LBL-26796
UC-405

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

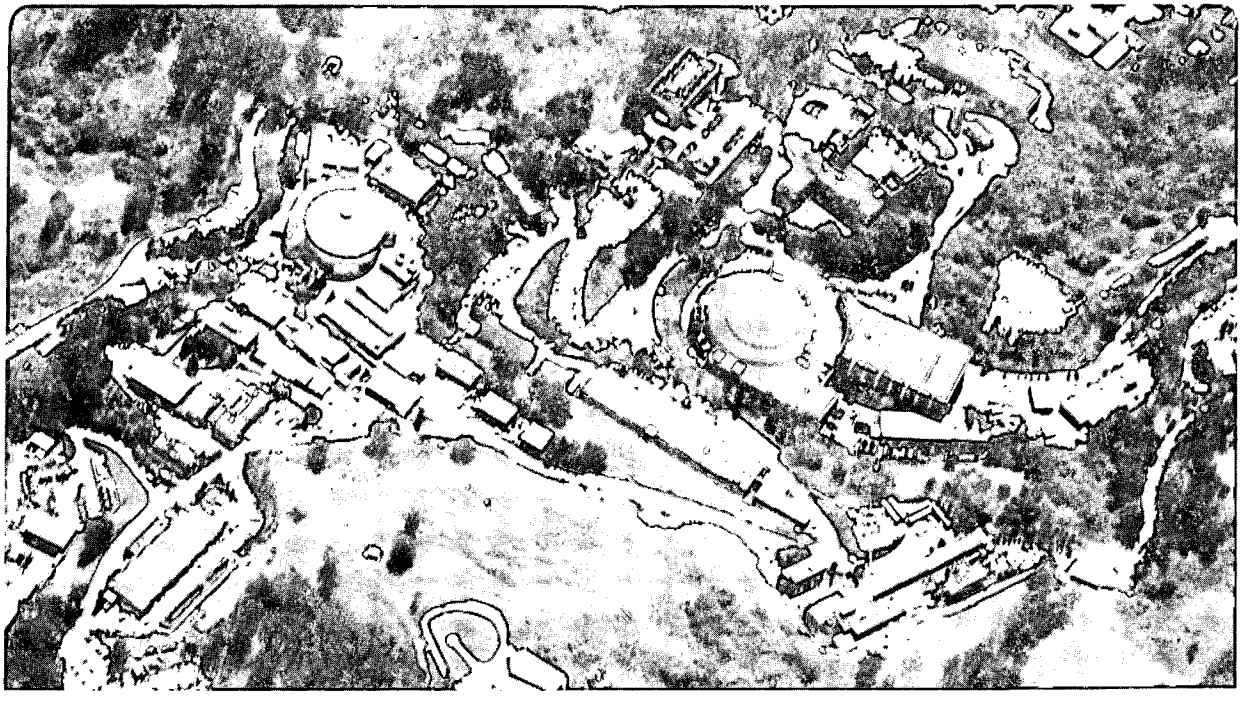
Information and Computing Sciences Division

Scry: A Distributed Image Handling System

Version 1.1

D.W. Robertson, W.E. Johnston, T.-J. Chua, J. Huang,
F. Renema, M. Rible, N. Texier, and B.J. Wishinsky

April 1989



LOAN COPY
Circulates
for 2 weeks
Bldg. 50 Library.
LBL-26796
Copy 2

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

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

April 6, 1989

LBL-26796

Scry: A Distributed Image Handling System

Version 1.1
April 1989

David W. Robertson, William E. Johnston, Teck-Joo Chua,

James Huang, Fritz Renema, Max Rible, Nicole Texier,

and BJ Wishinsky

Advanced Development Projects
Information and Computing Sciences Division
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, CA 94720

D. W. Robertson, W. E. Johnston, J. Huang, F. Renema, and M. Rible can be reached via USMail at: Lawrence Berkeley Laboratory, Bldg. 50B, Rm. 3238, Berkeley, CA 94720. Email addresses of those at LBL are: davidr@csam.lbl.gov,johnston@csam.lbl.gov,huang@csam.lbl.gov, fritz@csam.lbl.gov, and max@csam.lbl.gov

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.

Scry 1.1 README file

Scry is a distributed image handling system that provides both image compression and transport on local and wide area networks, and a collection of higher level graphics and scientific visualization functions. The system can be distributed among workstations, between supercomputers and workstations, and between supercomputers, workstations and video animation controllers. The system is most commonly used to produce video based movie displays of time dependent data and complex 3D data sets, and to handle the images resulting from certain image processing operations.

In the scientific visualization process, the system is typically used in one of two configurations. In the first, the supercomputer application generates data (e.g. flow field vectors) that are compressed and sent to a graphics workstation over a wide area network. The graphics workstation environment is used to design and debug the graphics visualization for a movie. Finally, the raster images that represent the frames of the movie are compressed and sent over a local area network to the video animation controller.

Alternatively, the data, and even its representation in terms of graphics primitives, may be too voluminous to reasonably transport across a network. In this case the visualization, rendering and image compression are all done on the supercomputer. The compressed images are then sent over a wide area or local area network to a user workstation at the local site.

The image servers, that is the local systems that receive the image, can be either a window based workstation, such as a Sun color workstation using SunView, or a PC based animation controller. All of the image servers present the same interface to the client programs. A typical use of the Sun based server, for example, is to have a graphics window which displays the images as they come in from the remote supercomputer, while storing the compressed form of the image on disk for later video recording or preview. In the case of the PC server the images are recorded in video format, either on tape or video-optical disk.

Layered on top of the raster image interface are several visualization algorithms that are of general interest. One of these is is Lorensen's Marching Cubes [SIGGRAPH, 1987]. 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, the function value to be displayed, and the presentation details of color, light source position, viewing position, etc.

The system has been used to make a number of movies for Lawrence Berkeley Laboratory scientists. Scry led directly to new insights from scientific data because of the ease with which the system generates movies due to a simple software interface, the ubiquitous availability of video technology, and to the rapid turnaround between generating and viewing movie "clips".

References:

W. E. Johnston, D. E. Hall, J. Huang, M. Rible, and D. Robertson. "Distributed Scientific Video Movie-Making". Proceedings of the Supercomputing Conference 1988 (The Computer Society of the IEEE). Also available as LBL-24996, University of California, Lawrence Berkeley Laboratory, Berkeley, CA (1988).

William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm 4 (July 1987), 163-169.

Additional references can be found in the man pages.

Note on this Distribution:

This distribution consists of the source code and documentation needed to build both the Scry servers and clients. The development of Scry is supported by the U.S. Department of Energy, Energy Research Division, under contract DE-AC03-76SF00098.

This is the first revision of the original Scry distribution. It now runs under UNICOS on Cray's (as well as under UNIX on Sun's and Vaxen), with exceptions noted in the man pages. The main additions are an implementation of Lorensen and Cline's marching cube algorithm (for reference see above), the capability for 3D viewing, and an expanded Anima. There has been little change to the original material except for Anima.

Scry is available by anonymous ftp (login: "anonymous", password: "guest") from csam.lbl.gov (128.3.254.6) in *pub/scry.tar.Z* (a compressed tar file, so don't forget to set binary mode in ftp). Be aware that the compressed file is just over 1 megabyte. Once on your machine, run uncompress on *scry.tar.Z*, and extract the files using

```
tar xvf scry.tar scry
```

Besides the tar file this distribution is contained in, there is an additional file, *scry.trouble*, also contained in the same directory on csam.lbl.gov. This file contains hints on dealing with problems people have encountered, and bug fixes as they become available.

We invite your comments and suggestions about this code. For further information contact:

Bill Johnston, (wejohnston@lbl.gov, ...ucbvax!lbl-csam.arpa!johnston)
or
David Robertson (dwrobertson@lbl.gov, ...ucbvax!lbl-csam.arpa!davidr)
Advanced Development Group
MS 50B/3238
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, CA 94720

There are a number of UNIX-style man pages that describe the various modules and applications that make up Scry. The order to read the documents is:

scry.l -
an overview

scry_client.l -
a more detailed look at the client strategy

scry_pc_server.l -
user documentation on the PC animation server

scry_sun_server.l -
user documentation on the windowing (Sun) workstation-based server

anima.l -
user documentation on the animation playback editor

scry_pc_config.l -
comments on the PC animation controller hardware configuration and porting considerations

greyrgb.l -
tests low-level client

test_c_gks.l -
2D GKS tester

grid.l, molecules.l -
3D sample programs

woggle.l, surfmov.l, march.l -
3D applications

scry_3D.3, scry_3D_control.3, scry_3D_ctm.3, scry_3D_proj.3, scry_3D_obj.3, scry_3D_render.3 -" 3D viewing module

scry_gks.3, scry_gks_control.3, scry_gks_prims.3, scry_gks_xforms.3 -
Two versions: the 2D graphics interface library, and support for 3D

scry_scan3d.a.3 -
3D scan conversion library

scry_scan2d.a.3 -
2D scan conversion library

scry_send_frame.3 -
the image compression and transport library that is used by the Scry GKS layer

scry_client.a.3 -
the image compression and transport library that is used by a user application

The client programs are built upon several modules which are described in the man pages in the directory *scry/man*. 2D clients are built on a GKS module, a 2D scan conversion module, and a module at the RPC level. 3D clients have an additional, 3D viewing module. **(Elements in brackets are under development and will be provided in a future distribution.)** The PC server program is divided into two modules: the server itself, and the modified Sun RPC library it is built on top of.

The software provided is contained in subdirectories, corresponding to each module, under the directory *scry*. It is written mostly in C; part of the 3D viewing module is written in Fortran.

scry	root directory
scry/libs	place libraries are put after modules compiled
scry/samples	sample client programs (**scry_client**(1),**woggle**(1) **test_c_gks**(1), **greyrgb**(1), **grid**(1), **molecules**(1)
scry/samples/data	data for sample client programs
scry/surfmov	marching cubes and rendering program (**surfmov**(1))
scry/marching	marching cubes library (**march**(1))
scry/view3d	3D viewing module (**scry_3D**(3))
scry/gks	GKS routines (**scry_gks**(3))
scry/scandrv	software frame buffer driver (**scry_gks2d.a**(3))
scry/scandrv/scan3d	3D scan conversion (**scry_scan3d.a**(3))
scry/scandrv/scan2d	2D scan conversion (**scry_scan2d.a**(3))
scry/scandrv/client	image transmittal (**scry_client.a**(3))
scry/pcserver	PC server (**scry_pc_server**(3))
scry/pcrpc	modified Sun RPC library (**scry_pc_server**(3))
scry/sunserver	Sun server (**scry_sun_server**(3))
scry/anima	Anima (**anima**(L))

All files except those in *scry/pcserver* and *scry/pcrpc* (for the PC server), and *scry/sunserver* (for the Sun server) should be set up on the client machine; the files in the latter directories should be installed on the server. Sample Makefiles for each machine type are provided in each directory to make the appropriate module libraries. Doing **make all** with the appropriate Makefile in *scry/samples*, i.e. *Makefile.cray* for the Cray, makes all the client libraries and generates the proper format data files for that machine. Separate commands also make the sample programs. On the Cray, it will be necessary to modify the Makefile and create two directories in a temporary partition to hold the rather large executables and the binary data files.

---------- Begin Copyright notice ----------

The Scry system is copyright (C) 1988, 1989, Regents of the University of California. Anyone may reproduce "Scry", the software in this distribution, in whole or in part, provided that:

(1) Any copy or redistribution of Scry must show the Regents of the University of California, through its Lawrence Berkeley Laboratory, as the source, and must include this notice;

(2) Any use of this software must reference this distribution, state that the software copyright is held by the Regents of the University of California, and that the software is used by their permission.

It is acknowledged that the U.S. Government has rights in Scry under Contract DE-AC03-765F00098 between the U.S. Department of Energy and the University of California.

Scry is provided as a professional academic contribution for joint exchange. Thus it is experimental, is provided "as is", with no warranties of any kind whatsoever, no support, promise of updates, or printed documentation. The Regents of the University of California shall have no liability with respect to the infringement of copyrights by Scry,

or any part thereof.

---------- End Copyright notice ----------

It should be noted that the Sun RPC code in the "pcrpc" directory is copyright Sun Microsystems, Inc. and is provided here as a modified version of the Sun code that was distibuted by Sun to the Usenet, comp.sources.unix bulletin board. The Sun code contains the following notice:

"Sun RPC is a product of Sun Microsystems, Inc. and is provided for unrestricted use provided that this legend is included on all tape media and as a part of the software program in whole or part. Users may copy or modify Sun RPC without charge, but are not authorized to license or distribute it to anyone else except as part of a product or program developed by the user.

Sun RPC is provided as is with no warranties of any kind including the warranties of design, merchantibility and fitness for a particular purpose, or arising from a course of dealing, usage or trade practice.

Sun RPC is provided with no support and without any obligation on the part of Sun Microsystems, Inc. to assist in its use, correction, modification or enhancement.

Sun Microsystems, Inc. shall have no liability with respect to the infringement of copyrights, trade secrets or any patents by Sun RPC or any part thereof.

In no event will Sun Microsystems, Inc. be liable for any lost revenue or profits or other special, indirect and consequential damages, even if Sun has been advised of the possibility of such damages.

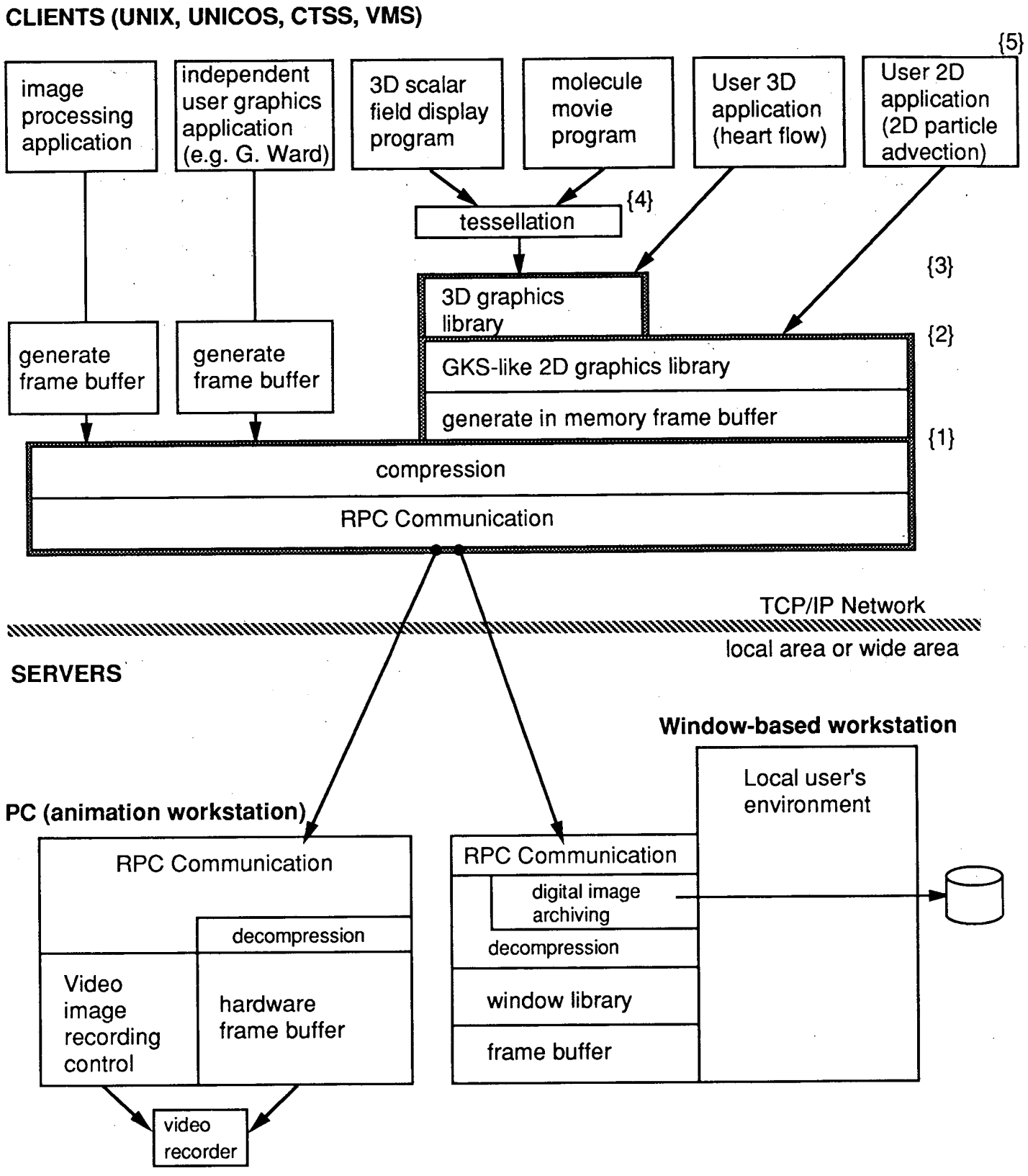
Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, California 94043 "

Additionally, the code in *hershey.c* carries the following copyright notice:

"Copyright (c) 1986, Marc S. Majka - UBC Laboratory for Computational Vision. Permission is hereby granted to copy all or any part of this program for free distribution. The author's name and this copyright notice must be included in any copy."

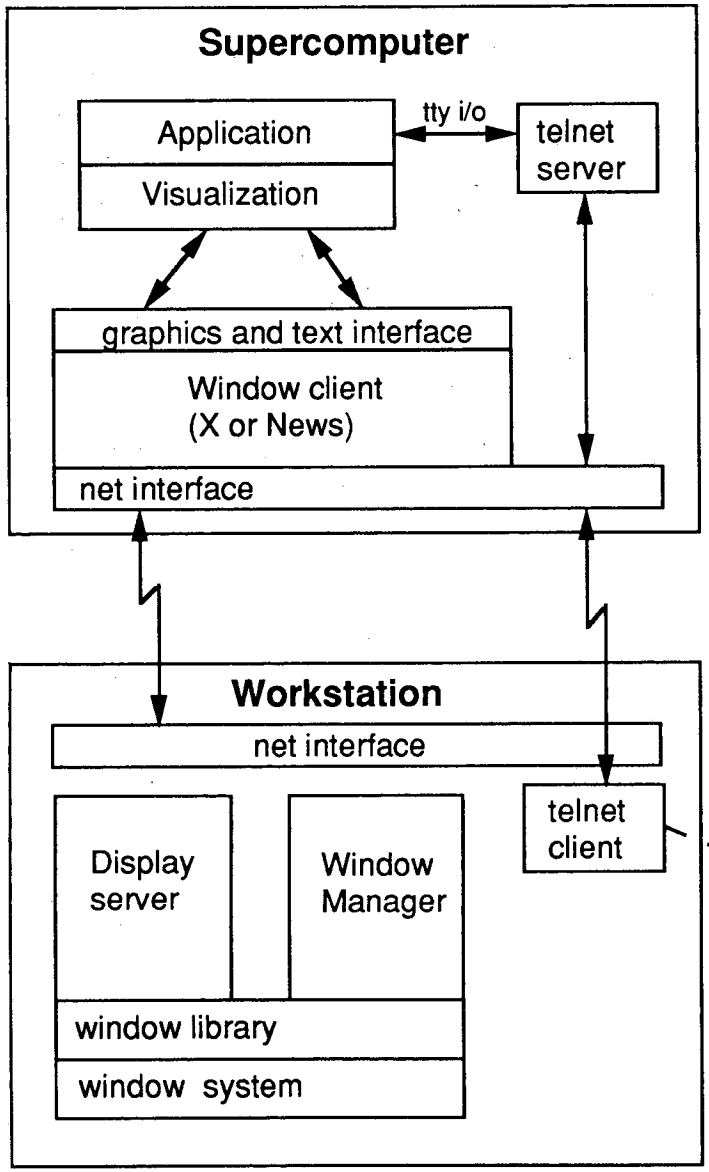
Neither the RPC code nor the Hershey code is any way covered by the University of California copyright notice given above.

LBL Advanced Development Project
"Scry" Distributed Image Handling System Architecture

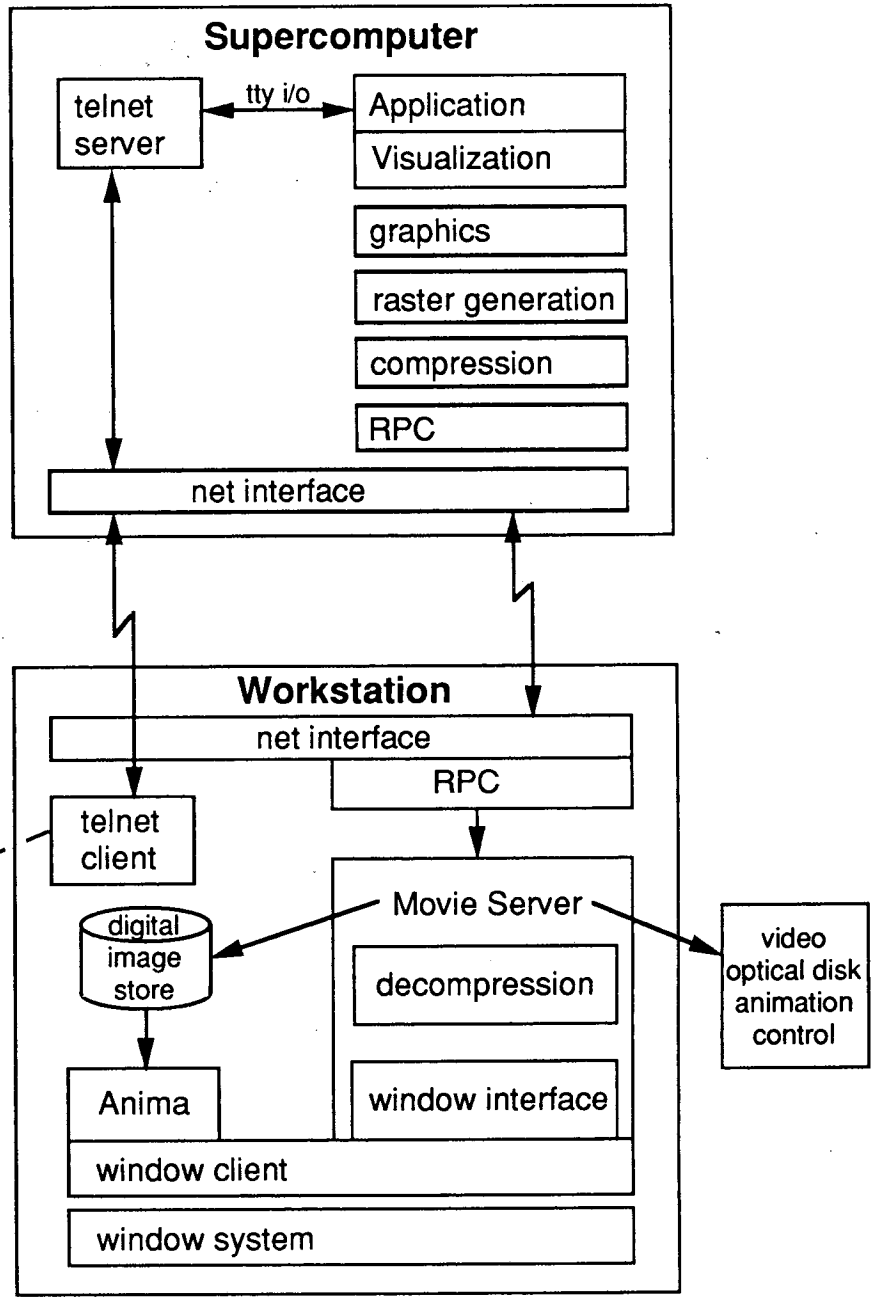


W. Johnston/dlc
11/1/88

Access via Distributed Window System



LBL Distributed Image Handling - Scry



W.Johnston/dlc
11/1/88

Scry 1.1 trouble shooting

April 12, 1989

Following are hints on what action to take, given a certain result or error message. This file is also kept separately in the same public directory on 128.3.254.6 where *scry.tar.Z* is found and will be updated occasionally if necessary.

1. Distributed Aspects

1. The error message *rpctcp_create: connection refused* or *rpcudp_create: Bad file number* appears: the Scry server is not running.
2. The error message *rpctcp_create* or *rpcudp_create: Bad file number* appears: the program number specified on the client with **-wp** is not the same as that specified on the server with **-P**. If using the PC server, the client should always specify the program number **300000**, since it is hard-wired in on the PC side. If using the Sun server, that program number should never be used.
3. The error message *RPC: Procedure unavailablecan't make _____ call* appears. The server does not support that kind of compression.
4. When running Anima, the **prevu** or **record** button is pressed and Anima dies with a segmentation fault: the server was not specified on the command line with **-W**.

2. SEW Problems

1. Opening SEW with **Seopensew(REPORT_ERRORS)** will generate a message on many errors. If **Seopensew** is not called at all, the program will die without explanation.
2. The object appears black. If the light source is **FIXED**, rotate the object. If the light source is **RELATIVE**, rotate the light source.
3. All graphics primitives appear black. Be sure that the color indices for the primitives have been set.
4. If changing the handedness of the coordinate system, for example after using **Serthand(OFF)** to set a left handed coordinate system, be sure to call **Sesetlightsrc** afterwards to ensure the proper setting of the light source.
5. The object is scaled improperly. The defaults will only work if the object ranges from 0 to 1 in world coordinates in all three dimensions. Make sure that **Seset3Dwindow**, **Sesetparallel**, and **Sesetviewvolume** have been called in that order. It will probably be necessary to use the routines (e.g. **Seuvn_obj**) to find the range of the object in view volume coordinates before being able to pass the proper arguments to **Sesetviewvolume**.
6. The object appears off center or goes off the screen. If using the modelling transformation, be aware that any coordinates passed to SEW when **ctm_set(ON)** has been called will be treated as modelling coordinates. This may cause improper setting of

the 3D world window, for example.

7. The object appears off center. Make sure that **Seset3Dwindow** was called before **Sesetparallel**, since **Sesetparallel** sets the view reference point to the center of the world window.

8. The edge of a polygon object appears ragged at the edge of the screen. Set side clipping on with **Sesetclip**.

NAME

Scry - a distributed image handling system

DESCRIPTION

Scry is a general purpose, distributed image handling system. It has been used for distributed scientific movie-making and image processing applications. The system consists of clients that produce images, and servers that display and/or record images. Scry is specifically designed for operation over wide-area networks, as well as local-area networks. It provides various image compression/decompression mechanisms to optimize use of the networks. [It handles a number of common image formats (RLE and HIPS), and provides format conversion tools for several others. Scry also provides an image archiving system for cataloging images and maintaining an off-line image library.] The clients are built on libraries provided with the distributed software. These libraries provide interfaces at three levels. The lowest level provides an interface to compress and transport raster images. The next level provides a GKS-like, 2D graphics interface. The highest level provides 3D graphics primitives with SIGGRAPH Core viewing, and surface rendering extensions.

The client libraries have been written to be fairly portable, and run on a variety of systems including Unix, Unicos, [VMS], and [partly on CTSS]. The client libraries are written in C, and have both C and Fortran interfaces. The clients require a Unix-like run-time library and an implementation of 4BSD Unix sockets to support them.

The client may run very slowly under UNICOS if the Cray used has only 4 megawords of memory. Scry takes up a fair amount of memory because of the Cray word size, e.g. because of data structures such as the software frame buffer.

There are two servers provided with Scry. One of these servers runs on a window-based workstation, and the other on a PC based animation workstation. The window based server uses SunView [or X-11/XT] to provide image display. When invoked, the server opens a window and displays each frame that is sent by the client. In addition to display, the Scry server has an option for writing the compressed images to a disk file. This disk file may be archived, sent to the PC animation workstation, or viewed and manipulated with Anima, a movie preview and editing program. A typical use for this server is to monitor the progress of a client on a remote compute server, while saving the frames locally for subsequent video recording.

The PC server is used primarily to control video recording equipment. As a server the PC runs standalone. It receives images from a client, decompresses them into a local frame buffer, and records them one frame at a time on video tape or a video optical disk. [While Scry is intended to be image size independent, many of the current modules assume 512H × 400V pixels. This will be generalized in a future release.]

SEE ALSO

scry_client(L), **scry_pc**(L), **scry_3D**(3), **scry_gks**(3), **scry_scan3d.a**(3), **scry_scan2d.a**(3), **scry_client.a**(3), **scry_pc_server**(3), **scry_sun_server**(3)

W. E. Johnston, D. E. Hall, J. Huang, M. Rible, and D. Robertson. "Distributed Scientific Video Movie-Making". Proceedings of the Supercomputing Conference 1988 (The Computer Society of the IEEE). Also available as LBL-24996, University of California, Lawrence Berkeley Laboratory, Berkeley, CA (1988).

AUTHORS

David Robertson, Bill Johnston, Nicole Texier, BJ Wishinsky, Fritz Renema, Max Rible, Teck-Joo Chua, and James Huang

NAME

Scry_client - Scry client overview

DESCRIPTION

Scry clients are built using one of several interfaces. These interfaces are implemented by modules that are provided as several different libraries. The modules are described in detail in **scry_3D.3, scry_gks.3, scry_scan3d.a.3, scry_scan2d.a.3,** and **scry_client.a.3.**

Scry clients generate and send images to the Scry servers. The client libraries and programs run on several different systems that provide the 4BSD Unix socket IPC mechanisms, and enough of the Unix run-time library to support the Sun RPC library, RPC's being the preferred communication mechanism. Scry clients run on 4.3BSD Unix systems, [VMS systems with TCP/IP] , and Cray UNICOS. For [Cray CTSS there is a separate client and server library that uses the socket IPC interface directly without the mediation of the Sun RPC library.]

A typical Scry client consists of several distinct layers that are fairly well insulated from each other. The first layer provides graphics primitives that support graphics representation algorithms ("scientific visualization" algorithm in the current parlance). Algorithms of this type convert data structures into a geometric form that can be used to generate graphics primitives. For example, a particle advection representation of a flow field is generated by transporting hypothetical particles using a vector on grid representation of a flow field. The resulting particle positions can be displayed on a frame-by-frame basis to produce an animation of the original (time-dependent) flow field. This position information is passed to the graphics primitives provided by the Scry, GKS-like (see man pages **scry_gks2d.a**(3)), or SIGGRAPH Core-like (see **scry_3D**(3)) interface. Another example is an implementation of the marching cubes algorithm (see **surfmov**(l)) [6].

The next layer of a Scry client converts the graphics primitives into an in-memory representation of the image. From the point of view of the GKS level this is just a device driver that generates an in-memory frame buffer. The two versions of this module are described in **scry_scan3d.a**(3). and **scry_scan2d.a**(3).

The final two tasks of the client are the compression and transmission of the image to the server, and are handled by the routines described in **scry_send_frame.a**(3). Like all of the Scry modules, this layer has been made as data-independent as possible. This module neither knows nor cares where a particular in-memory image has come from; it only needs to know what type of compression to apply and where to send the resulting byte stream. This permits user programs not built on the Scry, GKS interface to use Scry to compress and transmit images to a Scry server for display. Scry is currently used to handle images from both image processing clients, and alternative rendering clients. These clients produce a frame buffer in standard format and pass it to the **send_frame** routines.

Since the **send_frame** module does not know the characteristics of the image that it is dealing with, it is not possible to automatically select an optimal compression technique. Scry provides several compression techniques, some lossy, some lossless, some suitable for synthetic images (those generated by scan converting the graphics primitives generated by most visualization algorithms) and some suitable for natural (remote sensed) images. Generally speaking, the compression techniques are: (1) block truncation coding (BTC) [1]; (2) Heckbert's median cut color map algorithm [2]; (3) frame-to-frame differencing [3]; and (4) Lempel-Ziv coding [4]. A detailed analysis of the characteristics of all of the useful combinations of these is too tedious here (see [3] and [5]), but a few rules of thumb are useful. If the final display is to be a video movie, fairly inexact compression works well. For video movies made from images generated by rendering graphics primitives, we usually use a combination of BTC and the color map algorithm, with Lempel-Ziv cascaded if the images are being sent via a wide-area network. For natural images, color map compression, and/or Lempel-Ziv compression are frequently useful.

The final step is to send the compressed image to the server. While socket-based servers are identified by internet address and port number, RPC-based servers are identified by internet address and RPC program number. In some cases the program number has a default value, and in some cases it must be supplied. In particular, it must be supplied when there is the possibility of running more than one server as can be done on, for instance, a Sun workstation. See the sample file *scry/samples/greyrgb.c*, and **scry_sun_server**(3), for a program that operates at this level. *greyrgb*.c (see **greyrgb**(l)) is an example of how to test the low

level interface for compressing and transporting an already-generated raster image (see **scry_send_frame**(3)).

There are two sample 2D clients in *scry/samples*. One (in both a C and Fortran version) links in all the 2D client modules, and uses the GKS-level routines. Explanation of the GKS client interface is given in the **gks***(3) manual entries. The documentation and samples assume the user has some graphics experience. *test_f_gks*.f, generating the identical image as *test_c_gks*.c, is written in Fortran. See **scry_gks**(3) and **scry_3D**(3) on the use of Fortran with the C package. For more information on the use of the sample 2D clients, see **test_c_gks**(l)). *scry/samples* also includes an example 3D client *woggle*.c (see **woggle**(l)), and its analogue written in Fortran, *wogglef*.f.

The sample client programs have run on Sun 3's and 4's under OS 3.4 and above, VAX Unix systems, and on Cray UNICOS systems. The hardest parts of the client software to port to new systems are probably the Fortran to C interface routines.

AUTHORS

David Robertson, Nicole Texier, James Huang, and Bill Johnston

SEE ALSO

scry_pc(L), **scry_3D**(3), **scry_gks**(3), **scry_scan3d.a**(3), **scry_scan2d.a**(3), **scry_client.a**(3), **scry_send_frame**(3), **scry_pc_server**(3), **scry_sun_server**(3)

[1] 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)

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

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

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

[5] 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-25274 from Lawrence Berkeley Laboratory), 1988.

[6] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm *SIGGRAPH '87 Conference Proceedings 21*, 4 (July 1987), 163-169.

NAME

scry_pc_server – remote procedure call server running on PC server workstation

SYNOPSIS

serv -u|t [-o|v]

DESCRIPTION

Serves incoming remote procedure calls (RPC's) to display and optionally record images sent over the network. The Sun RPC package calls the remote "program" **graphics_dispatch** (located in *serv*.c) based on information in the header of the incoming RPC call. Cases in **graphics_dispatch** correspond to remote procedures. The remote procedure number identifying a particular case is also located in the RPC call header, which is internal to the Sun RPC package. Routines starting with **xdr_** decode incoming information from network byte to PC byte order. The type of compression is set by the client, and communicated to the server as part of the first RPC. For more information on how RPC's are implemented on the PC, see Robertson, D. W., W. E. Johnston, D. E. Hall, and Mendel Rosenblum. Video Movie Making Using Remote Procedure Calls and UNIX IPC, LBL-22767, University of California, Lawrence Berkeley Laboratory, Berkeley, CA (1986).

The routines in *scry/pcserver* and *scry/pcrpc* must be **ftp**'d to the PC, and compiled there. Before the client can run, the server must be invoked on the PC.

OPTIONS

–u selects the UDP protocol

–t selects TCP

One of the above must be selected

–o ready the optical disk for recording

–v ready the VTR for recording

If neither of the above two are invoked then the image is only displayed on the frame buffer (a preview mode).

FILES

Contained in *scry/pcserver*:

Makefile	makes the executable of the server
pcrpc.**h**	definitions and declarations
start.**c**	start up server
serv.**c**	remote "program"
commsupt.**c**	support routines common to metafile and software frame buffer approaches
framsupt.**c**	support routines for information sent using software frame buffer approach
framdisp.**c**	display and optionally record information sent with software frame buffer approach
dtarga.**c**	modified, special-purpose, TARGA routines
btc.**c**	BTC decompression
colormap.**c**	BTC and color map decompression
lzwdec.**c**	communicates, via a file on RAM disk, with 68020 coprocessor which performs Lempel-Ziv decompression. Only used when color map compression and the TCP protocol are selected.
definico.**bat**	copies relevant files onto RAM disk. Called at system startup.
decomp.**c**	performs Lempel-Ziv decompression (compiled on 68020 coprocessor). See README.lzw in this directory on how to compile.
vtr.**c**	videotape driver

opt.**c**	videodisk driver

The modified Sun RPC code is contained in *scry/pcrpc*. The PC-DOS names of the files are given in the left column, the original Sun file names and descriptions are given in the right column.

Makefile	compiles and installs the RPC library.
a.**h**	auth.h - "Authentication interface".
a_unix.**h**	auth_unix.h - "Protocol for UNIX style authentication parameters for RPC."
a_unprot.**c**	authunix_prot.c - "XDR for UNIX style authentication parameters for RPC."
c.**h**	clnt.h - "Client side remote procedure call interface." (Some declarations and definitions also needed by server side.)
nb.**c**	Non-Sun routines. Version of Excelan software used had buggy **ntohl** and **htonl** calls.
non-sun.**h**	Non-Sun include file. Contains hard-wired port number.
p_clnt.**c**	pmap_clnt.c - "Client interface to pmap rpc service" (portmapper unused in this server implementation).
p_clnt.**h**	portmap_clnt.h - "Supplies C routines to get to portmap services" (unused).
p_prot.**h**	pmap_prot.h - "Protocol for the local binder service, or pmap" (unused).
rpc.**h**	"Just includes the many rpc header files necessary to do remote procedure calling" (path names modified).
rpc_msg.**h**	"rpc message definition."
rpc_prot.**c**	"This set of routines implements the rpc message definition, its serializer and some common rpc utility routines."
s.**c**	svc.c - "Server-side remote procedure call interface."
s.**h**	svc.h - "Server-side remote procedure call interface."
s_auth.**c**	svc_auth.c - "Server-side rpc authenticator interface."
s_auth.**h**	svc_auth.h - "Service side of rpc authentication."
s_authun.**c**	svc_auth_unix.c - "Handles UNIX flavor authentication parameters on the service side of rpc."
s_tcp.**c**	svc_tcp.c - "Server side for TCP/IP based RPC."
s_udp.**c**	svc_udp.c - "Server side for UDP/IP based RPC."
time.**h**	Non-Sun routine. Stuff needed for Sun RPC that was not in Microsoft <*time.h*>.
types.**h**	Non-Sun routine. Stuff needed for Sun RPC that was not in Microsoft <*sys\types.h*>.
x.**c**	xdr.c - "Generic XDR routines implementation."
x.**h**	xdr.h - "External Data Representation Serialization Routines."
x_arr.**c**	xdr_array.c - "Generic XDR routines implementation" (for arrays).
x_mem.**c**	xdr_mem.c - "XDR implementation using memory buffers" (used with UDP).
x_rec.**c**	xdr_rec.c - "Implements TCP/IP based XDR streams with a 'record marking' layer above tcp (for rpc's use)."

AUTHORS

David Robertson, Fritz Renema, Max Rible, Nicole Texier, and James Huang

SEE ALSO

scry_pc_config(l), scry_client(l), scry_pc(l), scry_client.a(3)

NAME

scry_sun_server - window-based remote procedure call server running on a Sun 3 or 4 workstation

SYNOPSIS

serv [**–t|u**] [**–P** prog_num] [**–f** filename] [**–w** winx winy] [**–C** ident]

DESCRIPTION

Serves incoming remote procedure calls (RPCs) to display images in a SunView window that are sent from the Scry client (OS 4.0 is necessary to run the server). This graphics window is created separately from the text window in which the server is invoked. The image data can be written to a disk file for later use by the Anima movie preview program. Runs on both a color and a monochrome Sun (care must be taken with the Scry client with the monochrome Sun server -- see **scry_gks_prims**(3)).

The RPC server calls **graphics_dispatch** (located in *serv*.c) based on information in the header of the incoming RPC call. Cases in **graphics_dispatch** correspond to supported client procedures. Routines starting with xdr_ decode incoming network information into the Sun format. The type of compression is set by the client, and communicated to the server as part of the first RPC.

The server will run indefinitely unless explicitly terminated by the user. To do this, type **ctrl-C** in the window in which the server was invoked.

In the case of a Sun eight bit frame buffer with a color lookup table, the managment of the LUT with multiple windows is somewhat tricky. In particular if greater than 128 colors are being used, all of the text windows may go black when the mouse is in a graphics window, and visa versa. The subtleties of LUT managment can be found in the SunView document.

OPTIONS

–t use the TCP protocol

–u use the UDP protocol

–P set the program number to be used. The use of different program numbers allows several servers to be active on a particular workstation simultaneously. Each corresponding client must be invoked with the identical program number (see **scry_gks_control**.3).

–f Set a filename where data received by the server will be written to. At present, this only works with the TCP protocol.

–w set the coordinates of the uppermost left corner of the window where the image is to be displayed.

–C sets monochrome (0), or color (1), which is the default.

DEFAULTS

The defaults to the command line arguments are as follows:

1) The protocol used is **TCP.**
2) The starting point is **(0, 0).**
3) The program number is **4000.**
4) The filename is **testfile.**
5) The type of Sun is **color (1).**

FILES

The Sun server is located in *scry/sunserver*

Makefile
makes the executable of the server

sunserv.h
definitions and declarations

start.c start up server

serv.c decides which server routine to call based on information in the header of the incoming RPC call.

colormap.c
: BTC and colormap decompression routines

framdisp.c
: load the color lookup table and display the images on the Sun screen as they come in

framsupt.c
: contains the xdr routines to convert data to the correct format. also writes the data to a file if specified by the user

lzw_decomp.c
: performs Lempel-Ziv decompression

SEE ALSO

scry_client(l), scry_pc(l), scry_client.a(3), scry_pc_server(l)

Teck-Joo Chua, William Johnston, David Robertson, "Sun Movie Server and An Animation Playback Program", LBL-25991, Lawerence Berkeley Laboratory, Berkeley, CA, 1988.

AUTHORS

Teck-Joo Chua, Max Rible, and David Robertson

NAME

anima - animation playback program on a Sun 3 or 4 workstation.

SYNOPSIS

anima [–W PC workstation id] [–w winx winy]

DESCRIPTION

Reads image data from a file (usually created by the Scry sun server) and allows the user to review the frames of a movie in forward or reverse order at various speeds, advance or reverse frame by frame, skip to a particular frame, or capture a frame on disk. (Anima will only run under Sun OS 4.0 and above.)

Since the frames are usually compressed, and therefore not of constant length, the Scry Sun server writes a table of pointers (to the beginning of each frame) at the end of the movie file. If this table is missing Anima will create it on the fly, but this will substantially slow down the first pass through the movie.

Only files created using the TCP option on the Scry Sun server are accepted.

After Anima is invoked, two windows will appear on the screen, one a control panel, and the other the window in which the image(s) will appear. These behave like ordinary Sun windows, e.g. they can be moved. Pushing the quit button causes the windows to be destroyed and Anima to exit.

To read a file, type its name in after "File:" and then click the left button on "get". (All control features require the use of the left button.) If the caret is not in the file field, click the left button with the cursor to the right of "File:". If a mistake is made, use the Sun "Delete" key to backspace over a character. To save an image, follow the same procedure, pushing the "save" rather than the "get" button.

To save a file on disk in Postscript form, follow the same procedure, but push the "lowpost" or "highpost" button. "lowpost" generates a lower resolution (60 dpi) image when the resulting file is sent to a printer. "highpost" generates an 100 dpi image.

Entering a number in the frame field and then pushing the "go to" button will display the image associated with that number. The current frame number and the total number of frames in the sequence are displayed after the "#:" sign. "< step" will step to the previous frame if one is not at the beginning of the sequence, and "> step" will step to the next frame if not at the end of the sequence. " > " instructs Anima to move forward through a sequence of images, and " < " instructs it to move backward. "full" instructs Anima to display images at full speed (several frames per second on a Sun 4/110). Alternatively, one can click on the slider to control the speed (10 is the same as "full" speed.) To stop at a particular frame and/or to clear all the text fields, click the "stop" button.

If a PC animation server was selected by the -W name option, a sequence of stored images can be sent to the PC to be previewed or recorded. Type in the starting number in the "Frame:" field and the ending frame number in the "End:" field. (<CR> will move the cursor from one field to the next, as will clicking the left button in the appropriate field.) If one is only interested in previewing, then click the "prevu" button and the images will be displayed on the PC workstation. As they are sent, the frame number in the "#:" field will be incremented, but the image will not be updated on the Sun side. The -W option can also be used to send images to another Sun server on the network by specifying that server's name on the command line and using the "prevu" button as above.

To record, type in the starting and ending frame numbers as before. In addition, type in the starting videodisk or videotape frame in the "Video:" field. If the server is equipped with a videodisk, typing in a negative number in the "Video:" field causes the videodisk to seek to the first frame of a block of frames long enough to hold the sequence of images. If recording on videotape, it will also be necessary to type in the number of copies per image in the "Copies" field. Now press the "record" button, and the images will be recorded.

OPTIONS

–W Symbolic name of PC workstation which will record sequence(s) of frames on videotape or videodisk

–w set the coordinates of the uppermost left corner of the window where the image is to be displayed.

DEFAULTS

The defaults to the command line arguments are as follows:

1) The starting point is **(0, 0)**.

FILES

Anima is located in *scry/anima*

Makefile
makes the executable of the server

anima.h definitions and declarations

anima.c starts up the animation playback program

colormap.c
BTC and colormap decompression routines

framdisp.c
load the color lookup table and display the images on the Sun screen as they come in

framsupt.c
contains the xdr routines to convert data to the correct format. Also writes the data to a file if specified by the user.

header.c
creates the header array if it is not present in the file

lzw_decomp.c
performs Lempel-Ziv decompression

procs.c routines to handle panel set-up and buttons

targa_to_ps.c
compressed TARGA to Postscript conversion

pshalftone.pro
public domain Postscript header

airfield.image
test Anima file

SEE ALSO

scry(L),scry_client(L),scry_sun_server(3)
Teck-Joo Chua, William Johnston, David Robertson, "Sun Movie Server and An Animation Playback Program", LBL-25991, Lawerence Berkeley Laboratory, Berkeley, CA, 1988.

AUTHORS

Teck-Joo Chua, Max Rible, and David Robertson

NAME

scry_pc - PC video animation workstation configuration

DESCRIPTION

The components of the PC server workstations are an IBM PC "compatible", equipped with an Ethernet board (Excelan, EXOS 205) and 4BSD socket library, an AT&T TARGA, 16-bit frame buffer (TARGA-16) and associated software, and a recording device. There are two workstations presently in use at LBL. Workstation #2 is equipped with a videotape recorder (Sony, SLO-383) and an animation controller (Dia-Quest, DQ-400). Workstation #1 has a video optical disk recorder (Panasonic, TQ-2026F) and a 68020 coprocessor (Definicon, 750/1), to aid in decompression of images sent over the network.

Substitutions can be made for the hardware components of the PC server workstation with appropriate changes to *scry/pcrpc* and *scry/pcserver*. Substitute routines of similar functionality in *vtr*.c and/or *opt*.c if different recording units are used. The rest of the server is recording-device independent. If a different frame buffer is used, locate all the places where the "TARGA" string occurs in a comment and replace the code immediately following the comment. If the Definicon coprocessor is not used, modify the code that follows after **if (buf.lempel_ziv)** in the routine **display_tcp()** in *framdisp*.c. If the Microsoft 4.0 or 5.0 C compiler is not used, you are on your own.

The potentially most difficult modification to make to the code occurs if the PC server workstation has an Ethernet board other than an Excelan. If the Ethernet interface does not provide some version of sockets, forget it. Sun RPC is built on top of sockets. The include files with path names starting with *\etc\libskt* will have to be replaced; these are Excelan specific.

Two different strategies can be taken to modify the Sun RPC routines to work with different Ethernet boards. Every change made to the publicly distributed Sun RPC library (version 1.1), to make it work with the Excelan socket library and the PC, has been marked by a comment in front of it containing the string "CHANGERPC". These are the places most likely to need modification when using a different Ethernet board and software. Alternatively, the server portion of the original public-domain Sun RPC routines can be used as a starting point. Although we have not tried it yet, we believe that the Sun PC-NFS development environment would work perfectly well as an alternative.

AUTHORS

Bill Johnston, David Robertson, Fritz Renema, and Max Rible

SEE ALSO

scry_client(L), **scry_client.a**(3), **scry_pc_server**(3), **scry_sun_server**(3)

D. Robertson, W. Johnston, D. Hall, and M. Rosenblum. "Video Movie Making Using Remote Procedure Calls and UNIX IPC", LBL-22767, University of California, Lawrence Berkeley Laboratory, Berkeley, CA (1986).

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.

NAME

greyrgb – demonstrates use of low-level client

SYNOPSIS

greyrgb { **–ww** *workstation* **–wp** *server program number* } [*options*]

DESCRIPTION

greyrgb demonstrates the use of the low-level client, using an image from the University of Southern California Image Data Base [1], in *scry/samples/data/usc.2.1.1*, as an example. If **MAPPED_STYLE** is not defined, the image is read in in TARGA format and can be compressed. Otherwise the image can be read in as indices into a color map, demonstrating the other image option besides the TARGA format.

The image is binary, but since it is all char's, it can be used on Sun's, Cray's, and Vax's.

OPTIONS

Options may appear in any order. Later options override previous ones.

–ww *hostname*
: Use *hostname* to indicate the server that will display the image.

–wp *prognum*
: Contact the host at the specified program number. This is unnecessary with hosts with one display screen (like a PC), but is required for those that can specify the program number (like Suns).

–wc *compr*
: Select compression as a combination of Lempel-Ziv compression, BTC compression, colormap compression, or frame-to-frame differencing; you may also select none. This only has an effect if **MAPPED_STYLE** is not defined. *compr* is just a string composed of the first letters of the protocol(s) you wish: *l, b, c, f,* or *n*.

-wt Use TCP network protocol to contact the workstation. This is the default protocol.

-wu Use UDP network protocol to contact the workstation.

–h Get quick help on all these options without wasting CPU on the nroff program.

AUTHORS

David Robertson and Fritz Renema

SEE ALSO

scry_send_frame(3), scry_sun_server(l), scry_pc_server(l), scry_client(l)

Weber, A. Image Data Base. University of Southern California USCIPI Report 1070. Image Processing Institute, University of Southern California, University Park, Los Angeles, CA 90089-0272. 1983.

NAME

test_c_gks – tests the 2D GKS package

SYNOPSIS

test_c_gks { **–ww** *workstation* **–wp** *server program number* } [*options*]

DESCRIPTION

test_c_gks tests the 2D GKS package by generating and transmitting a sample image. This program is also provided in Fortran form (*scry/samples/test_f_gks.f*), in which case modifying the appropriate items in the input file *scry/samples/testf.arg* will have the same effect as the options below.

OPTIONS

Options may appear in any order. Later options override previous ones.

–ww *hostname*
: Use *hostname* to display the image.

–wp *prognum*
: Contact the host at the specified program number. This is unnecessary with hosts with one display screen (like a PC), but is required for those that can specify the program number (like Suns).

–wc *compr*
: Select compression as a combination of Lempel-Ziv compression, BTC compression, colormap compression, or frame-to-frame differencing; you may also select none. The default is a combination of BTC and colormap compression. *compr* is just a string composed of the first letters of the protocol(s) you wish: *l, b, c, f,* or *n*.

-wt Use TCP network protocol to contact the workstation. This is the default protocol.

-wu Use UDP network protocol to contact the workstation.

–h Get quick help on all these options without wasting CPU on the nroff program.

AUTHORS

David Robertson

SEE ALSO

scry_3D_obj(3), scry_gks_xforms(3)

NAME

molecules – generates frame from molecular dynamics movie

SYNOPSIS

molecules { **–ww** *workstation* **–wp** *server program number* } [*options*]

DESCRIPTION

molecules generates a frame from a molecular dynamics movie. The data (contained in *scry/samples/data*) is provided by Larry June of the Chemistry Department of the University of California at Berkeley. **molecules** illustrates the use of multiple polygon objects (see **scry_3D_obj**(3)), the modelling transformation (see **scry_3D_ctm**(3)), depth intensity cueing for lines (see **scry_3D_render**(3)), and the marching cubes library (see **march**(l)). The atomic positions are provided in 3 ASCII files, so no data conversion is necessary for these files. However, *spheredat.asc* should be converted into binary. Invoke the program **field_to_bin** (made by the Makefile in *data*) without any command-line arguments.

OPTIONS

Options may appear in any order. Later options override previous ones.

–r α β γ
: The three rotation values α, β, and γ, specifying the position of the synthetic camera. The package will accept floating point numbers; specifications should be in degrees. The program will default to a rotation of 0, 0, 0. For further information, read the man page for **scry_3D_proj**(3).

–ww *hostname*
: Use *hostname* to indicate the server that will display the image.

–wp *prognum*
: Contact the host at the specified program number. This is unnecessary with hosts with one display screen (like a PC), but is required for those that can specify the program number (like Suns).

–wc *compr*
: Select compression as a combination of Lempel-Ziv compression, BTC compression, colormap compression, or frame-to-frame differencing; you may also select none. The default is a combination of BTC and colormap compression. *compr* is just a string composed of the first letters of the protocol(s) you wish: *l, b, c, f,* or *n*.

-wt Use TCP network protocol to contact the workstation. This is the default protocol.

-wu Use UDP network protocol to contact the workstation.

–gg Use Gouraud shading rather than constant shading. The default is constant shading.

–h Get quick help on all these options without wasting CPU on the nroff program.

AUTHORS

Max Rible and David Robertson

SEE ALSO

scry_3D_obj(3), scry_3D_ctm(3), scry_3D_render(3)

NAME

grid – generates frame from modelling of scattering of helium by hydrogen

SYNOPSIS

grid { **–ww** *workstation* **–wp** *server program number* } [*options*]

DESCRIPTION

grid generates a frame from a movie in which the scattering of helium by hydrogen is modelled. The data (contained in *scry/samples/data*) is provided by Pascal Pernot of MCSD of the Lawrence Berkeley Laboratory. **grid** illustrates the use of the grid style of polygon object (see **scry_3D_obj**(3)), and the use of viewports and the workstation transformation (see **scry_gks_xforms**(3)). The top surface shows the first excited electronic state, and the bottom the ground electronic state. The x, y, and connectivity data is contained in gridfile0.asc. Run the program grid_to_bin without any command-line arguments (the program is made by the Makefile in *data*), since Scry expects binary files. An ASCII file is provided to avoid machine dependency.

OPTIONS

Options may appear in any order. Later options override previous ones.

–ww *hostname*
: Use *hostname* to indicate the server that will display the image.

–wp *prognum*
: Contact the host at the specified program number. This is unnecessary with hosts with one display screen (like a PC), but is required for those that can specify the program number (like Suns).

–wc *compr*
: Select compression as a combination of Lempel-Ziv compression, BTC compression, colormap compression, or frame-to-frame differencing; you may also select none. The default is a combination of BTC and colormap compression. *compr* is just a string composed of the first letters of the protocol(s) you wish: *l, b, c, f,* or *n*.

-wt Use TCP network protocol to contact the workstation. This is the default protocol.

-wu Use UDP network protocol to contact the workstation.

–gg Use Gouraud shading rather than constant shading. The default is constant shading.

–h Get quick help on all these options without wasting CPU on the nroff program.

AUTHORS

David Robertson

SEE ALSO

scry_3D_obj(3), scry_gks_xforms(3)

NAME

woggle – makes movie of rotating object

SYNOPSIS

woggle { **–ww** *workstation* **–wp** *program number* } [*options*]

DESCRIPTION

woggle provides a smoothly changing view of an object described by a Mosaic-style polygon file, and optionally records the sequence of frames generated on videotape or videodisk. Experimenting with command-line options can shed light on 3D routines (see **scry_3D**(3)). The "rainbow" effect can be turned off by defining **ONECOLOR** in *woggle.c*; however, it is useful in trying out all options except Gouraud shading. Regular clipping can be tested by defining **NORMAL_CLIP** in *woggle.c*, and prepass clipping by defining **PREPASS_CLIP** instead (see **scry_3D_proj**(3)).

A sample polygon file, *scry/samples/data/torus.poly.asc*, is provided, generated from data produced at New York University by Charles Peskin and Dave McQueen. Be sure to run the program poly_to_bin (made by the Makefile in *data*), as Scry expects binary files. An ASCII file is provided to avoid machine dependency.

The woggle program is also provided in Fortran form (*scry/samples/wogglef.f*), in which case modifying the appropriate items in the input file *scry/samples/woggle.arg* will have the same effect as the options below.

OPTIONS

Options may appear in any order. Later options override previous ones.

–i *file* Polygon input file. Default is a chambered torus.

–r α β γ

The three rotation values α, β, and γ, specifying the position of the synthetic camera. The package will accept floating point numbers; specifications should be in degrees. The program will default to a rotation of 0, 0, 0. For further information, read the man page for **scry_3D_proj**(3).

–dr $\Delta\alpha$ $\Delta\beta$ $\Delta\gamma$

The amount to rotate the object per frame changed. $\Delta\alpha$, $\Delta\beta$, and $\Delta\gamma$ are added to their respective values given in the **–r** command.

–ww *hostname*

Use *hostname* for display of images and/or recording. The graphics package handles everything having anything to do with recording and type of device.

–wp *prognum*

Contact the host at the specified program number. This is unnecessary with hosts with one display screen (like a PC), but is required for those that can specify the program number (like Suns).

–wc *compr*

Select compression as a combination of Lempel-Ziv compression, BTC compression, colormap compression, or frame-to-frame differencing; you may also select none. The default is a combination of BTC and colormap compression. *compr* is just a string composed of the first letters of the protocol(s) you wish: *l, b, c, f,* or *n*.

-wt Use TCP network protocol to contact the workstation. This is the default protocol.

-wu Use UDP network protocol to contact the workstation.

–mr *n* Begin recording at absolute frame *n* on the recording device. The default is to simply preview; there is no other way to enable recording mode.

–ml *l* Make a movie of length *l* frames. The program defaults to displaying a single frame.

–gg Use Gouraud shading rather than constant shading. The default is constant shading.

–gs α β *type*

Set the light source to the angle specified by α, β. This follows the same system as **–r** and **–dr**. See **scry_3D_render**(3). The light source is relative to the viewpoint if type is **SRCREL**. It is fixed relative to the object if **type** is **SRCFIXED**.

–gc *front back*
Front face cull set if *front* = 1. Back face cull set if *back* = 1.

–gr *flag*
Set right-handed coordinate system if *flag* is 1, otherwise left-handed.

–h Get quick help on all these options without wasting CPU on the nroff program.

AUTHORS

David Robertson and Max Rible

SEE ALSO

scry_3D(3), scry_3D_proj(3), scry_3D_render(3)

NAME

surfmov – tesselate and display a dense array of data

SYNOPSIS

surfmov { **–ww** *workstation* **–i** *datafile* | **–c** *inputfile* } [*options*]

DESCRIPTION

Surfmov is a program designed to produce images or movies of three-dimensional scalar fields. It uses an implementation of the marching cubes algorithm described by Lorensen and Cline [1]. The program will take an array of data and tesselate it using triangles. It will then display the file, leaving an intermediate data file containing the polygon information. Under standard conditions, the intermediate file has a name of the format "trianglesXXXXXX" with the "XXXXXX" being a unique identifier. This file is by default placed in /tmp and removed after use. Another directory may be specified for this file if desired. Further, an explicit filename may be given, in which case the program will *not* remove it; this allows great savings of time when merely rotating an object at the same functional value.

The data file's format is three unsigned long integers *l*, *m*, and *n*, followed by $l \times m \times n$ floating-point numbers. These numbers are made up of *l* arrays, each of *m* arrays *n* floats long. Thus, breaking a box into columns based on the z dimension, the ordering goes (0,0), (0,1), ..., (0,m-1), (1,0), ... (l-1,m-1). Examine the source to *writearr.c* in the *scry/surfmov* directory for an example of the data format. (writearr.c must be compiled with one certain mathematical surface defined, then invoked with "writearr file size", size being the dimension used for *l*, *m*, and *n*. Examples are in the Makefile in *surfmov*.) When this is all loaded in, the program's internal representation of it all goes from 0 to *l-1*, *m-1*, or *n-1* (respectively) in each dimension. Normally, the program takes the longest dimension in any direction parallel to an axis and defines a cube of this side length for the 3-D viewing window; it is easy to tell the program minimum and maximum values and get a result that is scaled in an absolute manner, so any change in size is immediately apparent.

Warning to Cray users: **surfmov** uses a large amount of memory. Grid sizes over 50 cubed are not recommended for machines with lesser amounts of memory, especially on a Cray with only 4 megawords of memory.

OPTIONS

Options may appear in any order. Later options override previous ones.

–i *filename*
: Tesselate the data in *filename*.

–o *outfile*
: Use *outfile* for the intermediate file, rather than the standard /tmp/trianglesXXXXXX file. The intermediate file will *not* be destroyed; this will save time when simply rotating the object without changing the surface value.

–v ϕ
: Evaluate the surface at functional value ϕ. If is not specified, the value 0.0 will be used.

–I *valfile*
: Read surface values (as ASCII text, one value per line) from *valfile*, matching each given ϕ value with a frame.

–r $\alpha\ \beta\ \gamma$
: The three rotation values α, β, and γ, specifying the position of the synthetic camera. The package will accept floating point numbers; specifications should be in degrees. The program will default to a rotation of 0, 0, 0. For further information, read the man page for **scry_3D_proj**(3).

–dr $\Delta\alpha\ \Delta\beta\ \Delta\gamma$

The amount to rotate the object per frame changed. $\Delta\alpha$, $\Delta\beta$, and $\Delta\gamma$ are added to their respective values given in the **–r** command.

–dv δ The amount to change the function value by with each frame. As a default, the value will remain the same ϕ specified by **–v**. This option is not to be used with the **–E** option.

–c *argfile*
Get any or all arguments (as you can specify a great number of them) from the file *argfile*. These arguments are overridden by any contradictory command line arguments.

–ww *hostname*
Use *hostname* for display of images and/or recording. The graphics package handles everything having anything to do with recording and type of device.

–wp *prognum*
Contact the host at the specified program number. This is unnecessary with hosts with one display screen (like a PC), but is required for those that can specify the program number (like Suns).

–wc *compr*
Select compression as a combination of Lempel-Ziv compression, BTC compression, colormap compression, or frame-to-frame differencing; you may also select none. *compr* is just a string composed of the first letters of the protocol(s) you wish: *l, b, c, f,* or *n*.

-wt Use TCP network protocol to contact the workstation.

-wu Use UDP network protocol to contact the workstation.

-W3 *x X y Y z Z*
Set the 3-D window coordinates to be from *(x, y, z)* to *(X, Y, Z)*. This option can be used to set an absolute bound on the scaling. Normally, the package will attempt to display the object at maximum size. However, for movies where the relative size of the object being displayed is important, this option can be used to preserve objectivity.

–mr *n* Begin recording at absolute frame *n* on the recording device. The default is to simply preview; there is no other way to enable recording mode.

–ml *l* Make a movie of length *l* frames. The program defaults to displaying a single frame.

–me Ω The functional value to display at the ending frame. Use of the **–me** option will cause the program to take the starting and ending values and the number of frames in the movie (specified by **–ml**) and work out its own δ value. This option is not to be used with the **–dv** option.

–gg Use Gouraud shading rather than constant shading. This supplies normals to the graphics package and is possibly more dependable. The CPU usage doesn't change much either way. When you've got enough triangles, it becomes academic whether you are using Gouraud or constant shading anyway. This option is mutually exclusive with the one for turning off interpolation.

–gi Turn off interpolation in the tessellation algorithm. This will turn smooth spheres into Mayan beach balls and really not save all that much CPU time. This option is mutually exclusive with Gouraud shading.

–gs α β Set the light source to the angle specified by α, β. This follows the same system as **–r** and **–dr**. See **scry_3D_render**(3). The light source is always relative to the viewpoint, rather than absolute; this prevents the object from rotating 180 degrees and becoming absolutely black.

–t *directory*
Use *directory* for the temporary directory, instead of /tmp. Do not append a slash to the directory name.

–R *filename*
Render the file *filename* assuming that it is a valid LBL BYU data file. The options **–mr, –W2, –W3,** and **–gs** will be the only ones to do anything serious; use of the **–v** option will put up whatever value is given rather than the actual value the data file represents. The **–gg** option must be

given if the file was generated using the **–gg** option.

–h Get quick help on all these options without wasting CPU on the nroff program.

FILES

/tmp/trianglesXXXXXX
Temporary file

SOURCE

Contained in the marching/src directory of the distribution:

Makefile	Makefile for creating *surfmov*.
march.h	contains type declarations, user macros, extern declarations, and program-specific structures and definitions.
march.c	reads the input file, writes out the data, and calls the routines in array.c and tessel.c.
array.c	scans the dataset and creates an array of arrays containing information about all cubes intersecting the given surface.
tessel.c	uses the output from array.c to do the actual tessellations, leaving all data in memory.
movie.c	contains the main routine and calls to set up the graphics package.
data.c	sets up the arrays for the marching cubes algorithm.
render.c	renders the tessellated surfaces.
fixfrag.c	Code fragment included three times in tessel.c to check for and fix the holes that sometimes appear in surfaces due to an error in the conception of the marching cubes algorithm.
normfrag.c	Code fragment included in tessel.c to extract the normals from points selected.
writearr.c	generates sample input for **surfmov** and the **march** library

SEE ALSO

scry_3D(3),march(l)

[1] Lorensen, William E. and Cline, Harvey E. Marching Cubes: A High Resolution 3D Surface Construction Algorithm *SIGGRAPH '87 Conference Proceedings 21*, 4 (July 1987), 163-169.

DIAGNOSTICS

Most of the time, when surfmov detects an error, it calls perror with the name of some file or array that is being worked on. There are a few custom diagnostics, mostly errors specific to bad arguments.

Marching at ____...
The program is reading in the data file and tessellating at the surface value given. No newline should appear.

____ triangles. Rendering...
Control has been transferred to the SCRY graphics package. Again, no newline should appear.

Frame rendered.
The rendered frame has been sent to the specified host.

Did ___ triangles out of ___???
The number of predicted triangles is different from the number of triangles that have been tessellated. Check to make sure that array.c and tessel.c have the same compilation date; if one has been set up with PARANOIA defined and the other hasn't, this error is likely to occur.

AUTHORS

Max Rible and David Robertson (max@csam.lbl.gov, davidr@csam.lbl.gov)

BUGS

In a special, well documented case, the marching cubes algorithm will create quadrilateral holes in a surface that should be completely filled in. This can be fixed by recompiling with the PARANOIA option defined in march.h.

NAME

march – tesselate a dense array of data

SYNOPSIS

long march(dx, dy, dz, surface_value, grid, vertex_list, norm_list, conn_list, numverts, numsides)
unsigned long dx, dy, dz;
float ***grid, surface_value, *vertex_list[3], *norm_list[3];
long **conn_list, *numverts, *numsides;

DESCRIPTION

march is a routine designed to tesselate a dense array of data (specified by **grid**) using an implementation of the marching cubes algorithm described by Lorensen and Cline [1]. (In this case the algorithm is part of a separate library instead of being integrated with the rendering package, as in **surfmov**(l).) The routine will take the array of data as input and tesselate it at **surface_value**, leaving the results in the various arrays and pointers.

grid must be of **dx** by **dy** by **dz** dimensions. When the function returns, ***numsides** and ***numverts** will contain the number of connections and vertices respectively. ***conn_list** will be the address of the beginning of an array of ***numsides** length of long integers containing indices into the vertex list, which describes the connectivity between vertices to produce triangles. It is movie.byu style, with negative indices indicating the last vertex of a polygon. For an example of this type of edge list, see **scry_3D_obj**(3). **vertex_list** should already be an array of three pointers to floats, and it will be filled with the x, y, and z coordinates of each vertex. The 0th element of the arrays is blank. Vertices start at the 1st element because of the movie.byu style of connectivity list. If **norm_list** is supplied, the marching cubes will oblige by spending extra CPU time on finding the edge normals to the data, each entry in **norm_list** being analogous to one in **vertex_list**. As with **vertex_list**, the zeroth element is unused. If you do not wish or need normals, simply substitute NULL for **norm_list**. **march** returns the number of triangles tessellated.

An example call to this routine might be:

```
float sludge[17][23][42];
long *connections, num_connections, num_corners;
float *corner_coords[3], *corner_norms[3];

/* fill in sludge with something */

march(17, 23, 42, 3.141592, sludge, corner_coords, corner_norms, &connections,
&num_corners, &num_connections);
```

Another example of its use is provided in the program **molecules**(l).

Warning to Cray users: this program uses a large amount of memory. Grid sizes over 50 cubed are not recommended for machines with lesser amounts of memory, especially on a Cray with only 4 megawords of memory.

SEE ALSO

surfmov(l)

[1] Lorensen, William E. and Cline, Harvey E. Marching Cubes: A High Resolution 3D Surface Construction Algorithm *SIGGRAPH '87 Conference Proceedings 21*, 4 (July 1987), 163-169.

BUGS

Normals are not returned based on a consistent ordering of the vertices of polygons. In other words, a polygon adjacent to another may have its normal incorrectly facing in the opposite direction. The suggested way around this is taken by the rendering package associated with **surfmov**(l). If a polygon normal faces in the opposite direction of the eye point, the normal's sign should be reversed.

AUTHOR

Max Rible (max@csam.lbl.gov)

NAME

SCRY_3D – 3D viewing

DESCRIPTION

3D viewing includes setting up a view from a certain angle of an object made of points, lines, or polygons, as part of the projection to 2D, and also 3D clipping. The 3D viewing module sits on top of a "GKS" module modified for 3D (see **gks**(3)). GKS routines are called to display projected graphics primitives (points, lines, and polygons). Most of the GKS routines that do not directly display primitives are identical to those used in the 2D version of Scry.

A separate device driver is used for scan conversion of 3D primitives. As in 2D, scan conversion takes place into a software frame buffer located in main memory. 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 assignment of color according to some lighting model (calculating the color of a pixel based on the position of the light source), and shading, which can involve varying the pixel color across the polygon surface to present the appearance of a curved surface, according to some interpolation model.

A large proportion of the module is an extension of an experimental 3D viewing package for displaying line drawings, which provides an easy means of changing the view of an object. This experimental package, SGP the Easy Way (called SEW for short), is a library of subroutines designed to simplify the use of the Simple Graphics Package (SGP) 3D viewing routines described in Foley and van Dam [1]. It allows 3D graphics applications to use the **synthetic camera** viewing provided by SGP, without having to determine and provide complex viewing parameters. An easier, more intuitive method of specifying views is used in SEW. There are also routines to assist in scaling and centering the object(s) being viewed.

The experimental package is coded in Fortran. User-level C routines are provided to make it easier to access Fortran routines from within C. For example, these C routines make sure that each argument is passed by reference to Fortran. Additions to the package were made both in C and Fortran. The majority of the additions allow rendering of 3D polygons. The 3D viewing module can also be accessed from within Fortran. The Fortran routines have different names; corresponding C and Fortran routines are identified in the manual entries listed below.

scry_3D_control(3) describes routines for opening and closing 3D viewing, and setting defaults.

scry_3D_ctm(3) describes routines performing modelling transformations.

scry_3D_obj(3) describes routines handling databases containing objects made up of 3D polygons.

scry_3D_proj(3) describes the SGP, and the alternative SEW manner of projection from 3D to 2D. It also describes the routines performing 3D clipping.

scry_3D_render(3) describes routines that display graphics primitives; set the light source and type of shading in the case of polygons; and set the type of depth intensity cueing in the case of points and polylines.

If it is still unclear how a particular routine works after reading a manual entry, refer to the **woggle**(l) sample program, and experiment with the command-line option that sets the arguments of that routine. This might be useful, for example, to see the interaction of **Serthand** and **Sesetparallel** (see **scry_3D_proj**(3)).

The purpose of Scry is to provide a means for exploratory graphics, where it is often unknown precisely what the visual output will look like. An example would be the output of the marching cubes algorithm [2] (see **surfmov**(l)). Another important goal is to make Scry as easy to use as possible. Thus the 3D rendering is intended to be close to the minimal implementation that will enable the user to see what is going on with his or her modelling. No attempt at state-of-the-art realism is made.

All of the rendering is done in software. On the one hand, this means that it is not close to real time. On the other hand, Scry can run on machines such as Crays and Vaxen without intrinsic support for graphics, and the images, placed in the software frame buffer, shipped to servers (possibly distant) which can display the image and record them on disk, videotape, or videodisk. Speed in rendering is also not as critical where the time spent in modelling dominates, and where Scry is being used to occasionally check on the progress

of the modelling, to make sure that a mistake in the algorithm used in the simulation has not been made.

FILES

Contained in *scry/view3d*:

Makefile	Makefile for creating *libview3d*.a, the 3D viewing library. The installation of this library will have to be modified for your site.
3d.**h**	user declarations and definitions.
3df.**h**	same, but for Fortran user interface.
3dint.**h**	internal declarations and definitions for C portion.
adjnct3d.**c**	sets up polygon clipping and depth intensity cueing
ctm.**c**	modelling transformation (translation only).
ctofor.**c**	interface to Fortran routines if user interface uses C.
errhnd.**c**	handles error messages.
fortoc.**c**	interface to C routines if user interface uses Fortran.
normals.**c**	handles calculation of polygon normals and sets up for Gouraud shading.
polyobj.**c**	routines having to do with handling polygon objects.
scaling.**c**	scaling and centering routines, i.e. 3D world window and the view volume.
seprimt.**f**	handles 3D viewing of primitives.
seuser.**f**	various SEW user-level routines.
sewint.**f**	internal SEW support routines.
sgclip.**f**	Cohen-Sutherland clipping.
sgprojt.**f**	parallel projection.
sgusint.**f**	SGP routines that were formerly user-level.
uvn.**f**	handles setting up scaling of view volume.

SEE ALSO

gks(3), scry_3D_control(3), scry_3D_obj(3), scry_3D_ctm(3), scry_3D_proj(3), scry_3D_render(3), woggle(l)

[1] Foley, J. D., and A. van Dam. Fundamentals of Interactive Computer Graphics. Addison-Wesley Publishing Company, Reading, MA. 1982.

[2] Lorensen, W. E. and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm *SIGGRAPH '87 Conference Proceedings 21*, 4 (July 1987), 163-169.

AUTHORS

David Robertson and BJ Wishinsky

NAME

SCRY_3D_CONTROL – 3D module control

SYNOPSIS

C	Fortran
#include "3d.h"	**INCLUDE '3d.h'**
Seopensew(errind) **int errind;**	**CALL seinit(errind)** **INTEGER errind**
Sesetdefaults()	**CALL sedefs()**
Seclosesew()	**CALL sefin()**

DESCRIPTION

Only the C names are given for the following SEW routines. See the table above for their corresponding names when called from Fortran.

Seopensew prepares the SEW package for use, setting defaults and allocating storage, among other things. If **errind** is **NOREPORT**, no errors in SEW's execution are reported. (Error messages may arise from other sources, i.e. the low-level client.) If it is **REPORT_ERRORS**, error messages are written to standard output. If an error is fatal, a message is printed and the program exits. If the error is non-fatal, a warning message is printed. After 20 warning messages, no more are printed. The default is **NOREPORT**.

Sesetdefaults resets all options to the defaults. The default for each option is listed after the manual entry for the routine that sets it.

Seclosesew closes the SEW package and deallocates storage.

SEE ALSO

scry_3D(3)

NAME

SCRY_3D_CTM – modelling transformations

SYNOPSIS

C	Fortran
#include "3d.h"	**INCLUDE '3d.h'**
ctm_set(flag) **int flag;**	**CALL cset(flag)** **INTEGER flag**
ctm_on_read(flag)	**CALL conread()**
ctm_set_abs(matrix) **double matrix[4][4];**	**CALL csetabs(matrix)** **DOUBLE PRECISION matrix(4,4)**
ctm_trans_abs(x,y,z) **double x,y,z;**	**CALL ctrnabs(x,y,z)** **DOUBLE PRECISION x,y,z**
ctm_trans_rel(x,y,z) **double x,y,z;**	**CALL ctrnrel(x,y,z)** **DOUBLE PRECISION x, y, z**

DESCRIPTION

Only the C names are given for the following Sew routines. See the table above for their corresponding names when called from Fortran.

These routines implement a 3D version of a portion of the CTM package described in Chapter 9 of Foley and van Dam [1], along with some additions. Calling the appropriate CTM routines allows an instance object described in master coordinates to be placed at various locations within the world coordinate system. Multiplication by the CTM (current transformation matrix) transforms master coordinates to world coordinates, which are then passed through the rest of the SEW rendering package. Scaling and rotation of objects in master coordinates are features that will be added in a later version of Scry.

ctm_set instructs SEW to multiply coordinates by the CTM if **flag** is **ON**, and to stop doing so if **flag** is **OFF**. The default is **OFF**. When **ON** is set, any coordinates passed to Sew, and not just those of primitives, for example those passed to **Seset3Dwindow**, **Sesetzrange**, etc., are treated as master coordinates.

ctm_on_read(ON) instructs SEW to multiply coordinates of a polygon object by the CTM before placing them in the arrays describing that object. The default is **OFF**. This routine is included to provide for cases where the CTM only needs to be applied once, instead of for each projection (as in a sequence of images showing rotation of an object). **ctm_set(ON)** should not be used in this situation.

ctm_set_abs sets the CTM matrix to the given **matrix**. The default matrix is the identity matrix. **ctm_set** must have been called with **ON** beforehand for this call to have effect.

ctm_trans_abs sets the CTM matrix to a particular translation matrix using **x**, **y**, and **z**. The default translation is x=0, y=0, z=0. **ctm_set** must have been called with **ON** beforehand for this call to have effect.

ctm_trans_rel premultiplies the CTM matrix by the particular translation matrix set up using **x**, **y**, and **z**. **ctm_set** must have been called with **ON** beforehand for this call to have effect.

SEE ALSO

scry_3D(3)

[1] Foley, J. D., and A. van Dam. Fundamentals of Interactive Computer Graphics. Addison-Wesley Publishing Company, Reading, MA. 1982.

NAME

SCRY_3D_PROJ – 3D to 2D projection

SYNOPSIS

C

```
#include "3d.h"

Sesetparallel(alpha,beta,gamma)
double alpha,beta,gamma;

Sestartuvncal(opt)
int opt;

Seuvncal(x,y,z,n)
double x[],y[],z[];
int n;

Secloseuvncal(center,vxn,vxx,
        vyn,vyx,vzn,vzx)
int center;
double *vxn,*vxx,*vyn,*vyx;
double *vzn,*vzx;

Seset3Dwindow(xmn,xmx,ymn,ymx,
        zmn,zmx)
double xmn,xmx,ymn,ymx;
double zmn,zmx;

Serthand(hand)
int hand;

Sesetclip(type,back,front,side)
int type,back,front,side;

Sesetviewvolume(xmn,xmx,ymn,ymx
        zmn,zmx)
double xmn,xmx,ymn,ymx;
double zmn,zmx
```

Fortran

```
INCLUDE '3d.h'

CALL separal(alpha,beta,gamma)
DOUBLE PRECISION alpha,beta,gamma

CALL seuvnon(opt)
INTEGER opt

CALL seuvncl(x,y,z,n)
DOUBLE PRECISION x(n),y(n),z(n)
INTEGER n

CALL seuvnof(center,vxn,vxx,
        vyn,vyx,vzn,vzx)
INTEGER center
DOUBLE PRECISION vxn,vxx,vyn,vyx
DOUBLE PRECISION vzn,vzx

CALL sewn3(xmn,xmx,ymn,ymx,
        zmn,zmx)
DOUBLE PRECISION xmn,xmx,ymn,ymx
DOUBLE PRECISION zmn,zmx

CALL serset(hand)
INTEGER hand

CALL seclip(type,back,front,side)
INTEGER type,back,front,side

CALL sevwvol(xmn,xmx,ymn,ymx
        zmn,zmx)
DOUBLE PRECISION xmn,xmx,ymn,ymx
DOUBLE PRECISION zmn,zmx
```

DESCRIPTION

As mentioned in **scry_3D**(3), "SEW" stands for "SGP the Easy Way". The main simplification has to do with the manner of setting the projection. (Much of the following is lifted from Wishinsky, BJ. A Simplified Interface for SIGGRAPH Core Viewing. LBL-25038, University of California, Lawrence Berkeley Laboratory, Berkeley, CA. 1987.)

The hardest thing to do in SGP is the preparation of the viewing parameters. In SGP a view is defined by a view reference point and a set of vectors. Calculating the vectors that will generate a given view is sometimes a difficult proposition. In SEW, a projection is defined by **revolving** the synthetic camera around the scene. No vectors need be calculated.

The following is a description of defining a projection in general, and how it is performed in SGP. In order to create a two-dimensional image from three-dimensional data, that data must be projected onto a plane, called the **projection plane** or **view plane**. If you follow an imaginary line from each three-dimensional point through the view plane, the point at which the line intersects the view plane is the projection of that

point. The imaginary line is called a projection ray or projector.

In a parallel projection, the projectors through all the data points are parallel to each other. A parallel projection is specified with a vector called the direction of projection (DOP), which gives the direction of the projectors.

In order to restrict the volume of 3-space that is to be projected, a **view plane window** must also be specified, a rectangular region of a view plane which together with the DOP defines the sides of a volume. The window is specified in the u,v coordinates of the view plane. The addition of front and back "clipping planes" gives rise to a six-sided volume, each side of which may be used for clipping (limiting) the 3-dimensional space to be viewed.

Suppose it is desired to view the data for a three-dimensional object from several different angles, for example, from the front, top and side. There are two ways to think of these different views. One is that the object is moving, viewed from some fixed location. The other is that the object is fixed and you move around it to see the differing views. These two methods are equivalent, that is, exactly the same views can be obtained either way.

SGP uses the second method, which is called **synthetic camera** viewing. The view plane can be imagined as the film in a camera, and the camera can be moved whenever a different view is desired. The view definition routines in SGP allow the user to specify the location of the camera by changing any of three view parameters:

the view reference point (VRP)
the view plane normal (VPN)
the view up vector (VUP).

The view reference point (VRP) may be thought of as the "center of attention". All other view parameters are relative to the VRP, which is generally (but not necessarily) one of the data points being plotted. This is the point about which the view is rotated. The projection of this point will be the origin of the view plane's u,v coordinate system.

The view plane normal (VPN) is a vector perpendicular to the view plane, or the film in the camera. Together, the VPN and VRP define the view plane. The view plane normal can be imagined as a string attached to the view reference point on the film, and coming out of the camera to point at the scene being viewed.

Now imagine holding the string fixed and rotating the camera about the string. As the camera is rotated about the string, the view will rotate in the view plane, changing which part of the scene points up. This angle of rotation is indicated by the view up vector (VUP). The view up vector can be thought of as an arrow from the view reference point in the view up direction. To set this portion of the view, look through the camera and rotate the camera until its "up" corresponds with the view up vector. The view up vector determines the orientation of the view plane's u,v coordinate system.

In SGP four calls are necessary to set up the projection. Calls are made to set the view reference point, set the view up vector, set the view plane normal, and set the DOP.

In SEW, only one call is needed to set the projection. **Sesetparallel** defines a 3D parallel projection based on the movement of a synthetic camera about the object(s) being plotted. (Only the C names are given for this and following SEW routines. See the table above for their corresponding names when called from Fortran.) It is important to remember that the input parameters define the movement of the CAMERA, not the movement of the scene being viewed. The view reference point (VRP) is set to the center of the scene in world coordinates, determined from the results of **Seset3Dwindow**. The direction of projection (DOP) is set to equal the VPN, providing an orthographic projection. The setting of the VPN is determined according to a (latitude,longitude) system. One end of the VPN is anchored on the VRP, and the other is set as if the camere were placed on a globe surrounding the scene. **alpha** represents the latitude, and **beta** represents the longitude. Zero degrees longitude is at the front of the scene in world coordinates. It should be noted that less predictable results will be obtained if **alpha** is greater than 90 degrees or less than -90

degrees. **gamma** determines the tilt of the camera (of the view plane). The VUP is set as a unit vector (0.0,1.0,0.0) that has been rotated according to **alpha** and **beta**, and then by **gamma**.

The angles **alpha**, **beta**, and **gamma** are relative to the default view, which is an orthographic parallel projection from the front. Multiple calls to **Sesetparallel** are not cumulative; for every call the view is recalculated relative to the default. The default **alpha**, **beta**, and **gamma** is **(0,0,0)**.

The SGP routines are no longer available because of the difficulty in specifying a view with vectors, especially a sequence of views resulting in a smooth rotation. For example, what orientation of the camera is set by a view up vector of (0.0,1.0,1.0)?

In the original SEW implementation, the perspective projection was available as well. In this modified version, only the parallel projection is available. SEW is intended to be used for exploratory graphics in which the positions of objects in a scene may not be known. In such a situation it is more likely that undesired results will occur with the perspective projection, e.g. if the center of projection is set wrong, the result will be garbage.

Another operation that is sometimes awkward in SGP is the definition of the view plane window. Often it is not certain in what range of u,v coordinates the projected image will appear, and part of the scene may be inadvertantly clipped. The following three routines go through the calculations of projection and return the coordinates of a view plane window that will display the complete scene, with the option of being centered, along with the range of z values in view volume coordinates.

The following trio of routines are used to find the minimum and maximum x, y, and z coordinates in the view volume coordinate system (see above), given set(s) of graphics primitives -- points, lines, and polygon objects.

Sestartuvncal indicates that one or more sets of coordinate data are ready to be accepted. If **opt** is set to **SINGLE_FRAME**, the minima and maxima are found for a particular setting of the synthetic camera. In this case, **Sestartuvncal** must be called after the projection has been set up with **Sesetparallel**. If **opt** is set to **ANIMATION**, the minima and maxima are found for all settings of the synthetic camera's orientation relative to the scene in world coordinates. In this case, **Sestartuvncal** should be called before setting the projection with **Sesetparallel**.

Each set of coordinate data is then passed to **Seuvncal**. **n** indicates the number of data points in the set, and can be as few as one. The minima and maxima resulting from a particular polygon object, obtained with **Sevpw_obj**, can be compared with these results to find the overall minima and maxima.

A call to Secloseuvncal indicates that all data has been submitted, and the minimum and maximum x, y, and z view volume coordinates returned. If **center** is set to 1, the x and y minima and maxima are returned, centered. The default minima and maxima bound the unit cube.

Seset3Dwindow informs SEW of the minimum and maximum x, y, and z world coordinates. This call should be performed before Sesetparallel, which sets the view reference point to the center of the world window, and before Sesetviewvolume if using object-oriented clipping. The default minima and maxima bound a unit cube in world coordinates.

Serthand sets world coordinates to be right- (**Serthand(ON)**) or left-handed (**Serthand(OFF)**). 3D coordinates that have been projected to the view plane are considered to be left-handed, that is, the positive z-axis points away from the synthetic camera (i.e., into the display). It is usual to consider that world coordinates are right-handed and therefore the z-coordinates would be negated for projection. The default system is right-handed. It is necessary to set up the handedness of the coordinate system before calling **Sesetlightsrc** (see **scry_3D_render**(3)) since that call depends on the handedness of the system. Toggling **Serthand** also has the effect of changing the (latitude,longitude,tilt) system described above so that positive changes in angle are turned to negative, and vice versa.

Sesetclip sets side, front, and back clipping on or off, i.e. **Sesetclip(type,OFF,ON,OFF)** sets only front clipping on. **type** specifies view-oriented or object-oriented front clipping. View-oriented clipping, where **type** is set to **NORMAL**, refers to clipping against the sides of the view volume. The default is the same as specifying **Sesetclip(NORMAL,OFF,OFF,OFF)**. If a vertex of a primitive goes outside the bounds of

the software frame buffer, and side clipping is not set, that primitive will be thrown away. This can result in a jagged appearance at the edge of the screen, especially with polygons.

In object-oriented clipping, front clipping is performed against a plane parallel to the front of the scene in world coordinates. In other words, the same portions of objects are always clipped. A prepass (with **type** set to **DOFIXED**) is made through all polygon objects (**Sedraw_obj** is called for each object). Ordinary viewing clipping is used at (0,0,0) rotation. The prepass identifies all polygons that are partially or fully clipped, using the Cohen-Sutherland clipping algorithm on their edges. It is not necessary for point and line primitives. After the prepass, type is set to **USEFIXED**, e.g. **Sesetclip(USEFIXED,OFF,ON,OFF)**. **Sedraw_obj** is called for each object again. During scan conversion, each pixel of polygons identified as partially clipped is checked for its relation to the clipping plane. If it is in front, it does not overwrite the pixel location in the frame or z buffer.

Sesetviewvolume has the purposes of setting the side, front, and back of the view volume, i.e. the clipping planes, and scaling the scene so that it just fills the screen if the values returned by **Secloseuvncal** are used. To clip part of the scene use a smaller range of coordinates than those returned by **Secloseuvncal**, and use **Sesetclip**. To make the scene smaller, use **Gsetviewport** (see **scry_gks_xforms**(3)). The default view volume is the unit cube.

SEE ALSO

scry_3D(3), scry_gks_xforms(3)

BUGS

1. It is assumed that a polygon will only intersect one clipping plane. In exceptional cases, as where a polygon intersects both a side plane and the front clipping plane, only one clipping test is performed during scan conversion. In the aforementioned case, side clipping but not front clipping is performed. The result will be a jagged edge.
2. The view volume that will result in an exact filling of the screen is not always calculated correctly by **Seuvncal**, but in the cases tested it has always been close.

NAME

SCRY_3D_OBJ – 3D polygon database handling

SYNOPSIS

C	Fortran
#include "3d.h"	**INCLUDE '3d.h'**
Seread_obj(filename,objnum) **char *filename;** **int objnum;**	**CALL seread(filename,objnum)** **CHARACTER filename*MAXCHAR** **INTEGER objnum**
Sedraw_obj(objnum,func) **int objnum;** **int *func();**	**CALL sedraw(objnum)** **INTEGER objnum**
Sedestroy_obj(objnum) **int objnum;**	**CALL sedstry(objnum)** **INTEGER objnum**
Seinqpolys_obj(objnum,numpolys) **int objnum,*numpolys;**	**CALL seinqpo(objnum,numpolys)** **INTEGER objnum,numpolys**
Sewind_obj(objnum,xmn,xmx, **ymn,ymx,zmn,zmx)** **int objnum;** **double *xmn,*xmx,*ymn,*ymx;** **double *zmn,*zmx;**	**CALL sew3obj(objnum,xmn,xmx,** **ymn,ymx,zmn,zmx)** **INTEGER objnum** **DOUBLE PRECISION xmn,xmx,ymn,ymx** **DOUBLE PRECISION zmn,zmx**
Seuvn_obj(objnum,opt,center, **xmn,xmx,ymn,ymx,zmn,zmx)** **int objnum,opt,center;** **double *xmn,*xmx,*ymn,*ymx;** **double *zmn,*zmx;**	**CALL sevpwob(objnum,opt,center,** **xmn,xmx,ymn,ymx,zmn,zmx)** **INTEGER objnum,opt,center** **DOUBLE PRECISION xmn,xmx,ymn,ymx** **DOUBLE PRECISION zmn,zmx**
Seread_grid(fp,objnum,start,time,scale) **FILE *fp;** **int objnum,start;** **double *time,scale;**	**not available**

DESCRIPTION

These routines handle rendering of 3D polygons, unless **Sepolygon3D** is used. The latter option is not recommended (see **scry_3D_render**(3)). Polygons are assumed to belong to some coherent object, e.g. a sphere. Only the object, identified by an object number, is accessible to the user from within Scry -- edge and vertex data are hidden. This brings simplicity at the cost of some flexibility in handling polygon rendering. For example, all storage allocation, and calculation involved with such as processes as object-oriented clipping and Gouraud shading are handled internally.

Polygon vertex and edge data are assumed to arise from some modelling process. They must be stored, one object per file, in a specific format. As used at LBL, the files are usually generated by a separate process. Some problems have been left to the user to handle, i.e. providing a vertex list with no duplicate vertices, and an edge list where the vertices in each polygons are provided in a consistent clockwise or counter-clockwise order.

The specific polygon input format expected is similar to the input format expected by the Mosaic package of movie.byu. To improve efficiency in reading large data files, Scry polygon object files are read and written as binary. A binary version of the following file would be rendered as two filled triangles.

```
4 6 10 0
 0.80000e+02 0.69282e+02 0.40000e+02 0.00000e+00
 0.00000e+00 0.40000e+02 0.69282e+02 0.00000e+00
 0.13000e+03 0.14000e+03 0.13000e+03 0.14000e+03
   4   1  -2   4   2  -3
```

The first two items on the first line contain the number of vertices in the vertex list and the number of vertices in the edge list. The third item specifies the transparency and should be set to 10. It is provided for compatibility with a future release of Scry, in which the option of using transparency will be added. The fourth item informs SEW whether the vertex normals are being supplied by the file. (For example, vertex normals are supplied by the marching cubes routines if Gouraud shading had been specified -- see **march**(l).) It should be set to 0 if not, 1 if so. All items on the first line should be written as **int**'s.

The vertex list is contained in the next two lines, i.e. x1, x2, x3, x4, y1, y2, y3, y4, z1, z2, z3, z4. These should be written as **float**'s.

If the vertex normals were supplied, they would be given at this point, i.e. normx1, normx2, etc., normy1, normy2, etc., normz1, normz2, etc. These should be written as **float**'s.

The fourth line is the edge list, consisting of indices into the vertex list. These should be written as **int**'s. A negative index signals the end of a polygon. Vertices are numbered starting with 1 rather than 0. The 3D scan conversion module connects the last vertex to the first vertex – don't do that in this file. It is important that the vertices of all polygons be listed in a consistent counter-clockwise order. Otherwise the polygon normals, if calculated by Scry, may have the opposite of their correct sign. This can result in an object being covered with a mesh of light and dark patches.

Only the C names are given for the following routines handling polygon objects. See the table above for their corresponding names when called from Fortran.

Seread_obj reads in an object from a file. **filename** should be the character name of the file, not a file pointer (this choice was made to enable calling this routine from a Fortran program). **Seread_obj** opens the file, read-only, and when done, closes it. **objnum** is the identifier for the object, and should be used to refer to it in other routines operating on objects.

Sedraw_obj renders all the polygons in the object. **func** can either be a pointer to a function, or **NOFUNC**, indicating no function to be passed. (Note that this option is not available from Fortran.) It is called as

func(x,y,z,n)

x, **y**, and **z**, the centroid of the polygon, and **n**, the number of vertices, are set internally in **Sedraw_obj** for each polygon. **func** can then rearrange attributes to provide some desired effect, for example, coloring polygons according to their depth in **z_finder** in *samples/woggle.c*. Note that communication of such things as the range of z coordinates is done by means of external variables. The object model is preserved because no internal data can be acted upon directly. The default for **func** is NOFUNC.

Sedestroy_obj deallocates the space for the object, and enables another object to have the particular object number **objnum**.

Seinqpolys_obj returns the number of polygons in object **objnum**.

Sewind_obj is used to find the minimum and maximum x, y, and z world coordinates, if unknown, of the object **objnum**. The minima and maxima may then be used to set the 3D world window, with or without modification.

Seuvn_obj is used to find the minimum and maximum x, y, and z coordinates in the view volume coordinate system, given a particular polygon object. (See **scry_3D_proj**(3) for an explanation of the view volume.) If **center** is set to 1, the x and y minima and maxima are returned, centered. If **opt** is set to **SINGLE_FRAME**, the minima and maxima are found for a particular setting of the synthetic camera. In this case, **Seuvn_obj** must be called after the projection has been set up with **Sesetparallel**. If **opt** is set to

ANIMATION, the minima and maxima are found for all settings of the synthetic camera's orientation relative to the scene in world coordinates. In this case, **Seuvn_obj** should be called before setting the projection with **Sesetparallel**. The output of **Seuvn_obj** can be compared to that of **Seuvncal** if there is more than one object or there are point and line primitives.

Another form of object can be used when the polygons are arranged in a grid and only the height or elevation data changes with time. In this case there are two separate object files: one for the x, y, and connectivity data, and one for the elevation data. The first file is exactly like the regular object file, except that the z vertex data is left out, and there is no provision for storing vertex normals. The second file is in the form

time0
z1, z2, z3, etc.
time1
z1, z2, z3, etc.
etc.

where time is a double.

Seread_grid reads the "grid object" data in two steps. with the result being a polygon object that can be operated on by **Sedraw_obj**, etc, exactly like a regular polygon object. **start** indicates to SEW whether the file pointed to by **fp** is a x, y, and connectivity file (**start** should be 0), or a z data file (**start** should be 1). **scale** scales the z data by that amount (before the modelling transformation if **ctm_on_read** has been called). **objnum** is an identifier for the object as before. The time step is returned by **Seread_grid** in **time**. See **grid**(l) for a description of the sample program using this option.

SEE ALSO

scry_3D(3), scry_3D_proj(3), scry_3D_render(3)

BUGS

1. Doesn't handle concave polygons. If one is encountered, there will either be an erroneous result or Scry will die horribly.
2. The view volume that will result in an exact filling of the screen is not always calculated correctly by **Seuvn_obj**, but in the cases tested it has always been close.

NAME

SCRY_3D_RENDER – 3D rendering extensions to SEW

SYNOPSIS

C	Fortran
#include "3d.h"	**INCLUDE '3d.h'**
Sepolygon3D(x,y,z,num) **double x[],y[],z[];** **int num;**	**CALL sepgon3(x,y,z,num)** **DOUBLE PRECISION x(foo),y(foo),z(foo)** **INTEGER num**
Sesetcull(front,back) **int front,back;**	**CALL sescull(front,back)** **INTEGER front,back**
Sesetlightsrc(alpha,beta,type) **double alpha,beta;** **int type;**	**CALL seltsrc(alpha,beta,type)** **DOUBLE PRECISION alpha,beta** **INTEGER type**
Sesetshading(type) **int type;**	**CALL seshade(type)** **INTEGER type**
Sepoint3D(x,y,z) **double x,y,z;**	**CALL sept3(x,y,z)** **DOUBLE PRECISION x,y,z**
Seline3D(x,y,z,num) **double x[],y[],z[];** **int num;**	**CALL seplt3(x,y,z,num)** **DOUBLE PRECISION x(foo),y(foo),z(foo)** **INTEGER num**
Sesetdepth(type) **int type;**	**CALL sedepth(type)** **INTEGER type**
Sesetzrange(zmin,zmax) **double zmin,zmax;**	**CALL sezrnge(zmin,zmax)** **DOUBLE PRECISION zmin,zmax**

DESCRIPTION

Only the C names are given for the following Sew routines. See the table above for their corresponding names when called from Fortran.

Sepolygon3D renders a convex 3D polygon with **n** vertices. The 3D scan conversion module connects the last to the first vertex. Use of this routine is an alternative to using the routines handling polygon databases, described in **scry_3D_obj**(3). It is provided to be used in situations where greater access to vertex and edge data is desired, or where the polygon object file format is undesirable, i.e. where there are large numbers of objects with only a few polygons each. There are drawbacks to using this approach to rendering polygons: Gouraud shading and object-oriented clipping are not available. These two options depend on having all the vertex and edge data for an object available, in which there are no duplicate vertices, before any polygon rendering is done. This situation occurs with the polygon object approach, but would be problematic in the case where the data for only one polygon is provided at a time.

Sesetcull sets a front or back-face cull for polygons, i.e. to set a back-face cull, use **Sesetcull(OFF,ON)**. A cull eliminates all polygons where the angle between the polygon normal and the view plane normal is negative (back face) or positive (front face). In other words, a back face cull eliminates all polygons that are normally hidden from the eye point anyway, and a front face cull exposes all hidden polygons. Usually the former option is much more useful. Culling occurs before any scan conversion or rendering is done, and can save a great deal of time in displaying an object. A back-face cull should not be used in

conjunction with clipping, or with objects with holes in them, because in these cases polygons whose normals face in the opposite direction from the view plane normal are exposed. The default is the same as specifying **Sesetcull(OFF,OFF)**. See Rogers, Chapter 4 [1] for more information on culls.

Sesetlightsrc sets the location of a white point light source. **alpha** and **beta** specify the position of the light source in a (latitude,longitude) system in the same manner as in the specification of the eye point (see **scry_3D_proj**(3)). If **type** is **SRCFIXED**, the light source stays in the same relation to the object no matter what the specification of the eye point. That is, if the light source is at (0,0), and the eye point is rotated by 180 degrees, only the side hidden from the light source will be visible, and the whole object will be colored black. If **type** is **SRCREL**, the relationship of the light source to the eye point is held constant as the eye point is moved around the object. Thus if the light source is specified directly in front of the object at the (0,0) position of the eye point, the object as it appears on the display will still be fully illuminated at the (180,0) position of the eye point. The default is the same as specifying **Sesetlightsrc(0.0,0.0,SRCREL)**. See Rogers, Chapter 5 [1] for more information on lighting models.

Sesetshading sets the type of polygon shading. Constant shading (**Sesetshading(CONSTANT)**) results in an object having a faceted appearance, unless the size of the polygons in an object are small compared to the resolution in pixels of the display. All the pixels in a scan-converted polygon are colored the same, depending on the relation of the polyon normal to the light source. Gouraud shading (**Sesetshading(GOURAUD)**) interpolates the color along and between each scan line as a polygon is scan converted, resulting in a smooth appearance of the object. It is more time consuming than constant shading, but can result in a more realistic appearance. However, if there are sharp edges within the object, this implementation of Gouraud shading will eliminate them. The default is constant shading. See Rogers, Chapter 5 [1] for more information on shading models.

Sepoint3D scan converts a 3D point. Unless the defaults are used, the width should have previously been set with **Gsetptwidth** and the color set with **Gsetptcolourind** (see **scry_gks_prims**(3)). Depth intensity cueing is used depending on the **Sesetdepth** call (see below).

Seline3D scan converts a 3D line using a version of Bresenham's algorithm. Unless the defaults are used, the width should have been set with **Gsetlinewidth** and the color set with Gsetlinecolourind (see **scry_gks_prims**(3)). Depth intensity cueing is used depending on the **Sesetdepth** call (see below).

Sesetdepth sets the type of depth intensity cueing, i.e. **NONE**, **DEPTHDIST**, or **WORLDDIST**. **NONE** chooses no depth cueing. **DEPTHDIST** colors a point or a point on a line "darker" or "lighter" depending on whether its z value after viewing has been performed is further from or nearer to the front of the scene (see Foley and van Dam [2] and **scry_3D_proj**(3)). **WORLDDIST** colors a point or a point on a line "darker" or "lighter" depending on its z value in world coordinates. This obviously assumes that what is desired to be the front of the object in world coordinates is correctly oriented, and may require a user-performed rotation of points or lines before handing them off to Scry. (A rotation of primitives in master coordinates may be provided in a future release of Scry -- see **scry_3D_ctm**(3)). The default is **NODIST**.

Sesetzrange sets the z range used in depth intensity cueing. If **Sesetdepth(WORLDDIST)** has been specified, the color of a point or a point on a line depends on its relative z position to the minimum z world coordinate value (**zmin**), divided by the total z range (**zmax** - **zmin**). If **DEPTHDIST** was chosen, the situation is similar, except that the color is based on the minimum and maximum z values as they have been transformed by the viewing operation. The default zmin is 0.0 and the default zmax is 1.0.

SEE ALSO

scry_3D(3)

[1] Rogers, D. F. Procedural Elements for Computer Graphics. McGraw-Hill Book Company, New York. 1985.

[2] Foley, J. D., and A. van Dam. Fundamentals of Interactive Computer Graphics. Addison-Wesley Publishing Company, Reading, MA. 1982.

NAME

scry_gks – device-independent graphics calls libraries

SYNOPSIS

#include "gks.h"

DESCRIPTION

GKS (the Graphical Kernel Standard) provides a device-independent manner of writing graphics applications programs. It is described in Enderle, G., K. Kansy, and G. Pfaff. Computer Graphics Programming: GKS - The Graphics Standard. Springer-Verlag, Berlin. 1984.

The routines in the gks2d.a and gks3d.a libraries are best described as "GKS-like", rather than as even a minimal GKS implementation. GKS does not support 3D; 3D graphics primitives have an additional non-standard z argument for use by the z buffer. The HSV rather than the RGB color model is used. (Algorithms for converting from one color representation to another are provided in get_hsv.c and get_rgb.c) There is really only one device driver (in 2 versions), for scan conversion of 2D or 3D graphics primitives into a software frame buffer. Several routines included in this library are not standard GKS.

The routines are written in C and have the standard C names (starting with "G") used in several commercial GKS packages (there is currently no standard C binding). However, the arguments to these routines are in the Fortran binding form. For those wishing to use this package from within Fortran, interface routines with Fortran binding names (starting with "g") are provided that take care of the details of calling C routines from Fortran. The file *gksf*.h has been provided for Fortran routines. It can either be included, or copied into the user's program at the appropriate place.

As mentioned in **scry_client(L)**, this initial release runs on Sun Unix, VAX Unix, and Cray UNICOS systems when user routines are called from Fortran. The details of calling C routines from within Fortran are usually system specific. A later release will allow the use of this package from Fortran on CTSS systems.

Routines common to 2D and 3D GKS include those controlling GKS and the workstation, controlling the use of color, converting from world to normalized device coordinates, and setting the workstation transformation. Strictly 2D graphics primitives (points, lines, and text) are available for the user in the 2D version. Strictly 3D calls are not user level. They are only made from within the 3D viewing package (see **scry_3D**(3)). Besides the 3D graphics primitives (points, lines, and polygons) information relating to polygon shading, clipping, and the use of depth intensity cueing is routed to the software frame buffer device driver for 3D.

FILES

Contained in *scry/gks*:

Makefile	Makefile for creating *libgks2d*.a, the 2D GKS library, and *libgks3d*.a, the 3D GKS library. The installation of these libraries will have to be modified for your site.
gks.**h**	user definitions for GKS
gksint.**h**	internal declarations and definitions
gksf.**h**	include file for use from Fortran
gks.**c**	generic GKS routines
gks2d.**c**	2D graphics routines
gks3d.**c**	3D graphics routines
fortoc2d.**c**	Interface allowing 2D Fortran routines to call C routines.
fortoc3d.**c**	Interface allowing 3D Fortran routines to call C routines.
get_hsv.**c**	algorithm to convert from RGB to HSV
get_rgb.**c**	algorithm to convert from HSV to RGB

AUTHOR

David Robertson

SEE ALSO

scry_3D(3), scry_gks_control(3), scry_gks_xforms(3), scry_gks_prims(3), scry(l), scry_client(l), scry_client.a(3)

BUGS

Virtually no error checking is done. The most common error encountered is attempting to display primitives outside the bounds of the frame buffer through improper setting of windows and viewports.

NAME

Gopengks (gopks), Gopenws (gopwk), Gclosews (gclwk), Gclosegks (gclks), Gclearws (gclrwk), Gsetrecmode (grectyp), Gsetcopynum (gcpynum), Gsetcompr (gcompr), Gupdatews (gupwk) – GKS and workstation control routines (alternative names are for use from Fortran)

SYNOPSIS

C	Fortran
#include "gks.h"	**INCLUDE 'gksf.h'**
Gopengks()	**CALL gopks()**
Gopenws(workid,host,protocol,prognum) **int workid;** **char *host;** **int protocol;** **int prognum;**	**CALL gopwk(workid,host,protocol,prognum)** **INTEGER workid** **CHARACTER host*MAXCHAR** **INTEGER protocol** **INTEGER prognum**
Gclosews(workid) **int workid;**	**CALL gclwk(workid)** **INTEGER workid**
Gclosegks()	**CALL gclks()**
Gsetrecmode(record,length) **int record;** **int length;**	**CALL grectyp(record,length)** **INTEGER record** **INTEGER length**
Gsetcopynum(copynum) **int copynum;**	**CALL gcpynum(copynum)** **INTEGER copynum**
Gsetcompr(compr) **int compr;**	**CALL gcompr(compr)** **INTEGER compr**
Gupdatews(workid,regfl) **int workid;** **int regfl;**	**CALL guwk(workid,regfl)** **INTEGER workid** **INTEGER regfl**

DESCRIPTION

Only the C names are given for the following GKS routines; routines to be called from Fortran are interface routines that take care of the details of calling C routines from Fortran and then call the corresponding C-binding routines.

Gopengks prepares the GKS package for use.

Gopenws opens the connection to the Scry server. **workid** must be **FRAMEB**, corresponding to the only driver available: a software frame buffer. **host** is the symbolic name of the server system, or its Internet address. (If the symbolic name is used, then an entry needs to be set in the system */etc/hosts* file to map this symbolic name into the Internet address of the server system.) **protocol** can either be UDP or TCP. Data is sent more slowly using the TCP protocol, but is guaranteed to reach its destination in an error-free fashion, unlike data sent using the UDP protocol. **prognum** is an identifier for the particular instance of the server executing. In the case of the Scry Sun server there can be several servers representing several different windows on a single Sun. **prognum** should always be **PCPROGRAM** for use with the PC server, which can run only one process at a time.

Gclosews closes the connection to the Scry server.

Gclosegks closes GKS (at present this routine does nothing).

Gclearws clears the software frame buffer to the background color.

Gsetrecmode (a non-standard routine) sets the record mode if the PC server is being used. If **record** is **PREVIEW**, images are displayed by the Scry server, but no recording is done. If **record** is a positive integer, the server attempts to start recording at the frame identified by that number. Be sure the recording media does not already have something recorded on the frames starting with **record**. On the VTR, previous material will be recorded over. On the videodisk, which is WORM, an error will be generated. If **record** is **AUTOSEEK**, the second argument indicates the number of images in the movie. This only has an effect if the videodisk is used, and is used to automatically seek to the first place on disk large enough to hold the movie.

Gsetcopynum (a non-standard routine) sets the number of frames an image will be recorded on. This is used to change the rate of playback if the recording unit is a VTR. For example, to play back at 10 frames per second, **copynum** should be set to 3 (the normal rate is 30 frames per second). This call has no effect if the recording unit is a videodisk which can play back a recorded sequence at various integral multiples or fractions of 30 frames per second.

Gsetcompr (a non-standard routine) sets the compression type. The particular compression is chosen by |'ing (in C) or +'ing (in Fortran) the options **BTC**, **COLORMAP**, **FRAME_FRAME**, and **LEMPEL_ZIV**. **BTC** chooses BTC compression, **COLORMAP** chooses Heckbert's color map algorithm, **FRAME_FRAME** chooses frame-to-frame differencing, and **LEMPEL_ZIV** chooses Lempel-Ziv compression. (The color map algorithm is described in Heckbert, P., Color image quantization for frame buffer display, SIGGRAPH 1982 Proceedings, pp. 297-307.) For example, to use BTC compression with a color map and frame-to frame differencing: **BTC | COLORMAP | FRAME_FRAME**. Use **NONE** by itself if no compression is desired. Lempel-Ziv compression, even if chosen, is only performed if the TCP transmission protocol was selected with **Gopenws**.

Not all combinations of compression are implemented, i.e. **LEMPEL_ZIV**, **LEMPEL_ZIV | FRAME_FRAME**, **FRAME_FRAME**, **COLORMAP | FRAME_FRAME**, **COLORMAP | LEMPEL_ZIV**, and **COLORMAP | FRAME_FRAME | LEMPEL_ZIV** are not implemented. The Sun server only understands a subset of compressed images, i.e. only **BTC | COLORMAP**, **BTC | COLORMAP | LEMPEL_ZIV**, and **COLORMAP**. The default (also chosen in case of error) is **BTC | COLORMAP**.

BTC compression by itself, and any combination of compression with frame-to-frame differencing, is not available under UNICOS. If these are attempted, the default will be chosen instead.

Gupdatews takes the data in the software frame buffer (optionally) compresses it, and sends it to the server to be displayed and (optionally) recorded if the PC server is used. Currently **regfl** has no effect.

SEE ALSO

scry_client(l), scry_gks(3), scry_client.a(3), scry_pc_server(l), scry_sun_server(l)
The compression algorithm references may be found in **scry_client(l)**.

NAME

Gpoint (gpt), Gpolyline (gpl), Gtext (gtxt), Gsetcolourrep (gscr), Gsetptcolourind (gsptci), Gsetlinecolouurind (gsplci), Gsetpgoncolourind (gspgonci), Gsettextcolourind (gstxtci), Gsetpointwidth (gsptwsc), Gsetlinewidth (gslwsc) – display graphics primitives (points, lines, and text). Alternative names are for use from within Fortran.

SYNOPSIS

C	Fortran
#include "gks.h"	**INCLUDE 'gksf.h'**
Gpoint(x,y) **double x, y;**	**CALL gpt(x,y)** **REAL x, y**
Gpolyline(num,x,y) **int num;** **double x[], y[];**	**CALL gpl(num,x,y)** **INTEGER num** **DOUBLE PRECISION x(foo), y(foo)**
Gtext(ndx,ndy,string) **double ndx, ndy;** **char *string;**	**CALL gtxt(ndx,ndy,string)** **REAL ndx, ndy** **CHARACTER string*MAXCHAR**
Gsetcolourrep(wkid,index,hue,sat,val) **int wkid;** **int index;** **int hue;** **double sat, val;**	**CALL gscr(wkid,index,hue,sat,val)** **INTEGER wkid** **INTEGER index** **INTEGER hue** **REAL sat, val**
Gsetptcolourind(index) **int index;**	**CALL gsptci(index)** **INTEGER index**
Gsetlinecolourind(index) **int index;**	**CALL gsplci(index)** **INTEGER index**
Gsetpgoncolourind(index) **int index;**	**CALL gspgonci(index)** **INTEGER index**
Gsettextcolourind(index) **int index;**	**CALL gstxtci(index)** **INTEGER index**
Gsetptwidth(width) **int width;**	**CALL gsptwsc(width)** **INTEGER width**
Gsetlinewidth(width) **int width;**	**CALL gslwsc(width)** **INTEGER width**

DESCRIPTION

Only the C names are given for the following GKS routines; routines to be called from Fortran are interface routines that take care of the details of calling C routines from Fortran and then call the corresponding C-binding routines.

Gpoint (non-standard routine) draws a 2D point at world coordinates (x,y). It is only available with the 2D version.

Gpolyline draws a 2D polyline with **num** points given by arrays **x** and **y** in world coordinates. It is only available with the 2D version.

Gtext (non-standard routine) draws 2D text **string** at NDC coordinates (**ndx**,**ndy**).

Gsetcolourrep sets an entry at location **index** in the color table for workstation **wkid**. The entry is given in the hue, saturation, value (HSV) color scheme. Color table entry 0 is reserved for the background color. The default is white. Color table entry 1 is reserved for the foreground color. The default is black. Other entries may be used to set the point, line, and text colors, unless a monochrome Sun server is used, in which case only the defaults will work.

Gsetptcolourind (non-standard routine) sets the current point color by setting the current point color index into the color table. Do not use with a monochrome Sun server.

Gsetlinecolourind sets the current polyline color by setting the current polyline color index into the color table. Do not use with a monochrome Sun server.

Gsetpgoncolourind (non-standard routine) sets the current polygon color by setting the current polygon color index into the color table. It has no effect in the 2D version, since a 2D polygon primitive is not implemented. Do not use with a monochrome Sun server.

Gsettextcolourind (non-standard routine) sets the current text color by setting the current text color index into the color table. Do not use with a monochrome Sun server.

Gsetptwidth (non-standard routine) sets the current point width in pixels.

Gsetlinewidth sets the current polyline width in pixels.

SEE ALSO

scry_client(l), scry_gks(3), scry_client.a(3)

NAME

Gselntran (gselnt), Gsetwindow (gswn), Gsetviewport (gsvp), Gsetwswindow (gswkwn), Gsetwsviewport (gswkvp), Gsetclip (gsclip), Ginqdisplaysize (gqmds) – GKS coordinate transformations routines (alternative names are for use from Fortran)

SYNOPSIS

C	Fortran
#include "gks.h"	**INCLUDE 'gksf.h'**
Gselntran(trn) **int trn;**	**CALL gselnt(trn)** **INTEGER trn**
Gsetwindow(trn,xmin,xmax,ymin,ymax) **int trn;** **double xmin,xmax,ymin,ymax;**	**CALL gswn(trn,xmin,xmax,ymin,ymax)** **INTEGER trn** **REAL xmin, xmax, ymin, ymax**
Gsetviewport(trn,xmin,xmax,ymin,ymax) **int trn;** **double xmin,xmax,ymin,ymax;**	**CALL gsvp(trn,xmin,xmax,ymin,ymax)** **INTEGER trn** **REAL xmin, xmax, ymin, ymax**
Gsetwswindow(wkid,xmin,xmax,ymin,ymax) **int wkid;** **double xmin,xmax,ymin,ymax;**	**CALL gswkwn(wkid,xmin,xmax,ymin,ymax)** **INTEGER wkid** **REAL xmin, xmax, ymin, ymax**
Gsetwsviewport(wkid,xmin,xmax,ymin,ymax) **int wkid;** **double xmin,xmax,ymin,ymax;**	**CALL gswkvp(wkid,xmin,xmax,ymin,ymax)** **INTEGER wkid** **REAL xmin, xmax, ymin, ymax**
Gsetclip(clipind) **int clipind;**	**CALL gsclip(clipind)** **INTEGER clipind**
Ginqdisplaysize(wkid,xmax,ymax) **int wkid;** **int xmax, ymax;**	**CALL gqmds(wkid,xmax,ymax)** **INTEGER wkid** **INTEGER xmax, ymax**

DESCRIPTION

Only the C names are given for the following GKS routines; routines to be called from Fortran are interface routines that take care of the details of calling C routines from Fortran and then call the corresponding C-binding routines.

See Enderle et al. (referenced in **scry_gks2d.a**(3)) for details on the coordinate systems used in GKS.

Gselntran selects the normalization transformation, which converts from world coordinates to Normalized Device Coordinates (NDC). World coordinates are those of the space the user is working in, and NDC, ranging from 0.0 (lower left corner) to 1.0 (upper right corner), is a device-independent way of placing an object on a certain portion of the display.

Gsetwindow sets the world window. **trn** is the normalization transformation number. **xmin** and **ymin** give the lower left hand corner in world coordinates, while **xmax** and **ymax** give the upper right hand corner. The default is **trn**=0, **xmin**=**ymin**=0.0, **xmax**=**ymax**=1.0. It is not available in the 3D version, being supplanted by Sesetviewvolume.

Gsetviewport sets the Normalized Device Coordinate (NDC) viewport. **trn** is the normalization transformation number. **xmin** and **ymin** give the lower left hand corner of a rectangular region in NDC, while **xmax** and **ymax** give the upper right corner. The default is **trn**=0, **xmin**=**ymin**=0.0, **xmax**=**ymax**=1.0.

Gsetwswindow sets the workstation window. This is a window on the whole NDC screen (there may be several NDC viewports contained in it). **wkid** is the workstation identifier, which can currently only be **FRAMEB** (see **scry_gks_control**(3)). The default is **xmin=ymin**=0.0, **xmax=ymax**=1.0.

Gsetwsviewport sets the workstation viewport. **wkid** is the workstation identifier. The workstation viewport specifies a portion of the actual display device, given in device (integer) coordinates. The default is the whole display. The package automatically scales workstation window into viewport so that the aspect ratio (y:x ratio) of the window is preserved. If the display device is not square and it is desired to use the full range of its coordinates, the workstation window should be set so that it has the same aspect ratio as the display device. Keep in mind that the legal range for the workstation window is (0.0 to 1.0, 0.0 to 1.0).

Gsetclip sets the clipping indicator. It is not available in the 3D version. If **clipind** is 0, clipping is done only against the current NDC viewport. (This is the only option in the 3D version.) If it is 1, clipping is done against the intersection of the current NDC viewport and workstation window.

Ginqdisplaysize returns the range of device coordinates, **xmax** and **ymax**, of the display for workstation **wkid**.

SEE ALSO

scry_client(L), scry_gks(3)

NAME

scan3d.a – 3D scan conversion library

DESCRIPTION

This library is the 3D image generator. GKS routes 3D calls into routines in this library. Scan converts points, lines, 2D text, and polygons to generate an image in the software frame buffer. Polygons are rendered depending on the lighting and shading model, and optionally clipped during scan conversion. Lines and points are colored according to their depth during scan conversion. Hidden points, lines, and polygons are removed using a z-buffer. [The software frame buffer has the same dimensions as the hardware display buffer]. The other component library of the 3D driver, **client.a**, optionally compresses the software frame buffer and transmits it to the server for display.

FILES

Contained in *scry/scandrv*:

pcwork.**h**

pcwork.**c**

scandef.**h**

scan.**h**

comattr.**c**

hershey.**c** All preceding files are shared with **scan2d.a** and descriptions are listed in that section.

Contained in *scry/scandrv/scan3d*:

Makefile	makes *libscan3d*.a. The installation of this library will have to be modified for your site.
color.**h**	color tables
color.**c**	set entries in color tables, clear frame buffer
zbuf.**h**	z-buffer
buffers.**c**	z-buffer algorithm
adj3d.**c**	used in object-oriented clipping
scan3d.**h**	include file for object-oriented clipping
point3d.**c**	renders 3D point with depth intensity cueing into software frame buffer
line3d.**c**	renders 3D line with depth intensity cueing into software frame buffer
text2d.**c**	renders stroke of 2D text into software frame buffer
poly3d.**c**	polygon rendering shared by both Gouraud and constant shading
poly3d.**h**	include file for polygon rendering
const.**c**	renders 3D polygon using constant shading
gouraud.**c**	renders 3D polygon using Gouraud shading

AUTHOR

David Robertson

SEE ALSO

scry_client(l), scry_3D(3), scry_gks(3), scry_client.a(3)

NAME

scry_scan2d.a – 2D scan conversion library

DESCRIPTION

This library is effectively the 2D image generator. The GKS level routes 2D calls into routines in this library to generate an image in the software frame buffer. These routines clip points and lines to the intersection of the current Normalized Device Coordinates viewport and the workstation window, and scan converts remaining points and lines, as well as unclipped text, into the software frame buffer. [The software frame buffer has the same dimensions as the hardware display frame buffer.] The other component of the driver for this approach, **client.a**, optionally compresses the software frame buffer, and transmits it to the server for display.

FILES

Contained in *scry/scandrv*:

scwork.**h**	used in 2D clipping and workstation transformation
scwork.**c**	performs 2D clipping and workstation transformation
scan.**h**	frame buffer and color index declarations
comattr.**c**	conversion from HSV to RGB, setting color indices
hershey.**c**	public domain text scan conversion

Contained in *scry/scandrv/scan2d*:

Makefile	makes *scan2d*.a library. The installation of this library will have to be modified for your site.
color.**h**	color table
color.**c**	clear frame buffer, set color table entries
point2d.**c**	renders 2D point into software frame buffer
line2d.**c**	renders 2D line into software frame buffer
text2d.**c**	renders stroke of 2D text into software frame buffer

AUTHOR

David Robertson

SEE ALSO

scry_client(l), scry_gks(3), scry_client.a(3)

NAME

scry_send_frame, scry_open, scry_close, scry_set_compress, scry_set_record, scry_set_copy_num, scry_set_max_color, scry_set_image, scry_set_map – routines for software frame buffer compression and transmittal

SYNOPSIS

```
#include "clnt.h"
#include "scan.h"

scry_send_frame()

scry_open(host,protocol,prognum)
char *host;
int protocol;
int prognum;

scry_close()

scry_set_compress(option)
int option;

scry_set_copy_num(num_copies)
int num_copies;

scry_set_record(option,length)
int option;
int length;

scry_set_max_color(max)
int max;

scry_set_image(type)
int type;

scry_set_map(num,map)
int num;
unsigned short *map;
```

DESCRIPTION

Routines to take software frame buffer generated in TARGA format, optionally compress it, and transmit it to a server workstation via Sun RPC's for decompression and display. (These are unavailable from Fortran.) The software frame buffer is **unsigned short sc_frame_arr[y][x]**, where y is 400 and x is 512, and is 15 bits/pixel (0,5R,5G,5B). It is declared in *scry/scandrv/scan*.h. An example of converting a red, green, blue pixel color representation into the packed TARGA 2-byte format is given in routine GreyRGB in *scry/samples/greyrgb*.c.

scry_open is called first to set up the connection to the server workstation. **host** is a symbolic reference to the server workstation as explained in **scry_gks_control**(3), e.g. the Internet address of the server system. **protocol** must be either **UDP** or **TCP**. **prognum** is an identifier for the server (see **scry_gks_control**(3)).

scry_close is called at the end of a session to terminate the connection with the server workstation.

scry_send_frame transmits the software frame buffer, optionally compressing it first, to the server workstation.

scry_set_compress sets the compression type. **Gsetcompr** calls this routine when the library is used with GKS and the 2D scan conversion module. See **scry_gks_control**(3) for setting the compression options.

scry_set_record sets the recording mode and only has an effect with the PC server. If **option** is **PREVIEW** the images are displayed on the PC server workstation but not recorded. If **option** is a positive number, frames are recorded starting at the frame on videotape or videodisk identified by that number. If **option** is **AUTOSEEK**, the second argument indicates the number of images in the movie. This only has an effect if the videodisk is used, and is used to seek to the first place on disk large enough to hold the

movie.

scry_set_copy_num only has an effect if the PC server workstation is equipped with a VTR (as in adv-pc-2). **num** is the number of frames to record the image on. This is used to change the speed of playback of the movie. For example, to play back at 10 frames per second, **num** is set to 3. (The normal speed of playback is 30 frames per second.) If the PC server workstation is equipped with a videodisk player (as in adv-pc-1), this call has no effect. The videodisk player can play a movie at various integral multiples or fractions of 30 frames per second.

Three new low-level routines are provided in this distribution. **scry_set_max_color** sets the maximum number of color table entries. The largest that can be set is 256. This routine is useful in conjunction with use of the Sun server (see **scry_sun_server**(l)). If the maximum is 128 or less, the image can be viewed in a Sun window even when the cursor is not in the server window. The default, 128, is chosen when the higher level 3D and GKS modules are used.

scry_set_image sets the image type to be either **TARGA_IMAGE** or **MAPPED_IMAGE**. The default, **TARGA_IMAGE**, is chosen when the higher level modules are used. An image in TARGA format can be compressed before transmission. IF **MAPPED_IMAGE** is chosen, the pixels in the image provided by the user are in the form of indices into a color map, also provided by the user. Each pixel is an **unsigned char** rather than an **unsigned short**. See **greyrgb**(l) for a description of the program illustrating the usage of the **MAPPED_IMAGE** option.

If **MAPPED_IMAGE** is chosen, the color map is provided by the user with **scry_set_map**. Do not use this routine if **TARGA_IMAGE** is the current mode. Be sure the number of entries is consistent with that chosen by **scry_set_max_color**.

AUTHORS

David Robertson, Nicole Texier, James Huang, and Bill Johnston

SEE ALSO

scry(l), scry_client(l), scry_pc(l), scry_gks_control(3), scry_client.a(3)

NAME

scry_client.a – library for software frame buffer compression and interprocess communication with the server

DESCRIPTION

This is the lowest level of the Scry client. Takes software frame buffer generated by routines in **scry_scan2d.a**(3) or **scry_scan3d.a**(3), optionally compresses it, and transmits to the Scry server workstation via Sun RPCs for decompression and display. If record mode is set, controls recording as well. Over a local-area network, Block Truncation Coding combined with color map compression and frame-to-frame differencing is usually used, since it is the least CPU consuming option (the other options are discussed in **scry_gks_control**(3) and **scry_send_frame**(3)). Over a wide-area network we usually use a further compression step, Lempel-Ziv compression, as well. Select Lempel-Ziv compression with **Gsetcompr** (see **scry_gks_control**(3)), and choose the TCP protocol in **Gopenws**. (The PC based Scry server will have to be modified if the 68020 coprocessor is not present on the PC. It is not practical to do LZW decompression on the PC without the coprocessor. See **scry_pc_server**(3).)

This library can also be used apart from *scan2d.a* or *scan3d.a*. As long as the software frame buffer is in TARGA format, it does not care how the images were generated. See **scry_send_frame**(3) and **greyrgb**(l) for information on the TARGA format, and an example of how to use the low-level interface to the routines in this library.

FILES

Contained in *scry/scandrv/client*:

Makefile	makes library *libclient*.a. The installation of this library will have to be modified for your site.
clnt.**h**	user definitions for send_frame
clntint.**h**	definitions and declarations for this library
control.**c**	controls communication with the Scry server
record.**c**	controls recording
btcalg.**c**	BTC compression
btcbuf.**c**	building up compressed buffer to send as final step of BTC, or BTC and color map compression
btcsnd.**c**	compresses (optionally) and transmits the software frame buffer
colalg.**c**	general color map compression algorithm
colmap.**h**	definitions and declarations for color map compression
colsnd.**c**	sends the image using only color map compression
auxsnd.**c**	routines that aid the routines in *btcsnd.c* and *colsnd.c*
lzw.**c**	Lempel-Ziv compression

SEE ALSO

scry(l), scry_client(l), scry_scan3d.a(3), scry_scan2d.a(3), scry_send_frame(3)

LAWRENCE BERKELEY LABORATORY
TECHNICAL INFORMATION DEPARTMENT
1 CYCLOTRON ROAD
BERKELEY, CALIFORNIA 94720