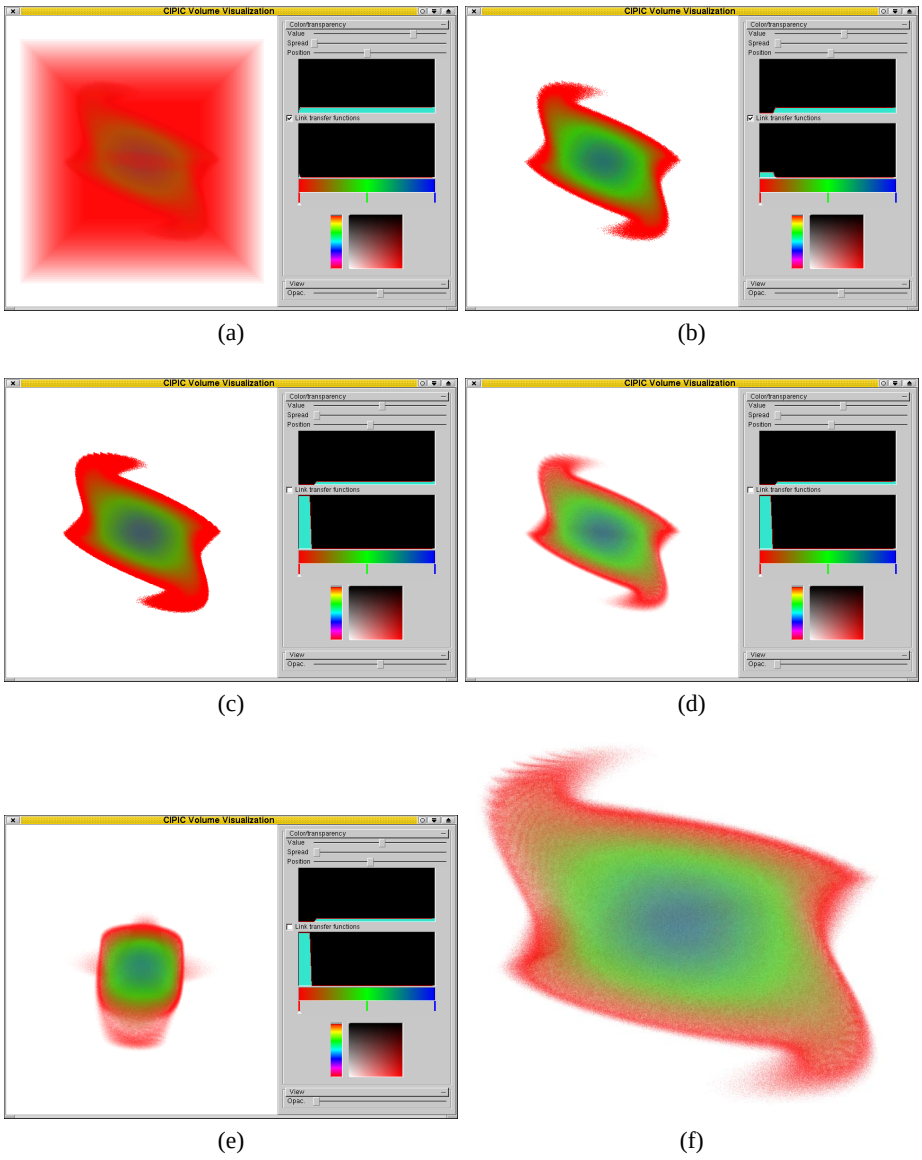

Fig. 5. A progression showing how exploration is performed. (a) Shows the initial screen, with a volume-only rendering. (b) The boundary between the high-density volume rendering and the low-density particle rendering has been moved to show more particles. (c) The transfer functions have been unlinked to show more particles while keeping the volume-rendered portion relatively transparent. (d) The point opacity has been lowered to reveal more structure. (e) The volume has been rotated to view it end-on. (f) A higher-resolution version similar to (d).

We will also investigate illumination methods to improve the quality of point-based rendering.

Acknowledgements

This work was performed under the auspices of the SciDAC project, “Advanced Computing for 21st Century Accelerator Science and Technology,” with support from the Office of Advanced Scientific Computing Research and the Office of High Energy and Nuclear Physics within the U.S. DOE Office of Science. The simulated data were generated using resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science under Contract No. DE-AC03-76SF00098. This work was also sponsored in part by the National Science Foundation under contracts ACI 9983641 (PECASE Award) and ACI 9982251 (LSSDSV).

References

1. B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction using Texture Mapping Hardware. In 1994 Workshop on Volume Visualization, October 1994, pp. 91–98.
2. J. Qiang, R. Ryne, S. Habib, V. Decyk, ”An Object-Oriented Parallel Particle-In-Cell Code for Beam Dynamics Simulation in Linear Accelerators,” J. Comp. Phys. vol. 163, 434, (2000).
3. J. Qiang and R. Ryne, ”Beam Halo Studies Using a 3-Dimensional Particle-Core Model,” Physical Review Special Topics - Accelerators and Beams vol. 3, 064201 (2000).
4. P. S. McCormick, J. Qiang, and R. Ryne. Visualizing High-Resolution Accelerator Physics. Visualization Viewpoints (Editors: Lloyd Treinish and Theresa-Marie Rhyne), IEEE Computer Graphics and Applications, September/October 1999, pp. 11–13.
5. M. Meissner, U. Hoffmann, and W. Strasser. Enabling Classification and Shading for 3d Texture Mapping Based Volume Rending Using OpenGL and Extensions. In IEEE Visualization ’99 Conference Proceedings, October 1999, 207–214.
6. A. Van Gelder, and U. Hoffman. Direct Volume rendering with Shading via Three-Dimensional Textures. In ACM Symposium on Volume Visualization ’96 Conference Proceedings, October 1996, pp. 23–30.