Title
DISTRIBUTED SCIENTIFIC VIDEO MOVIE MAKING

Authors
Johnston, W.E.
Hall, D.E.
Huang, J.

Publication Date
1988-03-01

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA, BERKELEY

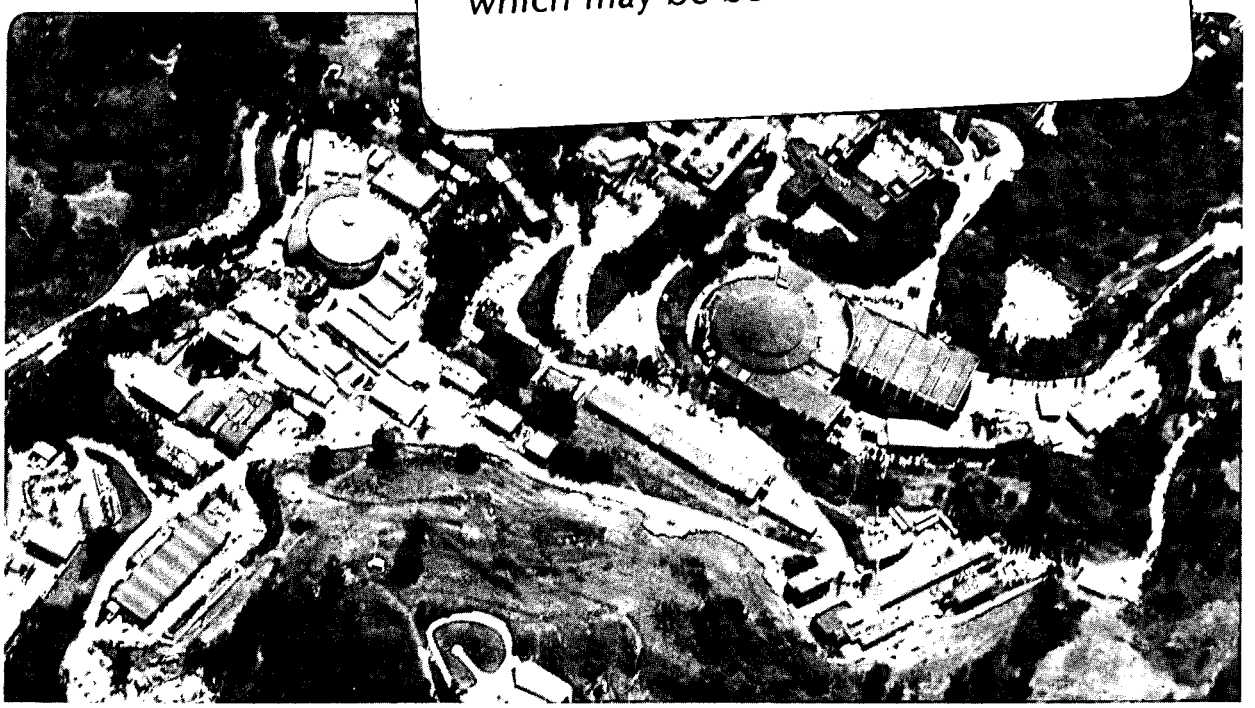
Information and Computing Sciences Division

To be presented at the IEEE Conference, Supercomputing 1988, Kissimmee, FL, November 14–18, 1988, and to be published in the Proceedings

Distributed Scientific Video Movie Making

W.E. Johnston, D.E. Hall, J. Huang, M. Rible, and D. Robertson

March 1988



Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

DISCLAIMER

Distributed Scientific Video Movie Making

W. E. Johnston, D. E. Hall, J. Huang, M. Rible, and D. Robertson*

Advanced Development Group
Lawrence Berkeley Laboratory
University of California
Berkeley, California, 94720

ABSTRACT

We describe a versatile, low cost, video movie making system for generating and displaying scientific graphics from remote supercomputers. The system makes video movies by single frame animation from the output of time dependent, numerical simulations typically done on supercomputers. The system uses extensive data compression to permit its use over wide area, as well as local area networks. The system demonstrates an easily used, elementary visualization capability for time dependent data in a heterogeneous, distributed computing environment.

1. Introduction

Most scientists access supercomputers remotely.** Supercomputing will probably always be a scarce and oversubscribed resource for the general scientific community. Usage is characterized by slow turn around, large quantities of data, and remote locations connected by wide area networks.

Supercomputer output may often be displayed as animated sequences that we will refer to as "scientific movies". These movies provide an insightful way to view time dependent results, such as flow fields and moving surfaces, and complex 3D structures that can be animated by rotation and translation.

We have developed a scientific movie system that is both easily used in the environment of remote supercomputing, and relatively low cost ($10K - $15K). The distributed graphics system has modules that are distributed among the supercomputer, a local workstation for some of the intermediate steps (optional), and a video animation controller workstation. The modules use Berkeley Sockets [1 and 2] and Sun Microsystem's Remote Procedure Call package (RPC) and External Data Representation language (XDR) [3], to effect interprocess communication (IPC). Using these, the movie system operates over the many networks that make up the TCP/IP Internet. This internet provides supercomputer access for a substantial fraction of the U.S. scientific community.

The system is written mostly in C, and has been demonstrated to run in a variety of environments including VAX VMS™, Sun OS™, 4.3 BSD Unix, Cray Unicos™ and CTSS.

2. Scientific Video Movies

Video graphics are different from pen plotter, terminal and film graphics. The spatial resolution of the NTSC video signal used for video recording is low compared to the more traditional media [4,5], while the color resolution is relatively high. These factors must be taken into account when designing a graphics representation for video tape. One does not count on producing the fine line drawings that are displayed, for instance, on laserprinters, but rather uses smoothly shaded areas that are easily reproduced on video tape. However, even with the relatively low spatial resolution of NTSC video, line and point drawings can still be useful if special care is taken in their design. Figure 1 shows sample frames from four movies made by using this system and supercomputer generated data. Each movie illustrates different data transfer and compression characteristics. (The original color movie frames are reproduced in black and white in Figure 1).

The system provides interfaces at several levels, including the compression and transfer of a frame buffer, 2D and 3D graphics primitives, and visualization modules such as 3D surface tessellation and particle advection.

The design goal of the system is to provide scientific visualization using state of the art graphics algorithms. A certain degree of "realism" is necessary, for instance, when presenting 3D geometric shapes to resolve any viewing hypothesis ambiguities. For example, the front of the torus in Figure 1d is clipped away so that the inside is visible. A reasonable level of sophistication in the rendering is needed to produce the realism necessary to be able to recognize that we are looking at the concave inner surface of the object, rather than the convex outside surface. The use of simple "flat" shading frequently produces an image in which the

* All of the authors can be reached via USMail at: Lawrence Berkeley Laboratory, Bldg. 50B, Rm. 3238, Berkeley, CA 94720. Email addresses are: wejohnston@lbl.gov, ...ucbvax!lbl-csam.arpa!johnston; hall@lbl-csam.arpa, huang@lbl-csam.arpa, max@lbl-csam.arpa, davidr@lbl-csam.arpa. The work presented in this paper is supported by the U.S. Department of Energy under contract DE-AC03-76SF00098. Any conclusions or opinions, or implied approval or disapproval of a company or product name are solely those of the authors and not necessarily those of The Regents of the University of California, the Lawrence Berkeley Laboratory, or the U.S. Department of Energy. Trademarks are acknowledged by TM.

** Only a handful of sites in the U.S. supply supercomputing to the general scientific community: National Magnetic Fusion Energy Center (NMFECC), National Center for Atmospheric Research (NCAR), and the NSF supercomputer centers (SDSC, PSC, JvNC, NCSA). The primary access to all of these centers is via wide area networks.

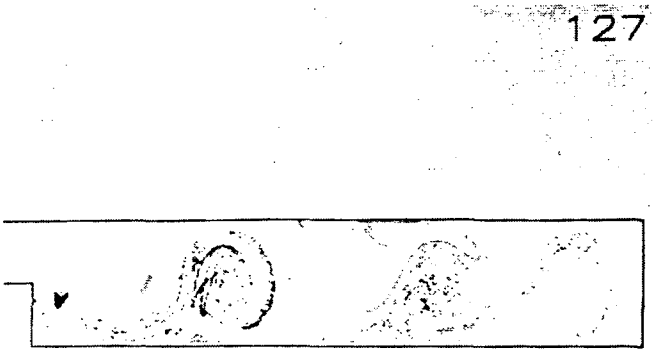


Figure 1a
Flow Over a Backward Facing Step - J. Sethian, UC Berkeley, A. Ghoniem, MIT

CBB 883-2578

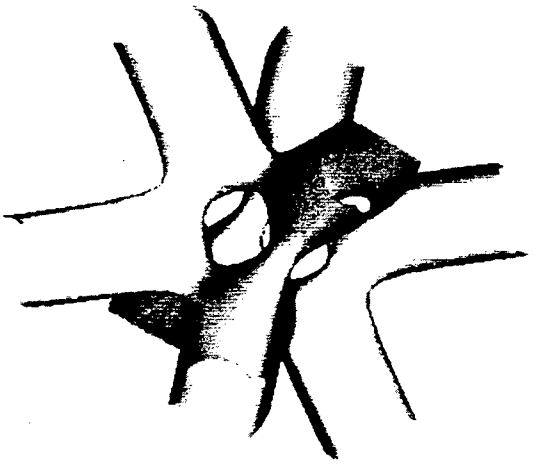


Figure 1c
Kummer Surface - E. Yeh, W. Johnston, D. Robertson and M. Rible, Lawrence Berkeley Laboratory

CBB 887-6918

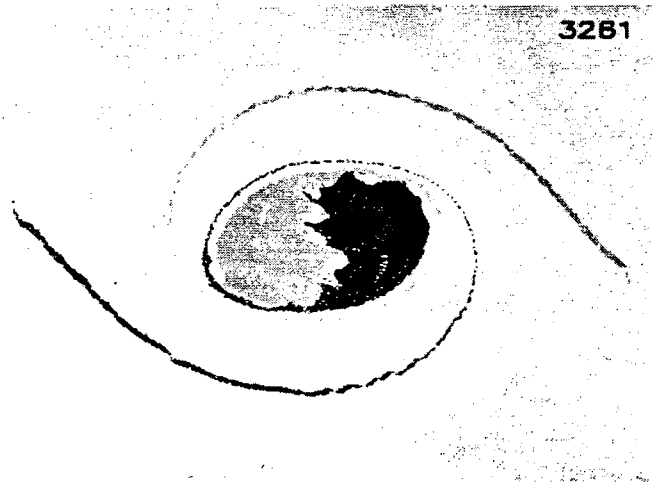


Figure 1b
Vortex Modeling - S. Baden, Lawrence Berkeley Laboratory

CBB 883-2580

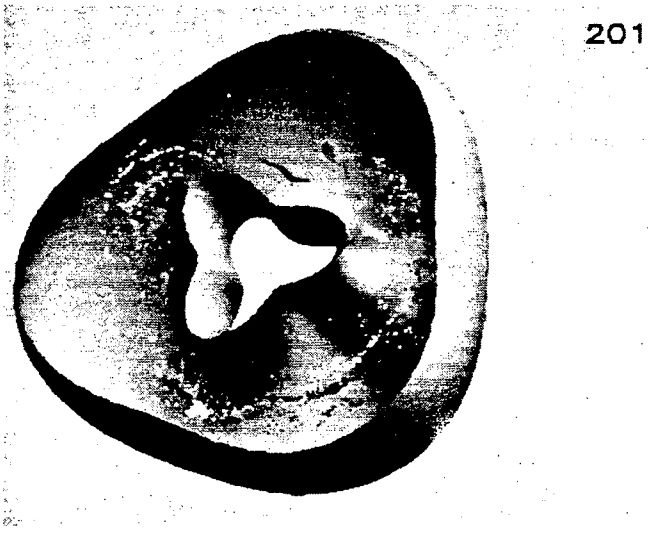


Figure 1d
3D Flow Through a Torus - C. Peskin and D. McQueen, Courant Institute, New York University, and D. Robertson, Lawrence Berkeley Laboratory

CBB 883-2584

concave-convex distinction cannot be made. On the other hand, for the Kummer surface in Figure 1c, the function is defined on a 100×100×100 grid and the tessellated surface for one function value results in about 100,000 polygons. With this many polygons flat shading is generally used, as the interpolation of Gouraud shading [18] is both expensive and not needed to improve the visualization. Our approach is to use the minimal shading and lighting algorithms necessary to accomplish an unambiguous representation. This reduces CPU and network demands while keeping the software volume manageable. See [6] for more information.

3. The Distributed Movie System

The video movie system software is a distributed application with several modules. The optimal distribution strategy for wide area networks is usually to select the partitioning that minimizes the data transported over the wide area network. However, this may be tempered by the relative cost of computing on supercomputers as opposed to workstations.

The modules that make up the system are illustrated in Figure 2. There are four main groups of modules: generating a numerical representation of some mathematical model (the application); converting the numerical representation to a graphics representation; converting the graphics representation to a raster image representation; and finally the display and recording of the image. These steps are connected pro-

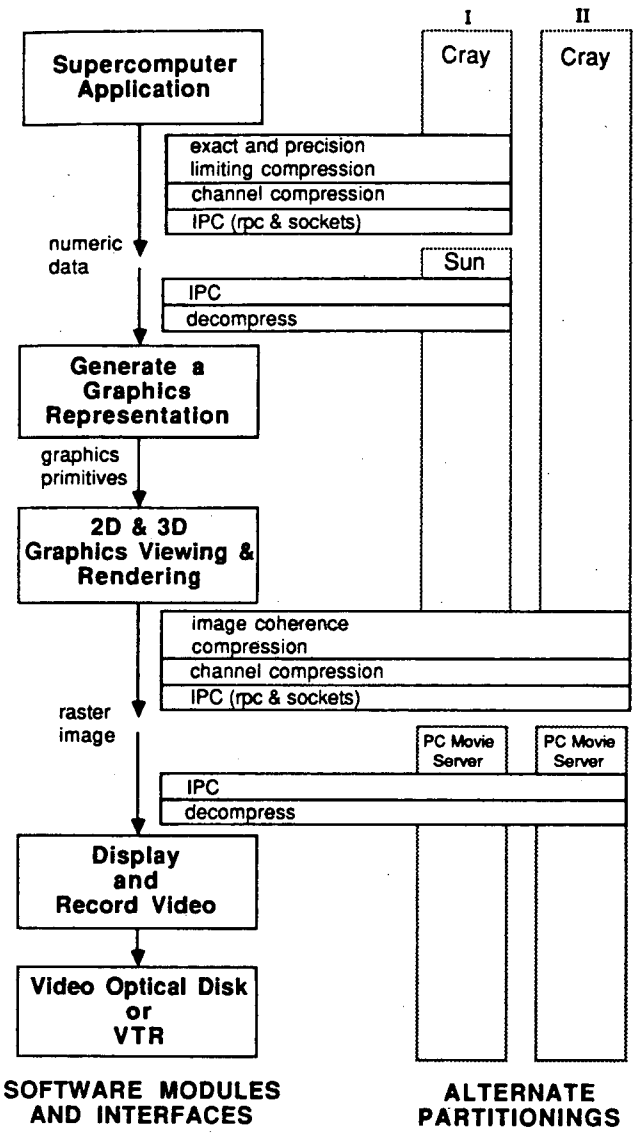


Figure 2
The Distributed Software Architecture

cedurally on a local system, or by data compression and transport to remote systems, depending on the distribution of the modules.

Though not formal "objects" in the Smalltalk sense [7], these modules are designed to have many of the characteristics found in an object oriented programming environment. We feel that this view of distributed computing aids in implementing distributed applications in a heterogeneous environment.

3.1 The Numerical Representation: Generating the Data

The supercomputer application is a numerical implementation of a mathematical model. Typical applications include modeling of 2D and 3D flow fields, wave propagation, 3D surface topology studies, and various simulations (e.g. accelerator design studies) that produce 2D and 3D particle position fields. These models produce a numeric representation that is converted to a graphics representation (either on the local system or on a remote workstation after the numeric data is sent over a network). If the data is sent over a network, it is compressed by limiting its precision and removing redundancy as described in section 7. Table 1 gives typical *per frame* volumes of data for several applications at various points in the pipeline illustrated in Figure 2 (partitioning II).

3.2 The Graphics Representation: Conversion to Graphics Primitives

A graphics representation algorithm converts a numeric representation of the model output into the graphics primitives (points, lines, polygons and text) that describe an image. The process is not graphics *per se*, but algorithms that transform the numeric representation to a geometric form that can be displayed as an image. Typical examples of graphics representations algorithms are: level curve following through a 2D scalar field ($z=f(x,y)$) to produce "contour plots"; advecting particles through a flow field to obtain tracer particle positions that can be plotted (as in Figure 1a and 1b); and tessellating the level surface of a 3D scalar field (Figure 1c), or some geometric description of the borders of an area of flow, to get a polygonal representation that can be rendered as a shaded surface (Figure 1d).

Table 1
Data Volumes per Frame for Typical Applications

Figure number and data type	Numeric representation of the data	Graphics representation type, and data volumes of the resulting graphics primitives	Compressed raster image size (uncompressed image size is 410KB)
1a; 2D flow field (u, v) on 100×11 grid	17KB	5000 tracer particles + 5000 attributes = 90KB	2.5KB (164:1)
1b; 2D flow field on 13,000 grid points	310KB	13000 tracer particles + 13000 attributes = 230KB	6.2KB (66:1)
1c; 3D scalar field on 100×100×100 grid	4000KB	Tessellated level surface: 110,000 polygons = 1800KB	7.8KB (52:1)
1d; 3D flow field on 100×100×100 grid	24000KB	2000 tracer particles + 4500 polygons = 100KB	19KB (21:1)

The volume of data associated with an op-code and coordinate form of the graphics primitives depends on the graphics representation algorithm. For example, particle advection converts a flow field description to positions of a set of tracer particles. The number of particles is independent of the flow field size. When tessellating a sequence of surfaces that evolve in time, the number of polygons in a surface at each time step is frequently roughly proportional to the number of points on the outer surface of the grid. Table 1 gives some examples of data volumes generated by graphics representation algorithms.

The graphics primitives produced by this step are usually sent by a procedural interface to a rendering module on the same system, though they are sometimes sent over the network.

3.3 Graphics Rendering: Conversion to a Raster Image Representation

The graphics rendering process converts the primitives generated by the graphics representation into a raster image. In the distributed movie system, graphics primitives are rendered by scan conversion into a software frame buffer located in main memory. For 3D primitives the hidden surface, or visibility problem, is solved by using a "z" or "depth" buffer approach. This results in the final image having visual priority that corresponds to the geometric priority of the scene and eye point. The rendering of polygons additionally entails shading (varying the pixel color across the polygon surface to present the appearance of a curved surface) according to some interpolation model, and assignment of color according to some lighting model (calculating the color of a pixel based on the position of the light source and glossiness of the surface).

The size of the software frame buffer is fixed according to the targeted output device. In the case of the current hardware of the distributed movie system [4] this is 400×512×3×5 bits ≅ 400K bytes. This software frame buffer is compressed and sent over the network to the video animation controller workstation.

3.4 Compressing the Raster Representation

The utility of compression is obvious in wide area networks. Sending 400KB/frame is currently out of the question. What is less obvious is that compression is also advantageous in local area networks. Typical 4 MIP workstations can send about 200 packets, or 300K bytes, per second among themselves on an Ethernet. This is well below the Ethernet bandwidth and packet rate limits. The limiting factor is the CPU time spent in the "protocol stack". For a slower processor, like the PC used for the animation workstation in our system, the maximum packet rate is substantially lower than 200 per second (more like 30 packets/second). Therefore, compression that reduces the number of packets per graphics frame will result in a net increase in the throughput, provided that the PC processor does less work decompressing the data than it would do to handle the packets of uncompressed data. Variations of the Block Truncation Coding (BTC) compression algorithms [8 and 10] provide this tradeoff. For BTC most of the work is in the encoding. The decoding process is just a table lookup to reconstruct the compressed image into the hardware frame buffer. This technique reduces the number of packets to be processed by the PC animation controller workstation without adding any significant overhead on the PC. The result is noticeably higher throughput, even on an Ethernet.

Over a wide area network, bandwidth dependence is the limiting factor. When using wide area networks, further compression of the image is obtained by processing the BTC compressed image through an adaptive Huffman compression algorithm [9]. This algorithm is much more symmetric than BTC, and takes as much CPU time to decode as it does to encode. Therefore, there is a network bandwidth break-even point above which the PC CPU time to decode is greater than the time to send an uncompressed BTC image over the network. We believe that the break-even point is a few hundred Kbits/second of network bandwidth for an IBM PC/AT™. However, this break even point is pushed to above 10 Mbits/second by using a 68020 co-processor in the PC to decode the Huffman coding. Therefore, compression pays off even in high speed local area networks where both BTC and Huffman coding are used to advantage.

3.5 The Video Animation Controller Workstation

The animation controller workstation accepts raster data from the network, and generates and records a video signal one frame at a time. The hardware consists of an IBM PC/AT™ (or clone), an Ethernet controller used for TCP/IP based IPC, a video frame buffer for reconstructing the raster image and generating the video signal, a single frame animation controller and a video tape recorder (VTR) (or alternatively a video-optical disk) for recording, and optionally a 68020™ co-processor board for doing the Huffman decompression. The hardware configuration is discussed in more detail in [4].

The PC runs as a remote procedure call server whose only task is to receive compressed images from the graphics workstation or remote supercomputer and record them on video tape or video disk. There is no user interaction with the PC server. It is just a peripheral hardware controller serving the system that generates and sends the compressed images. The PC server looks like any other RPC server, except that it will only accept one connection at a time, and there is no portmapper function [3]. This implies that the user must know both the Internet address of the video animation controller workstation, and the (fixed) port that the RPC server responds to. Given the role that it plays (running video recorders that record frame-at-a-time movies) the "single client" PC server is a useful paradigm.

In the case where a video-optical disk is used instead of the VTR, the PC also serves as a simple, but useful and convenient, video editing system. The movie clips from the supercomputer are recorded frame-at-a-time onto the optical disk, and title frames are usually added later, anywhere on the disk. (Titling may be generated from a front end system, with a paint program on the PC, or simply by aiming a video camera at laserprinter output and recording a single analog frame on the optical disk.) The optical disk has a simple set of commands that allow a sequence of frames to be played at

any speed, forward or backward, and separate sequences of frames to be played in any order. The movie making scenario is to: generate the graphics movie clip; put a titling frame on the video-optical disk; and write a script on the PC to compose the movie. A typical script contains instructions to: play the title frame for a certain number of seconds; seek to the graphics clip; hold the first frame for 10 seconds; play the remaining frames at 1/3 speed (10 frames/second animation); and hold the last frame for a few seconds. To achieve smooth animation, the graphics frames must be recorded in sequence on the video-optical disk, since the disk cannot seek to "out of sequence" frames fast enough to generate smooth video. When the user has constructed a script that produces a satisfactory movie, a VHS recorder is connected to the output of the optical disk, the script is loaded and executed, and the resulting video is recorded to produce a scientific animation that can be viewed on a home VHS player.

4. Partitioning a Distributed Application

We had several design goals related to the distributed aspects of the system. First the modules had to be easy to distribute. This meant we had to design the primary interfaces to be implementable on a variety of systems, and minimize our use of non-standard operating system functions. Second, whenever possible we wanted to take advantage of the coarse grained parallelism afforded by pipelines. Therefore, our algorithms were implemented to perform local operations on data streams rather than to read an entire data set before beginning processing. Third, we hoped eventually to compare our data stream design to an object oriented design. Therefore, all modules were instrumented for performance monitoring.

The movie making process is typically distributed across three different systems (partition I in Figure 2). The supercomputer application generates data (e.g. flow field vectors) that are compressed and sent to a graphics workstation. The graphics workstation environment is used to design and debug the graphics representation for a movie. The graphics workstation then produces the frames of the movie. These raster images are compressed and sent to the animation controller workstation. The modules are all tied together by remote procedure call based collections of servers and clients.

Partition II of Figure 2 has all the modules executing on the Cray up to, and including, the generation and compression of the software frame buffer. This partition is used when it is impractical to move the application data off the Cray. For instance, in making a movie of a 3D flow field, the vector field data may easily amount to 30 to 50 Mbytes per frame. Even the advected particles can be more than 100K bytes per frame. The compressed movie frame is usually less than 20K bytes (see Table 1), an amount that is practical to send over current wide area networks.

5. Internetworking

The established TCP/IP networking infrastructure is a versatile and powerful tool for distributed applications geographically dispersed around the country.

An internet is a network of networks. It accommodates multiple, diverse underlying hardware technologies, network technologies, and operating systems while providing a uniform set of conventions for usage. The distributed movie system takes advantage of the extensive internet technology of the TCP/IP Internet. The success of internetworking is that our application sees and uses only one, uniform interface: The Internet protocols and Berkeley sockets [2].

6. Numeric and Image Data Compression

Data compression is an important aspect of wide area network based distributed applications. Compression can be divided into two types: entropy reduction, an irreversible compression since some (maybe insignificant) information is lost; and redundancy reduction, a reversible compression technique that tries to identify redundant information and encode it more efficiently. We use both types of compression, frequently together, in the movie system.

Synthetic raster images generated by computer graphics have an enormous amount of redundancy. There is typically high spatial coherence in the image, and high temporal (frame to frame) coherence. To compress these images, the system first applies entropy reduction techniques. Block truncation coding (BTC) [8,10] reduces the total number of colors needed to represent an image by reducing the colors in a block of pixels to two "best" representatives. A further compression is achieved by limiting the choice of possible representative colors to those of a short table whose index is then used to represent the colors in the blocks. Heckbert's median cut algorithm is used to populate the table with a set of colors that best reproduces the image. See [11]. One effect of this encoding is that areas of an image that appear different may result in the same block code, thus increasing the redundancy. This entropy reduced form is then encoded with a redundancy reduction algorithm such as Lemple-Ziv coding [9]. Using these techniques in combination, the overall compression of synthetic raster images can be substantial. Averaged over the length of a movie we typically see 20:1 - 60:1 compression for complex, shaded 3D images that fill most of the frame (e.g. Figure 1c-d), and 100:1 - 200:1 for simpler 2D images (e.g. Figure 1a-b).

The traditional wisdom is that floating point numbers from numerical simulations do not compress. Indeed, this can be seen to be true by applying an adaptive Huffman coding to a collection of floating point numbers, where one seldom sees greater than 1.3:1 - 1.5:1 compression. However, if we accept a limiting of precision then we can compress floating point numbers. To compress these numbers we use what amounts to a sequential quantization entropy reduction technique similar in principle to differential pulse code modulation [12]. We quantize the floating point numbers, limit their dynamic range and reduce them to integers. We then take differences between adjacent integers to further reduce the size requirement. In addition to reducing the word size, these techniques tend to generate repeated patterns that are then encoded by redundancy reduction techniques. The amount of information lost by this scheme is controlled primarily by specifying the quantization. It has been our experience that quantization to five digits, and limiting dynamic range to five orders of magnitude, is adequate for the graphics representation algorithms that we use. Using these techniques we typi-

cally achieve 12:1 - 15:1 compression of numeric data. (See [13] for more information.)

In Figure 2 "channel compression" is a term that we use to describe a type of redundancy reduction algorithm that compresses an unstructured byte stream. Channel compression is frequently used as the last step of a multistep compression process before that data is handed off to the network transport (or disk storage) functions.

7. The Structure of the User Interface

The typical user of the movie system is a physicist/programmer who needs to produce a movie of the output of a mathematical model. The basic user interface to the distributed movie system is the rendering module. This module implements a low level, GKS like interface for 2D viewing and graphics [14], and a SIGGRAPH Core like interface for 3D viewing and graphics [15]. To the 3D interface we have added a more intuitive set of 3D viewing controls [16]. This interface also has a polygon primitive with the necessary provisions for shading and lighting. To the basic depth buffer we have added a mechanism that allows clipping of a fixed portion of the object, as opposed to the usual view volume clipping. This is illustrated by the clipped torus of Figure 1d. The display of complex 3D objects, such as those illustrated in Figure 1c and 1d, is facilitated by the use of a routine that varies the view position from frame to frame in a way that results in a smooth yaw and pitch rotation. This provides a slow, continuously changing view of the 3D object [6].

Layered on top of this interface are two graphics representation algorithms that are general enough to be considered part of the system. The first is an algorithm to advect particles through a rectangular region of 2D flow. The user interface to this algorithm is where, and how many particles, to inject into the flow field, and the specification of (u,v,x,y) defining the flow. No provision is made for at the interface to change the boundary conditions of laminar flow. This algorithm was used to make the movie illustrated in figure 1a.

The other representation algorithm is Lorensen's Marching Cubes [17]. This algorithm provide a mechanism for displaying the level surfaces of a 3D scalar field of arbitrary complexity. We have used this to explore mathematical functions by displaying a sequence of surfaces $f(x,y,z)=c_1,c_2,c_3,\cdots$ as a movie, as well as showing a single level surface evolving in time for applications such as flame front propagation studies. The user interface to this algorithm entails specification of a 3D grid of function values, and the function value to be displayed (as well as the color, light source position, viewing position, etc.). The Kummer surface in Figure 1c was produced using this algorithm.

In addition to the graphics, there is a user interface for numeric data compression. We have attempted to identify a few prototypical "types" of data, and to provide user callable functions to compress this data. We have, for example, routines that accept 2D particles and attributes, 2D flow fields (u,v,x,y), and 3D scalar fields (function values on a 3D grid). In each case the user supplies the data, and the precision that needs to be recovered after decompression. The compressed data stream is written to a network stream or disk file.

8. Conclusions

From the success of this system we conclude: 1) Distributed computing among remote supercomputers and local workstations is a viable technique even over wide area networks. 2) Sophisticated graphics techniques and displays can be made available to the general scientific community by leveraging on the low cost of home video equipment. 3) Compression is an essential part of distributed computing.

References

[1] S. Leffler, R. Fabry, W. Joy, P. Lapsley, S. Miller, C. Torek, "An Advanced 4.3BSD Interprocess Communication Tutorial," Computer Science Research Group, University of California, Berkeley, CA, 94720, 1986.

[2] D. Comer, *Internetworking With TCP/IP: Principles, Protocols, and Architecture*, Prentice Hall, 1988.

[3] Sun Microsystems, Inc., "Networking on the Sun Workstation," Report 800-1324-03, Mountain View, CA, 1986.

[4] W. Johnston, D. Hall, F. Renema, D. Robertson, "Principles and Techniques for Low Cost Computer Generated Video Movies," LBL-22330, University of California, Lawrence Berkeley Laboratory, Berkeley, CA, 1987.

[5] D. Fink, Editor, *Color Television Standards: Selected Papers and Records of the National Television System Committee*, McGraw-Hill, 1955.

[6] D. Robertson, "Use of a Distributed Movie Making System for Presentation of Fluid Flow Data," San Francisco State University, San Francisco, CA, (Masters Thesis - available as LBL-25036 from Lawrence Berkeley Laboratory), 1988.

[7] A. Goldberg, D. Robson, *Smalltalk-80: The Language and Its Implementation*, Addison Wesley, 1983.

[8] G. Campbell, T. DeFanti, J. Frederiksen, S. Joyce, L. Leske, J. Lindberg and D. Sandin, "Two Bit/Pixel Full Color Encoding," *Computer Graphics*, vol. 20, no. 4, 1986. (Proceedings ACM SIGGRAPH, 1986)

[9] T. Welch, "A Technique for High Performance Data Compression," *IEEE Computer*, vol. 17, no. 6, June, 1984.

[10] N. Texier, W. Johnston, D. Robertson, "Encoding Synthetic Animated Images," LBL-24236, University of California, Lawrence Berkeley Laboratory, Berkeley, CA, 1987.

[11] P. Heckbert, "Color Image Quantization for Frame Buffer Display," *Computer Graphics*, vol. 16, no. 3, 1982. (Proceedings ACM SIGGRAPH, 1982)

[12] T. Lynch, *Data Compression*, Van Nostrand Reinhold, 1985.

[13] J. Huang, "Numeric Data Compression for Graphics," San Francisco State University, San Francisco, CA, (Masters Thesis - available as LBL-25037 from Lawrence Berkeley Laboratory), 1988.

[14] G. Enderle, K. Kansy, G. Pfaff, *Computer Graphics Programming: GKS, 2ed.*, Springer-Verlag, 1987.

[15] Graphics Standards Planning Committee, "Status Report of the Graphics Standards Planning Committee," *Computer Graphics*, vol. 13, no. 3, 1979.

[16] BJ Wishinsky, W. Johnston, "A Simplified Interface for SIGGRAPH Core Viewing," LBL-25038, University of California, Lawrence Berkeley Laboratory, Berkeley, CA, 1987.

[17] W. Lorensen, H. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, vol. 21, no. 4, 1987. (Proceedings ACM SIGGRAPH, 1987)

[18] J. Foley, A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, 1982.

LAWRENCE BERKELEY LABORATORY
TECHNICAL INFORMATION DEPARTMENT
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720