Deep Computing Visualization

Installation and User Guide

Version 1 Release 3

G224-9183-02

Deep Computing Visualization

Installation and User Guide

Version 1 Release 3

G224-9183-02

Note:
Before using this information and the product it supports, read the information in “Notices” on page 97.

Third edition (January 2007)

| This edition replaces G224-9183-01 for program number 5724-K69. Significant changes or additions to the text or
| illustrations are indicated by a vertical line (|) to the left of the change.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you can address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P181
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America
FAX (United States & Canada): 1+845+432-9405
FAX (Other Countries):
Your International Access Code+1+845+432-9405
IBMLink (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:
- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

About this document

This document describes the IBM® Deep Computing Visualization (DCV) software offering. It describes how to use DCV to:

- Enhance graphical rendering of complex applications
- Provide high-performance visualization across low-bandwidth, high-latency networks to remote or distributed locations

This document is divided into two parts:

- Part 1 provides user information for working with DCV software, including:
  - Using Scalable Visual Networking (SVN)
  - Using Remote Visual Networking (RVN)
  - Setting up combined SVN and RVN sessions
  - Information about SVN and RVN command options
- Part 2 provides information for installing DCV software, including:
  - Installation prerequisites, including setting up the MPI library for SVN
  - Installing DCV
  - Installation verification and problem determination
- Appendixes provide information about messages, environment variables, and programming considerations.

Save this book

Retain this book with your original system. This book emphasizes recent system information and does not include complete information about previous releases.

Who should use this document

This document is designed for:

- End users of applications that are able to utilize the features provided by DCV software
- System administrators responsible for installing DCV software

Typographic conventions

This document uses the following typographic conventions:

- Commands appear in bold, monospace font when they appear in text.
  - Example: **logon**
- File and directory names are italicized when they appear in text.
  - Example: */etc/passwd*
- Variables for user-supplied information on commands are indicated by italics.
  - Example: **telnet** *app_host*
- Literals in commands appear in monospace font:
  - Example: `rpm -i --force` *`abc.rpm`*
- Greater than and less than symbols (< >) indicate that you must make a substitution.
  - Example: `<username>` indicates that you must supply your own username value.

- Square brackets ([ ]) indicate that the enclosed values are optional.
- Curly brackets ({ }) indicate that you must select one of the enclosed values.
- Multiple options for a value are separated by a vertical bar (|).
- Default values are underlined.

Related information

For the latest information about Deep Computing Visualization (DCV), refer to the documents at:

http://www.ibm.com/servers/deepcomputing/visualization/

http://techsupport.services.ibm.com/server/cluster/home.html

Accessibility information

Accessibility information for IBM products is available online at the IBM Accessibility Center. Go to http://www.ibm.com/able/, then click **Product accessibility information**.

User's responsibilities

Before contacting IBM for service, the system administrator should refer to the README files provided on the DCV page listed under "Related information" and then, if necessary, use the problem determination procedures described in this document for initial problem determination. If the recommendations described in the README have been followed and there is nothing wrong with the customer operating procedures, the customer should then call IBM for service.

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book:

- Send your comments by e-mail to **mhvrcfs@us.ibm.com**. Be sure to include:
  - The name of this book
  - The order number of the book
  - If applicable, the specific location of the text you are commenting on (for example, a page number or table number)
- Fill out one of the forms at the back of this book and return it by mail, by fax, or by giving it to an IBM representative.

What's new

G224-9183-02

Changes made since the previous edition (G224-9183-01) are indicated throughout this book by a vertical bar on the left margin of the page. Changes in this edition include:

- **New information:**
  - Added information about VNC Visualization Edition.
    - Use of this version of VNC improves the usability of the interface to DCV. For example, it eliminates the need for specifying an additional conference ID and conference key when joining an RVN session.
    - VNC Visualization Edition is now the only supported VNC software (other VNC products from RealVNC or other companies are not supported).
  - Added support for RVN on application servers that run under Microsoft® Windows®.
    - VNC Visualization Edition must be used for RVN sessions with these application servers. X11 export mode is not supported.
    - The new **rvn_sender.bat** command can be used from these application servers, similar to the **rvn_sender** command for Linux® application servers.
  - Added information about the RVN launcher, a graphical user interface for Linux and Windows users that can be used instead of the **rvn_sender** or **rvn_sender.bat** command.
  - Added information about SVN support for DMX servers.
  - Added information about the **rvn_viewer** command, which is used instead of the **vncviewer** command by Linux end stations to start the VNC viewer.
- **Updated information:**
  - Revised the information about X11 export mode for RVN sessions to clarify that it can be used only for application servers running under Linux.
  - Updated the description of the RVN dashboard to reflect the new graphical user interface for Linux and Windows.
  - Noted that SVN supports OpenGL up through specification level 2.0, including support for shader programming.
  - Changed references to the SSC component of RVN to *accelerated graphics*.
  - Corrected the name of the `SVN_SYNC_ON_RETRACE` environment variable. The correct name is `SVN_SWAP_ON_RETRACE`.
  - Moved information about environment variables to an appendix.
  - Moved information about programming considerations for SVN and RVN to an appendix.
- **Deleted information:**
  - Removed the requirement for the pthreads compatibility library when running RVN under Windows.
  - Removed information about Microsoft Windows 2000 as a platform for RVN end stations.
  - Removed the `RVN_DASHBOARD_DELAY` and `RVN_DASHBOARD_PERMIT_CLOSE` environment variables.

G224-9183-01

Changes in this edition include:

- **New information:**
  - Added RVN end station support for xSeries® nodes using Microsoft Windows 2000 or Microsoft Windows XP
  - Added support for simultaneous SVN and RVN sessions
  - Added support for RVN nodes to dynamically join and leave a session
  - Added the ability to override OpenGL functions with user-implemented modules
- **Updated information:**
  - Revised SVN commands
  - Revised RVN commands
  - Revised installation procedures
- **Deleted information:**
  - Removed `-any` and `-loop` options for the **rvn_receiver** command
  - Removed `-n` option and `endstation` list for the **rvn_sender** command
  - Removed `RVN_LISTEN_ANY` and `RVN_LISTEN` environment variables

Part 1. Using IBM Deep Computing Visualization software

Deep Computing Visualization (DCV) software allows you to display complex visual data:

- On massive local displays capable of immersing the audience in stereo visualizations
- Across distributed displays using existing low-bandwidth networks

Part 1 of this book provides information for end users of applications that are able to utilize the enhancements provided by DCV software. This information includes:

Chapter 1. Introduction to visual networking

In a typical visualization scenario, a software application running on a workstation sends a stream of graphics commands to a graphics adapter installed in one of the I/O interfaces of the system. The graphics adapter then renders the data into pixels that are stored as raster content in the video memory of the adapter and outputs them to the local display as a video signal. Many areas of scientific or engineering data visualization can benefit from the ability to expand or extend the representation of three-dimensional (3-D) graphics beyond a simple local display. Two useful alternative modes of display are:

- Raster representations of high-performance graphics across a composite array of display devices (display wall)
- Raster representations of high-performance graphics on a remote display connected by a network communication link

Deep Computing Visualization (DCV) enables both of these alternative display modes by virtualizing the local display through the use of high-speed local networks linking a rendering cluster, or slow-speed, wide-area networks linking a display end station. In both cases, DCV works by inserting an *intercept* OpenGL library into the graphics software stack on a host system running a 3-D graphics application. This inserted library intercepts the function calls of the application to the OpenGL API, and transports appropriate data from the graphics pipeline over a network.

- In the high-speed, local-area case, scene geometry and graphics state are transmitted to a set of nodes for rendering. This is referred to as *Scalable Visual Networking* (SVN).
- In the slow-speed, wide-area case, scene geometry and graphics state are rendered locally, and pixels are sent to remote end stations. This is known as *Remote Visual Networking* (RVN).

An SVN-enabled application virtualizes the rendering pipeline over a high-speed, local-area network. For SVN configurations, you must install the intercept OpenGL library on a server running a 3-D visualization application. The library accepts the OpenGL calls made by the application, encodes them, and broadcasts the call data to a cluster of rendering servers using the Message Passing Interface (MPI). Each member of the cluster receives the call data, decodes it, and calls the OpenGL library on the rendering server. This allows each member of the cluster to render and display its portion of the final image as a tile of a display wall or projection system.

An RVN-enabled application allows for remote interaction with a 3-D scene across a slow-speed network, such as the Internet. As with SVN, the OpenGL calls are intercepted. However, the 3-D geometry is rendered locally, and a (possibly) compressed pixel stream is transported to a remote end station for display.

Notes:

1. If you require additional security, the SVN transport mechanism supports the use of Secure Shell (OpenSSH).
2. For information about the hardware and software requirements for SVN application hosts, refer to Chapter 6, “Prerequisites for installing Deep Computing Visualization,” on page 51 and “SVN prerequisites” on page 52.

3. For information about the hardware and software requirements for RVN application servers and end stations, refer to Chapter 6, “Prerequisites for installing Deep Computing Visualization,” on page 51 and “RVN prerequisites” on page 55.

This document details the installation, configuration and operation of the SVN and RVN components of DCV. It also describes the interaction with software components that work with the DCV system, such as a communication library implementing the MPI standard for SVN, or VNC to enable remote desktopping with RVN. For more information about using DCV, refer to:

- Chapter 2, “Scalable Visual Networking,” on page 5
- Chapter 3, “Remote Visual Networking,” on page 21
- Chapter 4, “SVN and RVN interoperability,” on page 45
- Chapter 5, “SVN support for DMX servers,” on page 47

Chapter 2. Scalable Visual Networking

Scalable Visual Networking (SVN) converts a 3-D OpenGL display running on a single application workstation or server into a multi-tile display running on a cluster of supported servers. For information about the supported hardware for SVN configurations, refer to Chapter 6, “Prerequisites for installing Deep Computing Visualization,” on page 51 and “SVN prerequisites” on page 52.

Note: Figure 1 provides an example of the hardware configuration required for SVN.

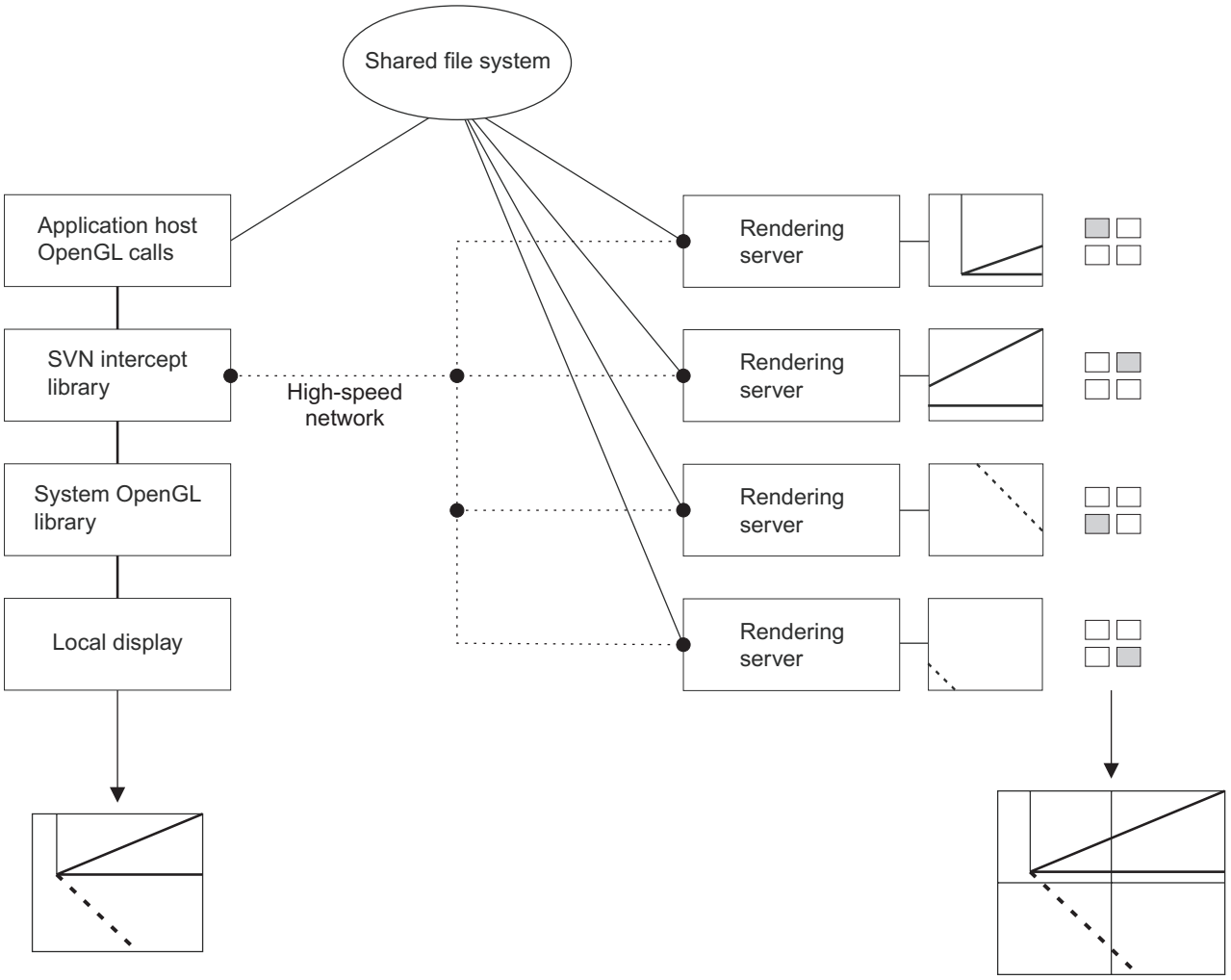

Figure 1. Hardware configuration for Scalable Visual Networking

Running an application with SVN involves these steps:

1. Define which part of the scene geometry is to be displayed by which rendering server, also called *tiling* (described later in this section).

2. Set environment variables to locate the SVN libraries and the MPI environment used to transport the encoded geometry between the application host and the rendering servers (described in “Environment variables for svn_sender” on page 85).
3. Run the **svn_sender** command to invoke the application (described in “Command-line options for svn_sender” on page 8 and “Starting SVN” on page 11).

The tiling is defined in a *wall configuration* file, which is a text file with one entry for each rendering server. You can name the wall configuration file anything you want. This is the format of a server entry:

```
server_name[:d.s] vwidth vheight x_offset y_offset width height [display_group]
```

This is the definition of the syntax for a server entry:

- `server_name[:d.s]` is the X display name for the rendering server.
  - If a private network is being used, `server_name` should be the private IP alias or dotted decimal address.
- `d` is the server instance and `s` is the screen number.
  - If `:d.s` is not supplied, the default is `:0.0`.
- `vwidth` and `vheight` are the width and height of the entire (multi-tiled) display surface.
- `x_offset` and `y_offset` are the starting point of the lower-left corner of the display area (tile) for this server within the entire display surface.
- `width` and `height` are the total width and height for the tile assigned to this server.
- `display_group` allows you to group individual displays into multiple display walls, and to group rendering servers so that each group acts as a single logical display.
  - The default value for this field is `1`.
  - Display group 1 is the reply display that returns results to the application as needed.
  - If you define display groups, one of these groups must be defined as display group 1.

Notes:

1. All values are expressed in pixels at the pixel pitch of the individual server.
2. If needed, the display areas specified among servers can overlap, and the pixel pitch of each tile can be different.
3. Lines starting with the # symbol are ignored, as are blank lines.
4. You must create a wall configuration file before running your application under SVN control.
5. The wall configuration file must be in a shared file system that is readable by the application host and by all rendering servers.
6. You must identify the wall configuration to SVN through either an option on the **svn_sender** command or the related environment variable. For more information, refer to “The svn_sender command” on page 8.

Example 1: Wall configuration file to define a symmetric display

This example of a wall configuration file defines a virtual display area of 3120 x 2048 pixels. The display area has four tiles, with each tile having an

area of 1560 x 1024 pixels. DCV arranges the tiles in a 2 x 2 array with the top two tiles being separate displays on the node `render56` (refer to Figure 2).

```
render54:0.0  3120 2048 0      0      1560   1024
render55:0.0  3120 2048 1560   0      1560   1024
render56:0.0  3120 2048 0      1024   1560   1024
render56:0.1  3120 2048 1560   1024   1560   1024
```

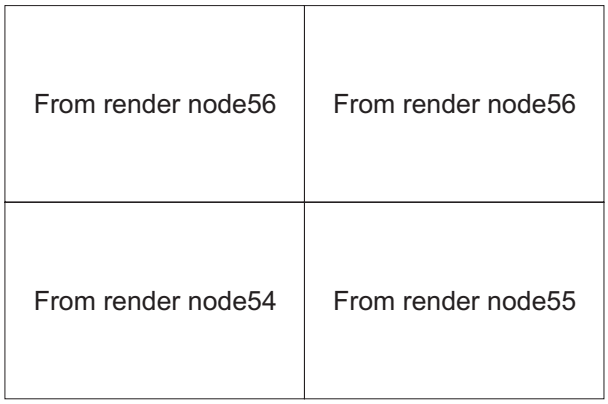

Figure 2. Virtual display with 2 x 2 tile array

Example 2: Wall configuration file to define display groups

Display groups enable multiple rendering servers to act as a single logical display. Using display groups has two effects:

- Operations that require a collective function across the servers forming the logical display (such as `glCopyPixels`) can be handled within the logical display.
- By assigning multiple servers to display group 1, you include them in the specific logical display that replies to OpenGL queries from the application.

Therefore, if the system issues a call to `glReadPixels`, pixels are sent to the application host from each server in display group 1. Pixels are not sent by any nodes driving portions of other logical displays.

To assign servers to SVN display groups, specify the `display-group` identifier in the wall configuration file. Make sure to use `1` as the first identifier, and number additional display groups sequentially.

This example of a wall configuration file uses display groups to define multiple display walls.

```
srvr1:0.0   1280   1024   0     0     1280   1024   1
srvr2:0.0   2560   2048   0     0     1280   1024   2
srvr3:0.0   2560   2048   1280  0     1280   1024   2
srvr4:0.0   2560   2048   0     1024  1280   1024   2
srvr5:0.0   2560   2048   1280  1024  1280   1024   2
```

Example 3: Wall configuration file to define overlap for a projector

This example of a wall configuration file defines a projector display that requires some overlap.

- Horizontally, vis22 overlaps vis23, and vis24 overlaps vis25.
- Vertically, vis24 overlaps vis22, and vis25 overlaps vis23.

```
vis22:0.0 2360 1848 0     0   1280 1024
vis23:0.0 2360 1848 1180  0   1280 1024
vis24:0.0 2360 1848 0     924 1280 1024
vis25:0.0 2360 1848 1180  924 1280 1024
```

The svn_sender command

The **svn_sender** command is used to start SVN and run the visualization application on the application host.

This is the syntax of the **svn_sender** command:

```
[</opt/IBM/dcv/svn/bin/>]svn_sender [<svn_sender_options>] <appname> [<app-args>]
```

The **svn_sender** command analyzes the options that were specified on the command, as well as the settings of related environment variables.

- For a description of the command-line options, see "Command-line options for svn_sender."
- For a description of the related environment variables, see "Environment variables for svn_sender" on page 85.
- For examples of using the **svn_sender** command and related environment variables, see "Starting SVN" on page 11.

Command-line options for svn_sender

This section defines the options that can be specified on the **svn_sender** command. Some of these options have a corresponding environment variable that you can set before running **svn_sender**.

Note: If you set both the command-line option and its corresponding environment variable, the command-line option takes precedence.

-clientdpy {*host-display-suffix* **| :0.0}**

The `-clientdpy` option specifies the X display that the client should use for the local display. The host display suffix information (:1.0, :0.0) can be useful if multiple screens are available to the application host.

Notes:

1. The default for this option is `:0.0`.
2. For information about using this option for DMX servers, refer to Chapter 5, "SVN support for DMX servers," on page 47.

-clienthost {*host-alias-address* **|** *application-host-name***}**

The application always runs on the node on which you run the **svn_sender** command. However, the `-clienthost` option allows the MPI implementation to use an alternate IP address to identify the source for the broadcast graphic data. For example, if the application host has multiple network adapters and IP addresses, the `clienthost` option can specify which IP address or adapter should be used for sending the graphics data to the rendering servers.

Note: If you do not supply the `clienthost` information, the **svn_sender** command uses the results of the **hostname** command as the MPI node name to be used as the source for the broadcast graphic data.

-clientrender {0 | 1}

The `-clientrender` option forces client-side image rendering. To the application, this makes it appear as if SVN is not being used. This option

causes the SVN OpenGL library to pass all application OpenGL instructions to the local OpenGL subsystem as well as the SVN server nodes.

Notes:

1. The `-clientrender` option is enabled by default.
2. If you export the application GUI to a remote display, you must use GLX to ship the OpenGL commands to the remote X server. This will have a substantial effect on performance. You can reduce the effect by using RVN or, if the system is connected by InfiniBand, by specifying `-svnrvn` on the **svn_sender** command on the remote system. For more information, refer to Chapter 4, "SVN and RVN interoperability," on page 45.

-firstwindowonly {0 | 1}

The `-firstwindowonly` option provides a solution for an NVIDIA graphics driver limitation. The current NVIDIA drivers require this option because they do not support multiple OpenGL windows being framelocked across systems. If you are using NVIDIA graphics drivers, opening multiple windows on the rendering servers prevents `framelock` from working. If you enable the `-firstwindowonly` option, the SVN intercept library sends only the OpenGL information related to the first OpenGL window to the rendering servers. The `-firstwindowonly` option guarantees that only the first window is created on the rendering servers, and prevents the window selector function from taking effect. In other words, this allows framelocking on the first window. As a result, only the first window will ever be visible on the rendering servers.

Notes:

1. The `-firstwindowonly` option is disabled by default.
2. The `-firstwindowonly` option can also be enabled by setting the `SVN_FIRST_WINDOW_ONLY` environment variable to 1.
3. This feature is not compatible with the `SVN_INITIAL_SCALED_WINDOW` environment variable and does not have any effect when `SVN_INITIAL_SCALED_WINDOW` is set to any value other than 1.

-mpicomm

SVN provides startup scripts to initialize the MPI subsystem for supported implementations. To use these scripts, specify the appropriate value on the `-mpicomm` option. These are the supported values for `-mpicomm`:

- `mpicomm.ib.rsh`: Initializes MPI for InfiniBand and the rsh command for rendering server access
- `mpicomm.ib.ssh`: Initializes MPI for InfiniBand and OpenSSH commands for rendering server access
- `mpicomm.p4`: Initializes an MPI implementation that has the same startup mechanism as the publicly available MPICH implementation from Argonne National Laboratory
- `mpicomm.scali`: Initializes MPI for Scali MPI over Infiniband

Notes:

1. The startup script is required. You must specify it using one of these methods:
   - The `-mpicomm` option on the **svn_sender** command
   - The `SVN_MPICOMM` environment variable

2. The scripts listed for this option are not the MPI startup scripts provided by the vendor of your MPI implementation. However, the scripts that are listed invoke the MPI startup scripts provided by the vendor.
3. If MPI was not installed in the default location, you must update the MPI scripts accordingly.

-mpitest {0 | 1}

If you enable the `-mpitest` option, the specified program (*application.name*) is run on all nodes as a Single Program Multiple Data (SPMD) MPI program. Any application options are passed to the program on the application host node only. This provides a test mechanism for verifying that MPI is running correctly within the SVN environment. Refer to Chapter 9, "Installation verification for Deep Computing Visualization," on page 65 for SVN verification procedures.

Note: The `-mpitest` option is disabled by default.

-svndpy *wall-configuration-file-full-path*

The `-svndpy` option defines the full path for the wall configuration file. The wall configuration file lists the names of the rendering servers, the virtual size of the display wall, and the portion of the geometry to be shown by each server. The **svn_sender** command uses all the (non-commented) rendering servers supplied in the wall configuration file. For more information about the wall configuration file, including display groups, refer to Chapter 2, "Scalable Visual Networking," on page 5.

Note: The full path for the wall configuration file is required. You must specify it using one of these methods:

- The `-svndpy` option on the **svn_sender** command
- The `SVN_DISPLAY` environment variable

-svnrvn *hostname*

The `-svnrvn` option identifies the host name for the machine that will serve as the RVN sender when you run DCV in interoperability mode, using a joint SVN-RVN session. In a joint SVN-RVN session, end stations connect to the specified machine to received the rendered images from the remote application, just as they would in an RVN-only session. For more information about setting up a joint SVN-RVN session, refer to Chapter 4, "SVN and RVN interoperability," on page 45.

The type of connection affects what you must specify on the `-svnrvn` option, as well as whether you must include or exclude other options on the **svn_sender** command.

- **VNC:**
  - Specify the `-v` option to use VNC.
  - On the `-svnrvn` option, specify the SVNRVN server as an X display name, for instance `-svnrvn myhost:0 /bin/myapp`.
  - Do not specify the `-clientdpy` option.
- **X11 mode:**
  - Specify `-conf` *`confID`* to identify the conference ID string for the SVN-RVN session. This information must be conveyed to each participant before the start of the conference.
  - Specify `-key` *`confkey`* to identify the conference access key for the SVN-RVN session. This information must be conveyed to each participant before the start of the conference.

- On the `svnrvn` option, specify the host name of the server node used as a SVNRVN server (and which will be the target of the `rvn_receiver` command), for instance `-svnrvn myhost`.
- On the `clientdpy` option, specify the X display that is used to render the application graphics, for instance `-clientdpy myhost:0 /bin/myapp`.

-sync {0 | 1}
If you enable `-sync`, the frame updates of the host and rendering servers are synchronized so that the host does not render frames ahead of the rendering servers.

Note: The `-sync` option is disabled by default.

-windowselector { 0 | 1 }
If you leave `-windowselector` enabled, the SVN intercept library displays a small X Window on the local display screen. This allows you to control which OpenGL application window should be displayed on the rendering servers. To use this feature:

1. Click the selector window and highlight its frame.
2. Click in the selector window.
3. The selector window turns red and displays the mouse cursor as a small plus sign (+).
4. Click in the window that you want to display.
5. The rendering servers display the indicated window.

Notes:

1. If you have several coincident windows, this process might have to be repeated.
2. You can also enable the window selector by setting the `SVN_WINDOW_SELECTOR` environment variable.
3. The default value for `-windowselector` is `1` (window selector enabled).

Starting SVN

To start SVN, you set the required environment variables, and use the **svn_sender** command to run the visualization application on the application host. The method that you use to start SVN depends on the MPI configuration for your system. For examples, see the following sections:

- "Starting SVN using Scali MPI"
- "Starting SVN using Cisco/TopSpin MPI" on page 12
- "Starting SVN using MPICH" on page 12

For details about the **svn_sender** command and the related environment variables, see "The svn_sender command" on page 8.

Starting SVN using Scali MPI

The following example illustrates the environment variables and the commands needed to invoke SVN using Scali MPI. This example displays a 64-bit sample application on the nodes defined in the `wall.cfg` file.

```
$ export SVN_HOME=/scratch/jr/rpmtest64
$ export SVN_DISPLAY=$SVN_HOME/wall.cfg
$ export SVN_MPILIB=/opt/scali/lib64/
$ export SVN_MPIBIN=/opt/scali/bin
$ export SVN_MPICOMM=mpicomm.scali
$ svn_sender app64
```

Starting SVN using Cisco/TopSpin MPI

The following example illustrates the environment variables and the commands needed to invoke SVN using the Cisco/TopSpin MPI. This example displays a 64-bit sample application on the nodes defined in the `wall.cfg` file.

```
$ export SVN_HOME=/scratch/jr/rpmtest64
$ export SVN_DISPLAY=/scratch/jr/rpmtest64/wall.cfg
$ export SVN_MPILIB=/usr/local/topspin/mpi/mpich/lib64
$ export SVN_MPIBIN=/usr/local/topspin/mpi/mpich/bin
$ export SVN_MPICOMM=mpicomm.ib.rsh
$ svn_sender app64
```

Starting SVN using MPICH

The following example illustrates the environment variables, the wall configuration file, and the commands needed to invoke SVN using MPICH. This example displays a 32-bit sample application on a 2 x 2 wall.

Notes:

1. This example assumes that you installed MPICH in `/usr/local/mpich-1.2.7`. Depending on how MPICH was installed on your system, that location might be different.
2. For information about the level of MPICH that is required, refer to “SVN prerequisites” on page 52.

```
$ cd  /scratch/jr/rpmtest
$ export SVN_HOME=/scratch/jr/rpmtest
$ export SVN_DISPLAY=$SVN_HOME/wall.cfg
$ export SVN_MPILIB=/usr/local/mpich-1.2.7/lib/shared
$ export SVN_MPIBIN=/usr/local/mpich-1.2.7/bin
$ export SVN_MPICOMM=mpicomm.p4
$ cat $SVN_DISPLAY
node3:0.0 2560  2048 0       0 1280 1024
node4:0.0 2560  2048 1280    0 1280 1024
node5:0.0 2560  2048 0    1024 1280 1024
node6:0.0 2560  2048 1280 1024 1280 1024
$ svn_sender /usr/X11R6/bin/glxgears
```

OpenGL overloads with SVN

SVN uses built-in programs called *overloads* to initiate several functions, such as creating a multi-tiled output for a single display application. You can also create your own overloads to modify the behavior of any OpenGL function. Using these custom overloads, you can affect the behavior of specific applications during an SVN session. In addition, you can also group individual overloads together and create an *overload set*. With an overload set, the individual overloads work in sequence to achieve specific application functions.

Notes:

1. To use an overload or an overload set, you must load it with SVN at run time.
2. For sample overload files, look in /opt/IBM/dcv/svn/examples/overload.

All SVN overloads and overload sets achieve their results by intercepting OpenGL calls. After the call is intercepted, the overload replaces the information in the call

with the modified instructions contained in the overload. The overload process works because the SVN graphics pipeline follows a specific sequence of events:

1. The SVN OpenGL library installed on the application host intercepts OpenGL calls.
2. SVN encodes the OpenGL calls and, using MPI, sends the encoded calls over the network to the rendering servers.
3. The rendering servers receive and decode the data.
4. Each of the rendering servers displays its portion of the final image.

At various stages of this pipeline you can modify the parameters of an intercepted OpenGL call. This process is called *overloading* an OpenGL call.

The basic overload

The following example introduces a simple overload that intercepts an OpenGL call and modifies its parameters to invert colors. The mechanism for this overload functions by intercepting all calls that the target application makes to `glColor**` and modifying the associated OpenGL parameters.

Note: In this example, the overload version of `glColor**` refers to `glColor3f`. However, this example easily extends to similar calls.

```
static void
o_glColor3f(float original_r, float original_g, float original_b)
{
float modified_r = 1 - original_r;
float modified_g = 1 - original_g;
float modified_b = 1 - original_b;

//call system OpenGL glColor3f passing it as arguments
//modified_r, modified_g, modified_b
}
```

Now, SVN must fill in the missing call to the system OpenGL `glColor3f` and make sure that any application call to `glColor3f` is diverted to the overloaded version, `o_glColor3f`. Before we proceed, however, we need to go into more detail regarding SVN behavior when loading and using a custom set of overloads. The main requirement for the custom overload set is that it must implement the `Overload` function, whose prototype is:

```
void Overload(glOp *glOperationsTable, glOp *glSystemOperationsTable);
```

When loading the custom overload set, SVN automatically calls the `Overload` function contained within the set and passes it pointers to an array of current SVN OpenGL operations and an array of the original system OpenGL operations. These arrays, especially the SVN OpenGL operations table, play a central role in the overload mechanism:

- The entry in the SVN operations table that corresponds to `glColor3f` provides a handle that calls the current SVN `glColor3f` from the overloaded version `o_glColor3f`.
- By replacing the entry in the SVN operations table that corresponds to `glColor3f` with a new entry that points to our overload version `o_glColor3f`, SVN always calls the overload version instead of the original.
- The array of original system OpenGL operations is provided in case the newly written overload needs to make OpenGL calls using the system OpenGL library rather than the current (possibly overloaded) versions that SVN is using.

To access these arrays, SVN provides a series of numeric constants, and each of these constants corresponds to an OpenGL call. For example, `glOperationsTable[GLCOLOR3F]` provides access to the entry that corresponds to `glColor3f`. All these constants have the form of the OpenGL call they are referring to, and the reference *must* be written in capital letters. In addition, you must include the file `wire.h` in the source file for the overload module.

Note: The `wire.h` file is located in the directory `$SVN_ROOT/include/wire.h`.

Using these requirements, you can complete the overload code file:

```
#include "wire.h"

static void(*s_glColor3f)(float,float,float);
static void   o_glColor3f(float,float,float);

//this function will be automatically called by SVN:
void Overload(glOp *glOperationsTable, glOp *glSystemOperationsTable)
{
  // save a handle to the system glColor3f(...) call:
  s_glColor3f = (void(*)(float,float,float)) glOperationsTable [GLCOLOR3F];

  //replace the entry with a handle to the overloaded version:
  glOperationsTable[GLCOLOR3F] = (glOp)o_glColor3f;
}

static void
o_glColor3f(float original_r, float original_g, float original_b)
{

  float modified_r = 1 - original_r;
  float modified_g = 1 - original_g;
  float modified_b = 1 - original_b;

  (*s_glColor3f)(modified_r,modified_g,modified_b);
}
```

Note: For this overload to operate, the overload `o_glColor3f(..)` calls the OpenGL version of the `glColor3f(..)` function with modified parameters. Although this is the common method, such behavior is not always required. In some cases, an overload might need to call different OpenGL methods or maybe none whatsoever. As long as the overload produces the desired output, this is perfectly acceptable.

After you finish writing the overload, you need to configure SVN to load and use the code. Follow these steps:

1. Compile the overload code into a named, shared object (for example, *invert_colors.so*).
   - You can use the sample *Makefile* provided with the SVN code.
2. Create a file named *overload_files.txt* that contains a list of overload modules for SVN to use.
   - The `overload_files.txt` file has one line for each overload module.
   - The format for each entry has the form */...correct_path.../invert_colors.so*.
3. Set the `$SVN_OVERLOAD_FILE` environment variable to point to `overload_files.txt`.
   - This variable might require the full path to the text file.

After you complete these steps, run SVN as you normally would, on an application that uses `glColor3f`. You can use the provided `rotvp` test application for this purpose. All the colors in the application should be the opposite of the normal colors displayed during standard operation.

System interface for overloads

The basic overload described in the previous section performs the same in all cases. However, more complex overloads (like the one in this section) might need to differentiate between SVN instances. In this case, the overload set must obtain information that is specific to the calling instance of SVN (client or one of the servers). This is accomplished through an internal SVN function `DVGetDisplayName()` that returns the name of the X display used by the calling instance of SVN. Here is the syntax for using this function:

```
char *DVGetDisplayName();
```

Note: Overload functions will be called by all SVN instances, client and servers.

This call returns a pointer to a string that is set as follows:

- On an SVN server, the return value of `DVGetDisplayName()` points to a string containing the name of the X display that it outputs to.
  - This string has the format *server_name[:d.s]*.
  - The display name is identical to the one found in the file that the environment variable `$SVN_DISPLAY` points to.
- On the SVN client, the return value of `DVGetDisplayName()` points to a string containing the word "client".
- If the `DVGetDisplayName()` call fails, the result will be `NULL`.

In order to call this function, an overload set must use the `glOperationsTable` array method required for making system OpenGL calls. The entry in this array corresponding to index `DVGETDISPLAYNAME` provides a handle to the function:

```
char*(*DVGetDisplayNameHandle)() =
  char*(*)()) glOperationsTable[DVGETDISPLAYNAME];
char* myDisplayName = DVGetDisplayNameHandle();
```

Note: The `DVGETDISPLAYNAME` constant is also declared in the file *$SVN_ROOT/include/wire.h*. The `wire.h` file needs to be included in the overload source code file.

For example, assume that you want to invert application colors on the SVN servers only. In this case, the new version of the `Overload(...)` function would be:

```
void Overload(glOp *glOperationsTable, glOp *glSystemOperationsTable)
{
char*(*DVGetDisplayNameHandle)() =
  char*(*)()) glOperationsTable[DVGETDISPLAYNAME];
char* myDisplayName = DVGetDisplayNameHandle();

if (myDisplayName==NULL)
{
 fprintf(stderr,"Failed to obtain SVN system info;\n");
return;
}
else if ( !strcmp(myDisplayName,"client") )
{
 fprintf(stderr,"SVN client: aborting color inversion overload...;\n");
return;
}
```

```
fprintf(stderr,"SVN server rendering on display %s
 using color inversion overload... \n", myDisplayName);

s_glColor3f = (void(*)(float,float,float)) glOperationsTable [GLCOLOR3F];
glOperationsTable[GLCOLOR3F] = (glOp)o_glColor3f;
}
```

Another example of differentiating between SVN instances is the Stereo Overload set. By using this set, you can display a regular (non-stereo) OpenGL application in stereo vision (active or passive). This is accomplished as follows:

- The application runs on the client node and displays on two SVN rendering servers, one for left eye and one for right eye.
- Each server uses the Stereo Overload set to output either a left eye or a right eye view for stereo viewing.
- SVN delivers the output from the servers into a stereo projection system.
- The SVN client can have different behaviors:
  - It can be used instead of one of the servers displaying the left eye or right eye image.
  - It can bypass the overload set and display the standard (non-stereo) application output.

In order to obtain left and right eye views through the Stereo Overload set, SVN applies the following methods:

- The overload set on each SVN instance obtains its own display name by using a handle to the function `DVGetDisplayName()`. You can then specify whether the output should be left eye, right eye or non-stereo for each display.
- In order to produce a left or right eye image, a stereo transformation needs to be added to the OpenGL projection matrix. Therefore, the Stereo Overload set overloads all OpenGL calls that affect the projection matrix and makes sure that the stereo transformation is always applied.

When you are using the Stereo Overload set, you follow the steps that are used for a simple overload, plus some additional steps.

- These are the basic steps that are required for all overload sets:
  1. Compile the overload code into a named, shared object (for example, *invert_colors.so*).
     - You can use the sample *Makefile* provided with the SVN code.
  2. Create a file named *overload_files.txt* that contains a list of overload modules for SVN to use.
     - The `overload_files.txt` file has one line for each overload module.
     - The format for each entry has the form */...correct_path.../invert_colors.so*.
  3. Set the `$SVN_OVERLOAD_FILE` environment variable to point to `overload_files.txt`.
     - This variable might require the full path to the text file.
- In addition to the basic steps, the Stereo Overload set also requires the following steps:
  1. Set up a stereo configuration file.
     - This file should contain one entry for each display that will be used for stereo vision.
     - The entry uses the form:

       server_name[:d.s] {STEREO_LEFT|STEREO_RIGHT}

Note: You must use the exact server and display names that are used in the `$SVN_DISPLAY` configuration file.

2. If the *client* is to be used for stereo vision, add an entry using the form:
   client {*STEREO_LEFT*|*STEREO_RIGHT*}
3. Set the environment variable `$SVN_STEREO_FILE` to point to the stereo configuration file.
4. Run the desired OpenGL application through SVN and feed the left eye and right eye displays to stereo projection hardware.

Notes:

1. If an entry corresponding to any of the SVN displays is missing from the stereo configuration file, the SVN instance running that display simply bypasses the Stereo Overload set and produces standard (non-stereo) output.
2. The steps listed in this section are specific to this particular implementation of the Stereo Overload set. They are not typically required by SVN.
3. Other developers might implement the same functionality through overloads using other methods to determine whether to output left eye, right eye or a standard image on a particular display.

Using multiple overload sets

The previous example described an overload *list file* with one entry that pointed to the associated overload shared object. The overload list file can also have more than one entry. In such a case, SVN loads and uses all the provided overload sets. For example, you could use both sets developed in the previous examples. If you did so, the resulting display would have stereo images with inverted colors. To accomplish this, you would create an overload list file that would look like this:

```
/...correct_path.../stereo_overload.so
/...correct_path.../invert_colors.so
```

Notes:

1. You must update the `$SVN_OVERLOAD_FILE` environment variable to point to the overload list file.
2. You could create a single overload with multiple functions. However, if you use an overload list file that points to multiple overloads, you can define specific end results by combining different overloads in an overload list file.
3. Although you can achieve positive results with multiple overloads, you can also create system conflicts if you are not careful with how you combine overloads in the overload file list. Review the overload list file and the associated overloads to confirm that you will achieve the desired results.

The following example illustrates what happens when two overload sets try to overload the same OpenGL call. In this case, overload_set_1 and overload_set_2 both overload `glColor3f(..)`. This is done using the following code fragments:

overload_set_1

```
void Overload(glOp *glOperationsTable, glOp *glSystemOperationsTable)
{
  s_glColor3f = (void(*)(float,float,float)) glOperationsTable [GLCOLOR3F];
  glOperationsTable[GLCOLOR3F] = (glOp)o_1_glColor3f;
}
static void
o_1_glColor3f(float r, float g, float b)
{
   ...
   (*s_glColor3f)(...)
   ...
}
```

overload_set_2

```
void Overload(glOp *glOperationsTable, glOp *glSystemOperationsTable)
{
  s_glColor3f = (void(*)(float,float,float)) glOperationsTable [GLCOLOR3F];
  glOperationsTable[GLCOLOR3F] = (glOp)o_2_glColor3f;
}
static void
o_2_glColor3f(float r, float g, float b)
{
   ...
   (*s_glColor3f)(...)
   ...
}
```

Note: For this example, the overload list file contains two entries:

```
/...correct_path.../overload_set_1.so
/...correct_path.../overload_set_2.so
```

When SVN loads the overload sets, these actions occur:

1. SVN calls the `Overload(...)` function of `overload_set_1` and passes it a pointer to the `glOperationsTable`.
2. `overload_set_1` stores the handle to `glColor3f(...)` that it finds in `glOperationsTable`.
3. `overload_set_1` replaces the handle to `glColor3f(...)` with a handle to its own `o_1_glColor3f(...)`.
4. SVN calls the `Overload(...)` function of `overload_set_2` and passes it a pointer to the `glOperationsTable`.
5. `overload_set_2` stores the handle it finds in `glOperationsTable`. However, that handle was replaced by `overload_set_1` and it is now a handle to `o_1_glColor3f(...)`.
   - As a result, whenever `overload_set_2` is using the handle it found in `glOperationsTable`, `overload_set_2` is actually calling `o_1_glColor3f(...)`.
6. `overload_set_2` replaces the handle from `glOperationsTable` with a handle to its own `o_2_glColor3f(...)`.
7. Because the handle in `glOperationsTable` was replaced twice, SVN calls are diverted to `o_2_glColor3f`.

Figure 3 on page 19 shows the effect multiple overload sets have when they make application calls to the same `glColor3f(...)` handle. The illustrated pipeline might be broken if, for example, `o_2_glColor3f(...)` would not attempt to use the handle it receives from SVN and stores in `s_glColor3f(...)`. In general, when multiple sets overload the same OpenGL call, that call triggers the execution of the overload function from the last set of overloads contained in the overload list. Then, depending on the implementation of each overload, the call might propagate to the overload version of the same function defined in a set that is higher in the list.

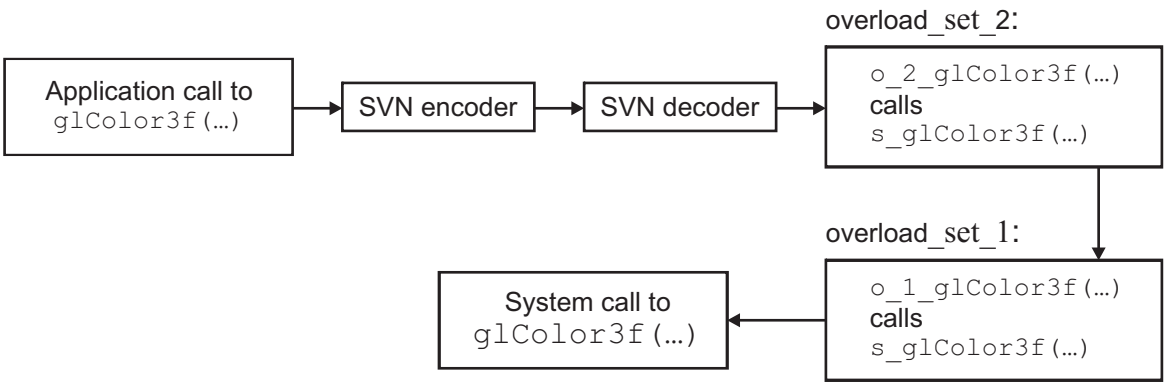

Figure 3. Multiple overload sets

Chapter 3. Remote Visual Networking

Remote Visual Networking (RVN) allows remote, collaborative access to OpenGL-based 3-D graphics applications. RVN intercepts data within the rendering pipeline, uses local resources for rendering, and then redirects the finished images to the remote displays (end stations). It permits the use of applications despite higher-latency and lower-bandwidth networks. Because it uses local rendering resources, RVN allows machines with limited 3-D rendering capability to display highly interactive 3-D applications. RVN can be configured to send images to multiple destinations, facilitating the simultaneous interaction of several collaborators.

Central to every RVN session is the notion of an RVN *sender* and an RVN *receiver*. A sender, also known as an application host or application server, is the host on which the graphics application runs. An RVN receiver or end station consists of a computer that is reachable from the sender through TCP/IP and a display.

Supported operating systems

RVN supports application servers and end stations under both Linux and Microsoft Windows. The application server and end stations do not need to run under the same operating system in order to participate in the same RVN session. For more information about operating systems and other software requirements for RVN, refer to “RVN prerequisites” on page 55.

Each RVN session involves exactly one sender and up to eight receivers. Both senders and receivers involve:

- Processing hardware
- Application (or RVN) software
- Display hardware

Note: The terms sender and receiver refer to some combination of the hardware, software and display on the application or end station side, respectively, depending on context. For additional information about sender and receiver, refer to “RVN sessions” on page 22.

To intercept data along the rendering pipeline on the sender, RVN uses an OpenGL intercept library that the application loads in place of the conventional OpenGL library. The RVN OpenGL library in turn loads the conventional OpenGL library. Graphics applications are invoked by the RVN launcher, which sets up the environment to load the RVN OpenGL library before the system OpenGL library. The execution of an RVN session is fully configurable by environment variable settings, options set through the RVN launcher, and a runtime dashboard that allows you to change RVN settings throughout application execution.

For maximum flexibility and data transfer rates, you should install RVN on the receiving end station as well as on the application server. Doing so allows you to take advantage of the accelerated graphics support in RVN, which provides fast transport of pixel data by using highly optimized image-compression methods well suited to most 3-D application domains.

In an RVN session, the application user interface is transferred to the end station through the remote desktopping support provided by VNC, or by the conventional `X11 DISPLAY` environment variable mechanism (Linux application servers only).

Required version of VNC

VNC Visualization Edition is the only supported VNC product. Using a different VNC product (whether from RealVNC or another company) on the RVN sender, the RVN receiver, or both, is not supported. For more information about the required version of VNC, see “RVN prerequisites” on page 55.

When the RVN sender gets the application commands to display a rendered image in 3-D, the RVN sender obtains the image, compresses it, and sends it over the network to the RVN receiver. The receiver decompresses the image and puts it into one of two locations:

- If you are using VNC, the image fills the application window in the `vncviewer`.
- If you are using `X11 DISPLAY` export, the image is displayed in the application window.

RVN typically performs 3-D rendering in off-screen memory. Therefore, 3-D images are usually not displayed on the application server. A blank graphics window is usually displayed in place of the 2-D interface for the application. If you are using VNC, you can configure RVN to write to the application server display by either setting the environment variable `RVN_HOST_SHOW_PIXELS=1` or by selecting the **Show on server** option in the RVN dashboard.

RVN sessions

An RVN session consists of the interaction among several hardware and software components. Under RVN, a 3-D visualization runs on an application server and is displayed on up to eight end stations. The end stations and application server must be connected by a local or wide area network that supports IP communication. Pixel data (and X events, in the case of Linux) each require a transport mechanism between the application server and the end stations. For each RVN session:

- Where the 3-D rendering occurs depends on whether RVN is running with SVN.
  - If you are running an RVN session without SVN, all 3-D rendering takes place on the application server.
  - If you are running an SVN-RVN interoperability session, 3-D rendering moves off of the application server and onto a separate rendering server that processes the 3-D image.
- RVN manages the transport of the image pixel data across the network to the remote end stations.

With RVN, each end station receives pixel data from the application server and sends user events (such as mouse or keyboard actions) to the server. Depending on the software components and transport mechanism involved, you can have up to eight end stations in a given RVN session. The application server is responsible for pushing updates to its application window (in the form of pixel data) across the transport to each connected end station. Each end station is responsible for:

- Displaying one or more application windows
- Interpreting user events on those application windows
- Sending user events back to the application server for processing

Notes:

1. RVN does not handle any of the event processing or transport back to the application server. It relies on either VNC transport or the X remote `DISPLAY` mechanism (Linux application servers only) for that function.
2. When the X remote `DISPLAY` mechanism is used by an application server running under Linux, the end station RVN code primarily runs within an application called `endstation` (for Linux) or `endstation.exe` (for Windows). That application is invoked by a front-end `rvn_receiver` script (for Linux) or `rvn_receiver.bat` file (for Windows) that sets up the internal environment and then invokes `endstation`.

References to the RVN transport layer or accelerated graphics pertain to the mechanism by which pixel data are sent from the application server to the end stations. It is assumed that the transport of desktop pixels (other than within the application window) to the end stations and the input events from the end stations get transported by other means. For sessions between a single application server and multiple end stations, VNC is required. For sessions between a single application server and a single end station, either VNC or X (through the remote `DISPLAY` mechanism) can be used for this transport.

Connecting to an RVN session

A convenient comparison for a connection to an RVN session is a business conference call. Like a conference call, each RVN session requires:

- A chairman (the graphics application running with RVN on the application server and specified in the RVN launcher or with the **rvn_sender** command)
- One or more participants (the `vncviewer` or `rvn_receiver` process running on the remote machines, or end stations)
- An operator (the `rvn_coordinator` process running on the application server)
  - The coordinator process arranges for the applications and participants to connect.
  - Typically, the coordinator runs automatically, behind the scenes, requiring no user involvement.

How participants are authorized to connect to an RVN session depends on whether VNC or X11 export mode is being used.

Authentication when VNC is used

If VNC Visualization Edition is being used on the RVN sender and the RVN receivers, authorization to connect to the RVN session is automatically handled by VNC password authentication. There is no need for the sender and receiver to provide additional information in order to join the RVN session.

Authentication when X11 DISPLAY export is used

RVN sessions using the X remote `DISPLAY` mechanism require the use of RVN session authentication. In addition to the hardware and software requirements listed above, each end station requires three pieces of information to "dial into" and join a session:

- The host name of the application server
- The conference ID string
- The conference access key

Note: For security purposes, you must convey the host name, ID string, and access key to each participant. Each participant must have this information before attempting to log in to the session.

Using the supplied host name, each participant attempts to connect to the RVN session. As part of establishing the connection, the coordinator checks the conference ID string. If the ID strings match, the coordinator directs the RVN sender and the RVN receiver to meet on a specific port and continue negotiating for a direct connection between the sender and receiver. During this negotiation, the participant must supply the conference access key. Having the correct access key authenticates the end station and enables it to join the session (also called a *conference*).

Configurations for RVN sessions

RVN sessions use one of the following configurations:

- Linux application servers only: RVN with X11 DISPLAY export (refer to Figure 4)
- RVN with VNC (refer to Figure 5 on page 25)
- RVN with VNC and multiple end stations, or collaborators (refer to Figure 6 on page 25)
- Linux application servers only: RVN with VNC and desktop isolation (refer to Figure 7 on page 26)

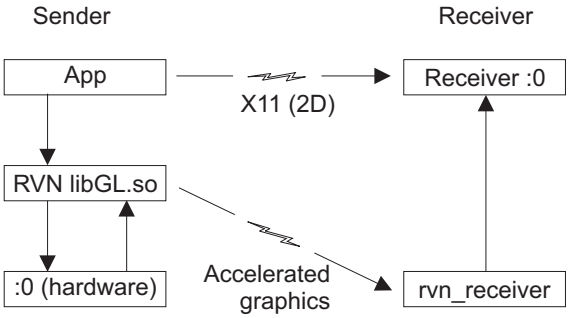

Figure 4. RVN with X11 DISPLAY export (Linux application servers only)

Figure 4 shows RVN using the `X11 DISPLAY` export mechanism. This configuration can be used only when the application server is running under Linux, and requires RVN code on the end station. Two-dimensional user data is transported by the `X11` remote mechanism. Three-dimensional data is transported through the accelerated graphics function of RVN.

Note: This configuration permits only a single end station. The `X11 DISPLAY` export mechanism is best used when the application server and end station are on the same local area network (LAN).

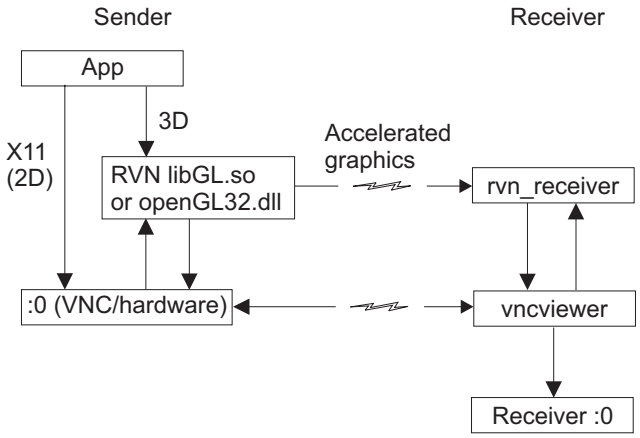

Figure 5. RVN with VNC

Figure 5 illustrates how you can use VNC for remote desktopping. The RVN sender writes 3-D data to the local application windows in a manner that triggers VNC to send updates to the end station. In this mode, VNC provides both the 2-D and pixel transport mechanisms.

Note: VNC also allows multiple end stations to operate in a mode similar to that shown in Figure 5.

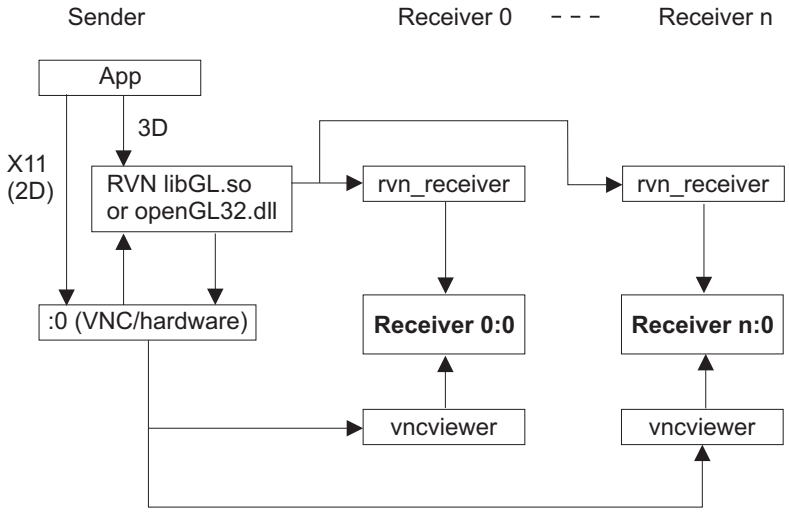

Figure 6. RVN with VNC and multiple end stations

Figure 6 illustrates the collaborative (multiple end station) versions of VNC and the accelerated graphics function of RVN. Note that the setup is similar to the VNC base configuration shown in Figure 5. However, in the collaborative version there are multiple receivers and all receivers behave similarly. In addition, 2-D data transport to multiple end stations is made possible by VNC, and pixel transport is enabled by the RVN receiver software running on each end station. Information about each receiver is passed to the RVN sender, causing RVN to send data to multiple receivers.

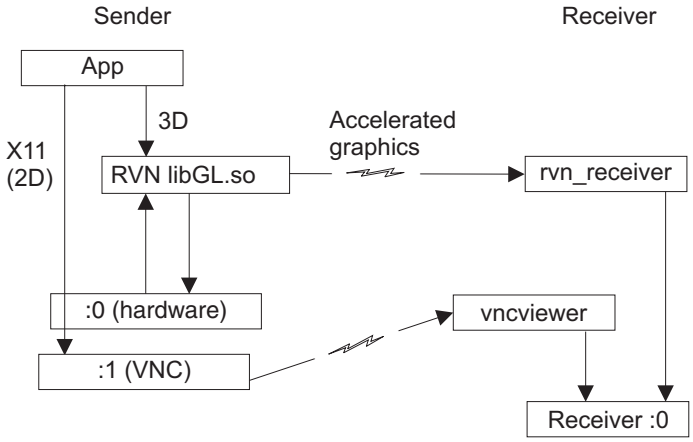

Figure 7. RVN with VNC and desktop isolation (Linux application servers only)

Figure 7 illustrates the ability to exploit VNC desktop isolation mode using RVN. In this mode, VNC can create multiple `X11` desktops on the application server for remote viewing. In addition, RVN can be configured to operate on an application running on one of the additional desktops.

Note: Although the application server must be running under Linux, the end stations can run under Linux or Windows.

User interface for working with RVN

The following elements form the core user interface for an RVN session:

- The RVN application launcher
- The RVN dashboard
- The **rvn_sender** command (Linux) or **rvn_sender.bat** command (Windows)
- The **rvn_receiver** command (Linux) or **rvn_receiver.bat** command (Windows)
- The **rvn_viewer** command (Linux only)
- The RVN coordinator (Linux only)

Table 1 on page 27 summarizes how the RVN session participants use the elements of the interface. Typically, the receiver initiates the session by starting the VNC viewer. However, either the sender or the receiver can start the session. For detailed examples of the sequence in which these elements operate in an RVN session, refer to “Running RVN” on page 40.

Table 1. Interactions with the user interface for RVN

Step	Notes
The **receiver** (on the end station) connects to the session.	Any of these methods can be used: • For sessions that use VNC, this is done through normal VNC password authentication after launching the VNC viewer with one of the following commands: – For Linux end stations, use the **rvn_viewer** command. – For Windows end stations, use the **vncviewer** command. • For sessions that use X11 export mode, this is done by running the **rvn_receiver** command (Linux) or **rvn_receiver.bat** command (Windows).
The **sender** (on the application server) launches the application and connects to the session.	Any of these methods can be used: • Run the **RVN application launcher**, using one of the following options: – Run the **rvn_sender** command (Linux) or **rvn_sender.bat** command (Windows) without arguments. – Windows only: Use the **RVN Sender** shortcut from the Windows **Start** menu (**Start > All Programs > IBM > IBM Deep Computing Visualization > RVN Sender**). • Bypass the launcher, using one of the following options: – Run the **rvn_sender** command (Linux) or **rvn_sender.bat** command (Windows) and specify the application with arguments. – Windows only: Copy the RVN OpenGL library (openGL32.dll) to the the same directory as the process that loads openGL32.dll, then launch the application directly.
The **RVN coordinator** handles the initial setup for connections between the sender and receiver.	
A graphical user interface (**the RVN dashboard**) can be used on the application server to configure RVN parameters while the application is running in an RVN session.	

You can use the RVN launcher and dashboard in various combinations:

- **Launcher and dashboard (recommended approach):** Run the launcher using one of the methods listed in Table 1. Then click **Launch** in the launcher window. This automatically launches the application and opens the dashboard.
- **Launcher without dashboard:** Set the `RVN_SUPPRESS_DASHBOARD` environment variable to 1. Then run the launcher using one of the methods listed in Table 1. No dashboard will be available during the session, and you will not be able to change settings at run time.

- **Dashboard without launcher:** This option is the same as has been available in the past under Linux. Under Linux or Windows, set the desired environment variables. Then start RVN and bypass the launcher, using one of the methods listed in Table 1 on page 27.
- **No launcher and no dashboard:** This option is the same as has been available in the past under Linux. Under Linux or Windows, set the desired environment variables (including `RVN_SUPPRESS_DASHBOARD` to disable the dashboard). Then start RVN and bypass the launcher, using one of the methods listed in Table 1 on page 27.

For more information about using these elements of the user interface for RVN, see the following sections:

The RVN application launcher

The RVN application launcher sets up certain environment variables for running the graphics application under RVN, ensures access to the system OpenGL library and the RVN OpenGL library, and identifies the application to be launched. After completing this setup, it calls the RVN dashboard so that you can configure RVN parameters while the application is running in an RVN session.

These are the environment variables that the RVN launcher sets:

- `RVN_USE_VNC` (Linux only; Windows always uses VNC)
- `RVN_EXTERNAL_TRANSPORT`
- `DISPLAY` (VNC mode only)

Before starting the RVN launcher, set any necessary additional environment variables needed for this RVN session, including the path to the system OpenGL library (unless that path was set during installation). For a complete list, refer to "Environment variables for rvn_sender" on page 88.

Note: For X11 mode sessions, you must set the `DISPLAY` environment variable to the IP address of the receiver before you start the launcher: `export DISPLAY=receiver:0.0`. You must also set `RVN_DASHBOARD_DISPLAY=:0.0`.

Then start the RVN launcher. Figure 8 on page 29 and Figure 9 on page 29 show the main window for the RVN launcher on Windows or Linux application servers.

Note: You can start the launcher by either of these methods:

- Run the **rvn_sender** or **rvn_sender.bat** command with no arguments. For more information, refer to "The rvn_sender command" on page 36.
- Windows only: Use the **RVN Sender** shortcut from the Windows **Start** menu (**Start > All Programs > IBM > IBM Deep Computing Visualization > RVN Sender**).

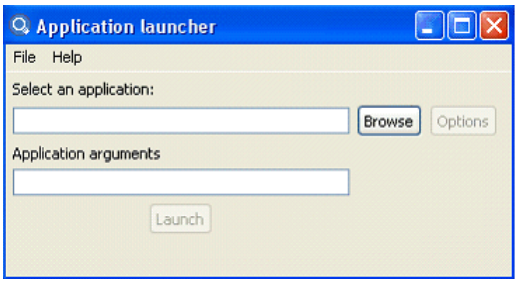

Figure 8. RVN application launcher: Main window (Windows)

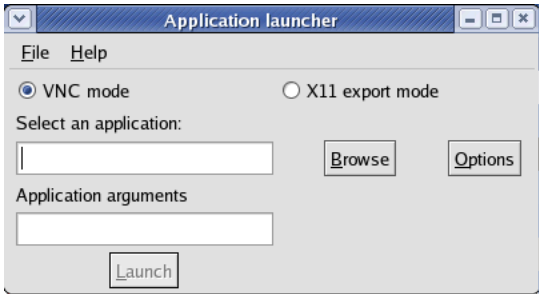

Figure 9. RVN application launcher: Main window (Linux)

These are the options that you must specify in the main window for the RVN launcher:

Mode options (Linux only)

VNC mode
Tells RVN to look for a VNC window when writing application images.

Note: Using this option disables the accelerated graphics function of RVN and sets the following environment variables:

- `RVN_USE_VNC=1`
- `RVN_EXTERNAL_TRANSPORT=0`
- `DISPLAY=:0`

X11 export mode
Tells RVN to use the X server that is running on the receiver.

Note: Using this option sets the following environment variables:

- `RVN_USE_VNC=0`
- `RVN_EXTERNAL_TRANSPORT=0`

Select an application
Specifies the path to the graphics application. Either enter the full path, or click **Browse** and navigate to the application. (**Launch** is not available if you have not provided this information.)

Application arguments
Specifies any additional arguments needed by the application.

If you need to provide any of the following information, click **Options** to enter the data in a window for additional options, as shown in Figure 10.

- The paths for RVN_HOME and the system OpenGL library. If these are not supplied, defaults will be used.
- Windows only: A user-specified directory to use as the destination of the RVN OpenGL library (openGL32.dll). If this is not supplied, the file will be copied to the application directory.
- A user-specified port for socket communication with the application. If this is not supplied, the port will be selected automatically.

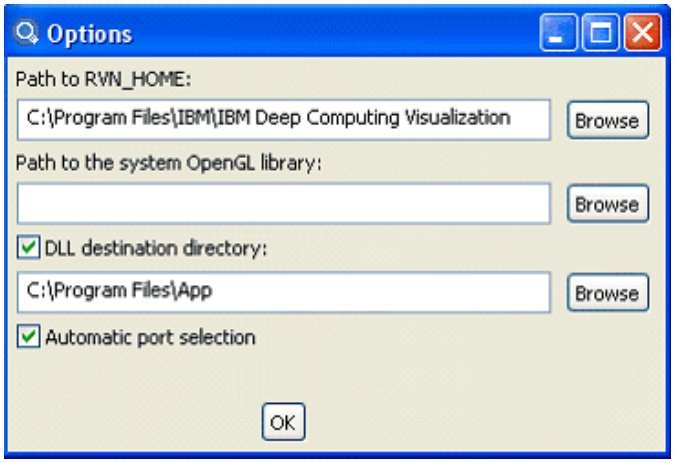

Figure 10. RVN application launcher: Additional options

These are the additional options that you can specify in the RVN launcher:

Path to RVN_HOME
Specifies the location of RVN OpenGL library, which is copied to the application directory or (for Windows only) to the location specified in the **DLL destination directory** field. This information is required, and might have been preset during installation. If you did not use an environment variable to set the location, provide that information now. Either type the full path, or click **Browse** and navigate to the file.

Path to the system OpenGL library
Specifies the location of the system OpenGL library, which is used along with the RVN OpenGL library. This information is required, and might have been preset during installation. If you did not use an environment variable to set the location, provide that information now. Either type the full path, or click **Browse** and navigate to the file.

DLL destination directory (Windows only)
Indicates whether the openGL32.dll file should be copied to a user-specified directory instead of to the application directory (the directory from which the application was started). Some applications load executable libraries from other directories or start other underlying processes from other directories that in turn load openGL32.dll.

- To copy the openGL32.dll file to the application directory, leave this option not selected. This is the default.

- To copy the openGL32.dll file to a user-specified directory, select this option, then provide the path for the directory. Either type the full path, or click **Browse** and navigate to the directory.

Note: Normally, you should not need to separately specify the DLL destination directory unless you know that the openGL32.dll file must be in a specific location other than the application directory. However, if you do not specify the DLL destination directory and openGL32.dll is not subsequently loaded, you need to specify the correct location. To determine that location, search for all directories that contain .exe and .dll files that were installed when the application was installed (excepting the system directory). This might reveal directories that could be appropriate for the openGL32.dll file. Possible locations can include directories and subdirectories in the \Program Files\Common Files\ directory for the application, if it has one, as well as the root directory and subdirectories for the application itself, normally in the Program Files directory. Try each of the locations until one of them works.

Automatic port selection
Indicates whether the port for socket communication with the application should be automatically selected. To specify a specific port, clear **Automatic port selection**, then select the port that you want to use for communication.

After you provide the required information and click **OK** in the Options window, you return to the main window for the RVN launcher and click **Launch**. This sets the environment variables for this RVN session, indicates the application to be invoked (along with the specified arguments), and launches the RVN dashboard.

Note: If you selected X11 export mode, you are prompted to provide the following information in the "Conference coordination" window:

Conference ID
Supplies the conference ID string for the RVN session.

Notes:

1. This information must be conveyed to each participant before the start of the conference.
2. Using this option overrides the setting of the `RVN_CONFERENCE_ID` environment variable.

Conference key
Supplies the conference access key.

Notes:

1. This information must be conveyed to each participant before the start of the conference.
2. Using this option overrides the setting of the `RVN_CONFERENCE_KEY` environment variable.

The RVN dashboard

The RVN dashboard allows you to control certain runtime options while the application is running in an RVN session. Figure 11 on page 32, Figure 12 on page 32, and Figure 13 on page 32 show the main window for the RVN dashboard for VNC mode sessions and X11 mode sessions.

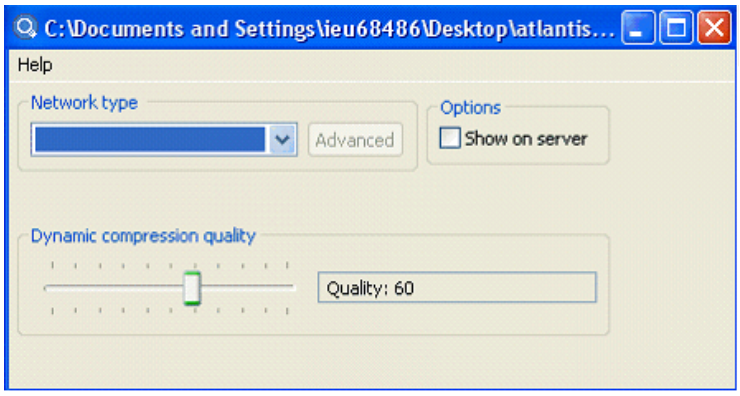

Figure 11. RVN dashboard: Main window (Windows)

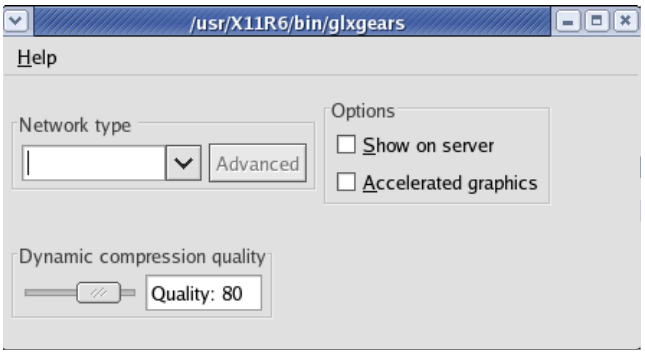

Figure 12. RVN dashboard: Main window (Linux, VNC mode)

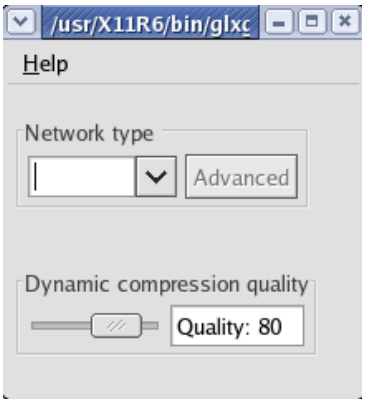

Figure 13. RVN dashboard: Main window (Linux, X11 mode)

The RVN dashboard provides the following options:

Network type

Enables you to use predefined settings for environment variables, based on network speed, or to specify your own values for these settings.

- To use predefined settings based on your network speed, select one of the network types. Table 2 on page 33 lists the options that are used by

default for each type of network. All of the settings apply to all of the connected end stations. To override any of these defaults, select a network type, then click **Advanced**. Then specify your settings in the "Advanced options" window, shown in Figure 14 on page 34.

Table 2. Default options for the RVN dashboard, based on network type

Network type	Default options
High performance network	Accelerated graphics: selected (Linux only) Interactive mode: selected Static image boost: not selected Dynamic compression quality: 80 Low bandwidth - automatic
Medium performance network	Accelerated graphics: selected (Linux only) Interactive mode: selected Static image boost: not selected Dynamic compression quality: 50 Low bandwidth - automatic
Low performance network	Accelerated graphics: selected (Linux only) Interactive mode: selected Static image boost: not selected Low bandwidth - automatic Display partial frames: selected

- To specify your own settings for these options, select **User options**, then click **Advanced**. Then specify your settings in the "Advanced options" window, shown in Figure 14 on page 34.

Show on server

Enables or disables displaying pixels on the local screen of the application server in addition to sending it to the RVN end station. If this option is selected and the accelerated graphics function of RVN is being used, RVN updates the application server screen directly. The initial setting of this option is determined by the environment variable `RVN_HOST_SHOW_PIXELS`.

Note: This option is available only in VNC mode.

Accelerated graphics (Linux only)

When this option is selected, the RVN pixel backchannel is enabled, and OpenGL graphics output is transferred to the viewer much more quickly. RVN will write data to a local window and not send it out across an accelerated graphics link.

Note: This option is available only in VNC mode.

Dynamic compression quality

Controls the balance between image compression and image quality when the image is changing. The range is a percent from 1 to 100. Choosing a high quality value (a large number) improves the visual quality of the dynamic images, but requires more data to be sent for each image, which can reduce responsiveness or lower the frame rate. If you select 100% compression, lossless compression will be used.

Note: This option can also be specified in the "Advanced options" window.

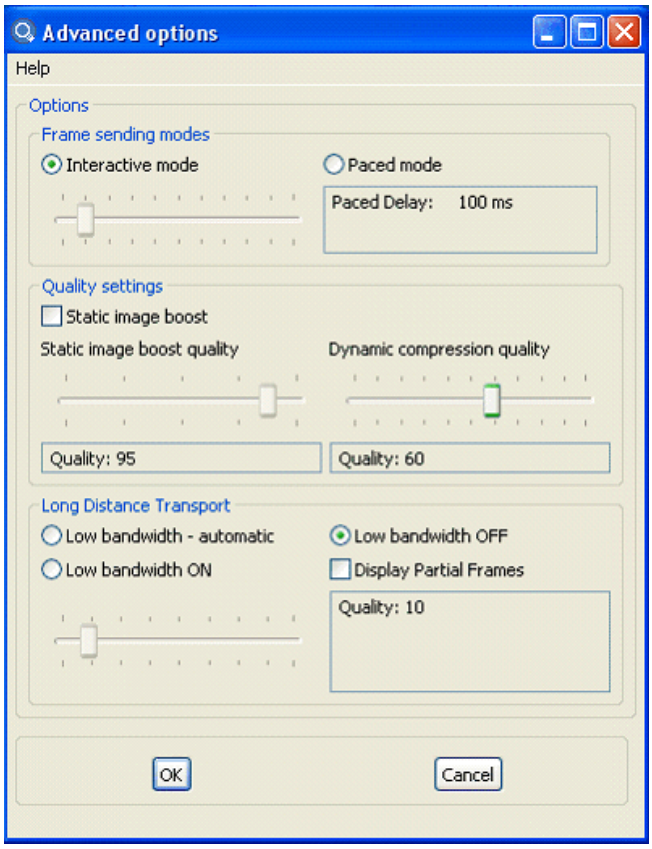

Figure 14. RVN dashboard: Advanced options

If you clicked **Advanced** in the main window of the RVN dashboard, the following options are available in the "Advanced options" window, as shown in Figure 14:

- **Frame sending modes**

  Interactive mode

  Attempts to allow the application to run as fast as possible (with the least RVN delay). It always transmits the most recently generated frame to the end-station display. In doing so, it might drop stale images before trying to compress and send them, in favor of sending newer images. Use this mode if you are most interested in always viewing the most recent frame, and not concerned with seeing every generated frame.

  Note: This option is the default.

  Paced mode

  Sends every frame that the application generates and, in so doing, can often significantly slow the application. Paced mode allows the users at the end stations to see all of the rendered data. It also addresses the issue of frame-rate jitter, where image frames are not displayed at regular intervals. This variability in the frame-to-frame interval is typically the result of network jitter or a variability in computation loads on the sender or viewer. To "smooth out" the jitter, paced mode introduces interval delays, causing the frames to be displayed at regular intervals. This creates a regular paced viewing experience.

After selecting **Paced mode**, increase the pacing period slider until the frame rate becomes sufficiently smooth and pleasing to the eye. (The slider determines the length of the delay, from 0 to 1000 ms.)

- **Quality settings**

  Static image boost
  Sends a higher quality image when the image is not being rapidly updated by the application. When this option is selected, RVN rapidly transfers lower-quality images when the image is changing, then updates the image with a high-quality update, thereby enhancing the quality of the static image. By default, this option is not selected.

  Note: In some circumstances, rapid flickering occurs when **Static image boost** is selected. To avoid this condition, do not select this option.

  Static image boost quality
  Controls the balance between image compression and image quality when the image is not changing. The range is a percent from the current dynamic compression quality to 100. If you select 100% compression, lossless compression will be used.

  Note: This option is available only if the **Static image boost** option is selected.

  Dynamic compression quality
  Controls the balance between image compression and image quality when the image is changing. The range is a percent from 1 to 100. Choosing a high quality value (a large number) improves the visual quality of the dynamic images, but requires more data to be sent for each image, which can reduce responsiveness or lower the frame rate. If you select 100% compression, lossless compression will be used.

- **Long Distance Transport**

  The setting of this option determines how pixels are sent when latency is high. Using the low bandwidth option provides a more responsive view when **Interactive mode** is selected and a sequence of dynamic frames is being sent. Select one of the following options:

  Low bandwidth - automatic
  Allows RVN to automatically determine whether to use the low bandwidth option, based on throughput, latency, and packet-loss threshold.

  Low bandwidth ON
  Indicates that the low bandwidth option will be used.

  Low bandwidth OFF
  Indicates that the low bandwidth option will not be used.

  If you select **Low bandwidth - automatic** or **Low bandwidth ON**, the following options are available:

  Display Partial Frames
  Indicates whether to display partially updated frames. Select this option to see a "frameless" display in the event of network losses, and to use old data to fill occasional gaps in the image when some data segments are lost.

  Image quality slider
  Adjusts the image quality when the low bandwidth option is selected manually or automatically. Because fewer frames are transmitted when a

low bandwidth is used, you might want to reduce the quality of the image to increase the compression and, therefore, speed. Low values mean high compression and poor image quality; high values mean little compression and high image quality.

The rvn_sender command

The **rvn_sender** command (Linux) or **rvn_sender.bat** command (Windows) starts the graphics application on the sender. Similar to **rvn_receiver**, **rvn_sender** sets up the internal RVN environment before invoking the application.

Note: Unless indicated otherwise, all references to the **rvn_sender** command also apply to the **rvn_sender.bat** command.

You can use the **rvn_sender** command to invoke the RVN application launcher or to bypass the launcher. For details, see “User interface for working with RVN” on page 26.

- To invoke the RVN application launcher, omit the application name and arguments from the **rvn_sender** command. This approach enables you to continue using the **rvn_sender** command in existing scripts, and to use the new launcher to set environment variables and invoke the application. For more information about the launcher, refer to “The RVN application launcher” on page 28.
- To bypass the RVN application launcher, include the application name and arguments on the **rvn_sender** command, as in previous DCV releases.

  Notes:

  1. With this approach, if RVN is not installed in the default directories, make sure to specify the location on the `RVN_HOME` and `DCV_ROOT` environment variables.
  2. Windows only: This approach sets environment variables and copies the openGL32.dll library to the application directory, just as the launcher does.

This is the syntax of the **rvn_sender** command:

```
rvn_sender [-v] [-conf <confID>] [-key <confkey>] [-port <portnum>] <appname> [<app-args>]
```

For example, suppose you want to start an RVN session with the following parameters:

- Using X11 DISPLAY export mode, not VNC
- With the conference name *myconf*
- Having the access key *mykey*
- Running the program *my_prog*
- Using the default port for socket communication with the application
- Bypassing the RVN application launcher

This is what you would specify on the **rvn_sender** command:

```
rvn_sender -conf myconf -key mykey my_prog
```

Note: When VNC Visualization Edition is used on the sender and the receivers, there is no need to specify the conference name and access key on the **rvn_sender** command. This information is automatically generated with VNC authentication and is not exposed.

The **rvn_sender** command analyzes the options that were specified on the command, as well as the settings of related environment variables.

- For a description of the command-line options, see "Command-line options for rvn_sender."
- For a description of the related environment variables, see "Environment variables for rvn_sender" on page 88.
- For examples of using the **rvn_sender** command and related environment variables, see "Running RVN" on page 40.

Command-line options for rvn_sender

This section defines the options that can be specified on the **rvn_sender** command.

-v Tells RVN to look for a VNC window when writing application images.

Notes:

1. Linux only: If this option is not specified, X11 export mode is used.
2. Windows only: This option is required.
3. Using this option disables the accelerated graphics function of RVN and sets the value of the environment variables `RVN_USE_VNC` and `RVN_EXTERNAL_TRANSPORT` to 1.

-conf *confID*

Supplies the conference ID string for the RVN session.

Notes:

1. This information is needed only for X11 export mode under Linux. When VNC Visualization Edition is used on the sender and receivers, this information is automatically generated during VNC authentication.
2. This information must be conveyed to each participant before the start of the conference.
3. Using this option overrides the setting of the `RVN_CONFERENCE_ID` environment variable.

-key *confkey*

Supplies the conference access key.

Notes:

1. This information is needed only for X11 export mode under Linux. When VNC Visualization Edition is used on the sender and receivers, this information is automatically generated during VNC authentication.
2. This information must be conveyed to each participant before the start of the conference.
3. Using this option overrides the setting of the `RVN_CONFERENCE_KEY` environment variable.

-port *portnum*

Specifies a port for socket communication with the application, instead of letting the port be selected automatically.

The rvn_receiver command (X11 DISPLAY export sessions only)

The **rvn_receiver** command is used in X11 DISPLAY export configurations where RVN code runs on the end stations. The primary purpose of this command is to invoke the `endstation` executable file. Before invoking `endstation`, **rvn_receiver** sets up some internal environment variables, such as the required `LD_LIBRARY_PATH` value on Linux.

Notes:

1. The `rvn_receiver` and `endstation` code uses the default port 7200.
2. If the default port is used by another application, or, if the application server is using a different port number, you can override the default value by setting the `RVN_COORDINATOR_PORT` environment variable to an available port number.
3. If you are invoking **rvn_receiver** from an end station that is running under Microsoft Windows, you must do so from a command prompt. For more information, refer to "Microsoft Windows considerations for rvn_receiver."

This is the syntax of the **rvn_receiver** command for Linux:

```
rvn_receiver [host] <confID> <confkey> <nickname>
```

The **rvn_receiver** command analyzes the options that were specified on the command, as well as the settings of related environment variables.

- For a description of the command-line options, see "Command-line options for rvn_receiver."
- For information about running **rvn_receiver** in a Windows environment, see "Microsoft Windows considerations for rvn_receiver."
- For a description of the related environment variables, see "Environment variables for rvn_receiver" on page 92.
- For examples of using the **rvn_receiver** command and related environment variables, see "Running RVN" on page 40.

Command-line options for rvn_receiver

This section defines the options that can be specified on the **rvn_receiver** command.

host The name of the host where `rvn_sender` is invoked.

confID A string identifying the RVN conference session you are connecting to. This information must be distributed to all conference participants before the conference starts, using the method of your choice. The person responsible for providing this information is the conference initiator (the person invoking **rvn_sender**).

confkey
The security password required to access the RVN conference session. This information must be distributed to all conference participants before the conference starts, using the method of your choice.

nickname
A string that provides an on-screen identity for each conference participant.

Microsoft Windows considerations for rvn_receiver

To participate in an X11 export session from a Windows end station, run the **rvn_receiver.bat** command in a command prompt window to invoke **endstation.exe**. To open a command prompt window, click **Start > All Programs > Accessories > Command Prompt**. Update the path so that **rvn_receiver.bat** can be typed directly at the command prompt. If the path is not set up correctly, run **rvn_receiver.bat** from the default location `C:\Program Files\IBM\DCV Endstation\` or from the directory where it was installed.

Here is the syntax:

```
rvn_receiver[.bat] <host> <confID> <confkey> <nickname>
```

rvn_receiver.bat launches **endstation.exe** for connecting to an RVN session, and it matches the syntax of the Linux command line for the `rvn_receiver` shell script. For details about this syntax, refer to “Command-line options for rvn_receiver” on page 38.

To run **endstation.exe**, **rvn_receiver.bat** works with an X server running on the end station. For information about the programs that are supported, refer to “RVN prerequisites” on page 55.

Note: A Win32 X server is necessary to operate without VNC. If you are not using VNC, you must configure the `DISPLAY` environment variable for the application to use X11 export mode.

The rvn_viewer command (Linux end stations in VNC mode only)

The **rvn_viewer** command starts the VNC viewer on a Linux end station.

Note: On a Windows end station, use the **vncviewer** command to start the VNC viewer directly.

This is the syntax of the **rvn_viewer** command:

```
rvn_viewer <servername[:port]> [<other viewer options>] [-h]
```

- For a description of the command-line options, see “Command-line options for rvn_viewer.”
- For examples of using the **rvn_viewer** command, see “Running RVN” on page 40.

Command-line options for rvn_viewer

This section defines the options that can be specified on the **rvn_viewer** command.

servername
: Specifies the name of the server to connect to.

port
: If desktop isolation will be used, specifies the port for the virtual session.

other viewer options
: Specifies any necessary additional options, as documented by RealVNC for the Enterprise Edition VNC viewer command.

-h
: Displays help information for the command.

The RVN coordinator

The `rvn_coordinator` process handles the initial setup for connecting an RVN sender to an RVN receiver that is participating in the same session. Under both Linux and Windows, `rvn_coordinator` is designed to be started automatically.

- With VNC, the automatic startup is handled internally by the sender on both Windows and Linux.
- With X11 export, the automatic startup is handled by the **xinetd** command. However, you can start `rvn_coordinator` manually in these situations:
  - Your system administrator has not configured **xinetd** to start `rvn_coordinator` automatically.
  - You need to use a different set of ports than those defined by the default `rvn_coordinator` configuration.
    - This could occur if you experience conflicts with the usage of other ports in your network.

Notes:

1. `rvn_coordinator` runs on the same system as the RVN sender (the application server).
2. For information about the environment variables for `rvn_coordinator`, see "Environment variables for rvn_coordinator" on page 93.
3. The following information pertains only to application servers that are running under Linux:
   - To manually invoke the RVN coordinator, use the **rvn_coordinator** command.
   - For information about setting up `rvn_coordinator`, refer to "Configuring the RVN coordinator (Linux only)" on page 61.
   - For details about configuring **xinetd**, refer to the `xinetd` and `xinetd.conf` man pages.

Running RVN

"RVN sessions" on page 22 provided several illustrations for common RVN configurations. The information in this section describes the basic steps necessary to run RVN using the following configurations:

- "Using X11 export mode with RVN (Linux application servers only)"
- "Using VNC with RVN" on page 41
- "Using VNC with RVN for desktop isolation (Linux application servers only)" on page 41
- "Using VNC with RVN to run multiple applications (Linux application servers only)" on page 42

Using X11 export mode with RVN (Linux application servers only)

When you are using X11 export mode, RVN permits only one receiver. In this configuration, RVN code runs on the end station without VNC. Figure 4 on page 24 illustrates the interaction between sender and receiver for this RVN configuration.

Note: Although the application server must be running under Linux, the end stations can run under Linux or Windows.

On the receiver

1. `export DISPLAY=:0`
2. `xhost +`
   - You can also use alternative methods for allowing X connections.
3. `rvn_receiver <host> <confID> <confkey> <nickname>`

On the sender

1. `export DISPLAY=receiver:0.0`
2. `RVN_DASHBOARD_DISPLAY=:0.0`
3. Start the RVN launcher.

   Note: You can bypass the launcher by setting the necessary environment variables and then running the **rvn_sender** command with the application name and arguments.
4. In the main window for the RVN launcher, click **X11 export mode** and provide the path to the application.

5. If the application requires any arguments, click **Options** and specify the arguments in the Options window. Then click **OK** to return to the main window.
6. Click **Launch**.
7. In the "Conference coordination" window, provide the conference key and the conference ID. Then click **OK**.

   Note: The values for the conference key and conference ID must be conveyed to all participants before the start of the conference.

Using VNC with RVN

In a basic VNC configuration or in a collaborative configuration with multiple end stations, RVN code runs on the end station with the VNC code. Figure 5 on page 25 illustrates the interaction between sender and receiver for a basic VNC configuration. Figure 6 on page 25 illustrates the interaction between sender and receiver for the RVN collaborative configuration (which can have up to eight end stations). There is no need to specify a conference ID and conference key in either case, because this information is automatically generated and processed during VNC authentication.

On each end station

Start the `vncviewer`.

- **Linux:** Enter `rvn_viewer <servername>` on the command line.
- **Windows:** Launch the `vncviewer` with the same information as indicated on the `rvn_viewer` command, using either its desktop icon or the **Start** menu.

On the sender

1. Start the RVN launcher.

   Note: You can bypass the launcher by setting the necessary environment variables and then running the **rvn_sender** command with the application name and arguments.
2. In the main window for the RVN launcher, click **VNC mode** and provide the path to the application.
3. If the application requires any arguments, click **Options** and specify the arguments in the Options window. Then click **OK** to return to the main window.
4. Click **Launch**.

Using VNC with RVN for desktop isolation (Linux application servers only)

This example starts a new `vncserver` (display number 9 in the following example) on the application sender. This example displays only the OpenGL application using RVN on the remote end stations (refer to Figure 7 on page 26). There is no need to specify a conference ID and conference key, because this information is automatically generated and processed during VNC authentication.

Notes:

1. Although the application server must be running under Linux, the end stations can run under Linux or Windows.
2. The new `vncserver` uses the display number explicitly specified on the command line (`:9` in the following example). If you do not specify a display, RVN defaults to the next available display number.

On the sender

1. Log in to the sender and run the following commands:

```
vncserver –depth 24 -pixelformat rgb888 :9
export DISPLAY=:9
export RVN_ALTERNATE_VISUALS=1  #only if needed
```

   Note: If an application works on the root console but not for desktop isolation, you might be able to address the problem by specifying export RVN_ALTERNATE_VISUALS=1 on the next attempt. This setting asks for a 24-bit true-color non-stereo visual, irrespective of what the application tried to request.

2. Start the RVN launcher.

   Note: You can bypass the launcher by setting the necessary environment variables and then running the **rvn_sender** command with the application name and arguments.

3. In the main window for the RVN launcher, click **VNC mode** and provide the path to the application.
4. If the application requires any arguments, click **Options** and specify the arguments in the Options window. Then click **OK** to return to the main window.
5. Click **Launch**.

On each end station

Start the `vncviewer`.

- **Linux:** Enter `rvn_viewer -shared <servername>:9` on the command line.
- **Windows:** Launch the `vncviewer` with the same information as indicated on the `rvn_viewer` command, using either its desktop icon or the **Start** menu.

Using VNC with RVN to run multiple applications (Linux application servers only)

In this example, multiple applications will be run on the RVN sender. Each application will be viewed on multiple end stations. These end stations have DCV code installed.

Note: Although the application server must be running under Linux, the end stations can run under Linux or Windows.

On the sender

Note: Someone (such as the system administrator) might have already completed the setup in the steps that precede starting RVN sessions for the applications.

1. Configure the **xstartup** script in `~/.vnc` This is typically a one-time setup for configuring VNC, such as to define the desktop properties.

```
#!/bin/sh

[ -r $HOME/.Xresources ] && xrdb $HOME/.Xresources
xsetroot -solid grey
#vncconfig -iconic &
xterm -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
#twm &
gnome-session &
```

Note: This is one possible configuration for the **xstartup** script.

2. Ensure that `X` is started on `:0` and that **xhost + localhost** gives every local user access to it.
3. Using the command **vncserver -geometry 1024x768 -depth 24 -pixelformat rgb888**, initiate one `vncserver` session for each application that you will run on the sender.

   Notes:

   a. Record the display number shown after invoking each `vncserver` session. The end stations will need this information to connect to the appropriate `vncserver` session.

   b. The geometry described in the command must match the desired display geometry.

4. For each application, start a separate RVN session using the RVN application launcher to invoke the application and its arguments. The launcher will set the appropriate `DISPLAY=:<n>` value for the associated `vncserver` session, and will set `RVN_USE_VNC=1`.

   Note: If an application works on the root console but not on the end stations, you might be able to address the problem by specifying export RVN_ALTERNATE_VISUALS=1 on the next attempt. This setting asks for a 24-bit true-color non-stereo visual, irrespective of what the application tried to request.

On each end station

Start the `vncviewer`, and specify the host name of the sender and the display number associated with the `vncserver` session for the running application.

- **Linux:** Enter `rvn_viewer -shared <servername>:<n>` on the command line.
- **Windows:** Launch the `vncviewer` with the same information as indicated on the `rvn_viewer` command, using either its desktop icon or the **Start** menu.

User guide: RVN

Chapter 4. SVN and RVN interoperability

When running in interoperability mode, both the SVN and RVN functions of DCV function simultaneously, in a joint SVN-RVN session. This allows both remote visualization and immersive visualization of OpenGL applications within a single DCV session. With interoperability, the application runs on the client machine and loads the SVN OpenGL library. The SVN OpenGL library encodes the OpenGL calls for transmission to the SVN rendering servers. As in a standard SVN session, the rendering servers must be listed in the SVN wall configuration file.

In addition to the rendering servers, the SVN OpenGL library also transmits the encoded OpenGL calls to the machine identified as the RVN sender by using the `-svnrvn` option on the **svn_sender** command. On the normal SVN rendering servers, OpenGL calls are decoded and then passed to the `system` OpenGL library (not the SVN OpenGL library). The specified RVN sender uses a modified **svn_server** executable file that decodes the transmitted OpenGL calls and passes them to the RVN OpenGL library. This allows the specified node to act as an RVN sender and allows end stations to connect and receive rendered pixels.

In most cases, the `vncserver` runs on the RVN sender. Therefore, users interact with the application on the sender even though the application is running on the original client machine. Because of that, you must set the `DISPLAY` environment variable to the RVN sender before running the joint SVN-RVN session. After the user starts the joint session, SVN functions as it would if RVN were not part of the session. In addition, the user can also invoke RVN scripts and establish end station connections as if RVN was functioning in stand-alone mode.

Notes:

1. Before you invoke **svn_sender** using interoperability mode, you must set the RVN options using the appropriate RVN environment variables (refer to “Environment variables for rvn_sender” on page 88).
2. The type of RVN connection affects the options that must be specified on the `svn_sender` command For more information, see the description of the `-svnrvn` option in “Command-line options for svn_sender” on page 8.
3. You no longer need to specify an SVN wall display with the `-svndpy` option or the `SVN_DISPLAY` environment variable.

For example, suppose you have an OpenGL application named *atlantis*, and an RVN sender named *vis99.watson.ibm.com*. This command invokes a joint SVN-RVN session using VNC:

```
export DISPLAY=vis99.watson.ibm.com:0.0
svn_sender -v -svnrvn myhost:0 ./atlantis
```

Note: This example does not illustrate usage of additional SVN command-line options. Your application might require other functions defined by those options. Refer to “Command-line options for svn_sender” on page 8 for additional information.

Chapter 5. SVN support for DMX servers

Distributed Multihead X (DMX) servers can be used to render the application display. SVN support for DMX servers allows an OpenGL application to export its entire user interface, including the 2-D graphics elements (windows and widgets), to a single composite display (wall display), providing better performance than a DMX-only implementation. An application can also take advantage of existing SVN features to export the graphics display to additional composite displays, without the 2-D graphics elements visible, or to an SVN-RVN session at the same time.

To use SVN support for DMX servers, follow these steps:

1. Install the DMX server. For more information, refer to "Installing the DMX server."
2. Start the DMX server. For more information, refer to "Starting the DMX server."
3. Start SVN using DMX support. For more information, refer to "Starting SVN using DMX support" on page 48.

Installing the DMX server

Before you can use DMX support in SVN, you must install the DMX server.

To download the DMX server RPM:

1. On the DMX Web site (http://dmx.sourceforge.net/), click the **Download DMX** link, then select the RPM for i386 systems.
2. Log in as root and use the **rpm -i** command to install the RPM. Here is an example: `rpm -i dmx-1.2.20040630-1.i386.rpm`

Starting the DMX server

Before starting an SVN session using DMX support, you must start a DMX server. The DMX server can be either a root window display server or a non-root window display server.

Here is a sample invocation of a DMX server as a non-root window display server:

```
Xdmx :1 -nowindowopt +xinerama -fontpath /usr/X11R6/lib/X11/fonts/misc
-configfile wall.config -config dv -input $DISPLAY -ac &
sleep 3
mwm -display :1 &
```

- `-nowindowopt` disables a lazy window creation option.
- `+xinerama` enables spanning the logical display across multiple physical displays.
- `-fontpath` specifies the path to the fonts to be used by the DMX server.
- `-configfile` specifies the path to the DMX configuration file.
- `-config` specifies the name of a configuration defined in the DMX configuration file.
- `-input` specifies the source to use for input.
- `-ac` suppresses authorization checks by the DMX server.

The **mwm** command in the example invokes the Motif window manager as the window manager for the DMX display. You can substitute the window manager of your choice.

Here is a sample wall configuration file for a 2 x 1 display:

```
virtual dv {
wall 2x1 deepview3:0.0 deepview4:0.0 ;
}
```

For more information, refer to the Xdmx man page or to the DMX Web site at http://dmx.sourceforge.net/.

Starting SVN using DMX support

To invoke DMX support when starting SVN, include the `-clientdpy` option on the **svn_sender** command, specifying the name of the DMX server to be used. Use the same format for the server name as when setting the DISPLAY environment variable before invoking an X application. When you specify a DMX server on the `-clientdpy` option, the `-clientrender` option is ignored.

Additional displays are optional. To specify them, use the `-svndpy` and `-svnrvn` options on the **svn_sender** command, or the `SVN_DISPLAY` environment variable.

Here is a sample SVN invocation for a DMX-only session, assuming all other required SVN environment variables are set. In this example, the `SVN_DISPLAY` environment variable is not set.

```
svn_sender -clientdpy deepview5:1 /usr/local/bin/atlantis
```

Here is a sample SVN invocation where a DMX session and a SVN wall display are used.

```
svn_sender -svndpy /home/deepview/wall.cfg -clientdpy deepview5:1 /usr/local/bin/atlantis
```

For more information about the **svn_sender** command, refer to “The svn_sender command” on page 8. For more information about related environment variables, refer to “Environment variables for svn_sender” on page 85.

Part 2. Installing IBM Deep Computing Visualization software

Part 2 of this book provides installation and problem determination information for system administrators who are responsible for managing DCV. Refer to the following chapters:

Chapter 6. Prerequisites for installing Deep Computing Visualization

This section lists the hardware and software prerequisites for installing DCV.

General prerequisites and configuration requirements for Deep Computing Visualization

User name (SVN)

The user name must be able to log in to each of the rendering servers and the application host.

Multiple displays (SVN)

If you intend to run more than a single display attached to an application host or a rendering server, set the Linux `runmode` value to 3. In addition, log in as the user of the SVN application and manually run the following commands on each rendering server:

- `startx`
- `xhost + <application host name>`

Notes:

1. If you intend to run only a single display, SVN supports setting the Linux `runmode` value to 5. This starts the X server when the node is booted. However, this might also export the desktop for the root user.
2. You can specify multiple nodes, such as an application host and associated rendering servers. Here is an example: `xhost + appl1 render52 render53 render54`
3. If you encounter problems, refer to “X server display problems” on page 71.

Required software for Linux systems

In addition to any software required for your application, the following packages should be installed with Linux:

- OpenGL
- X Windows
- C++ Runtime
- Secure Shell (OpenSSH), if required for remote access
- Remote shell (RSH), if required for remote access
- SVN only: DMX, if desired for a specific configuration

Shared file systems (SVN)

The wall configuration file should be in a shared file system that is readable by the application host and all of the rendering servers. Also, the default directory specified by `$SVN_HOME` or `$HOME` must be in a shared file system writable by the application host and readable by the rendering servers.

Notes:

1. It is the user's file system that should be shared, and not the root file system.
2. A shared file system is not required for RVN.

X server configuration (SVN)
Ensure that the X servers are configured consistently across all of the rendering nodes.

X server configuration (RVN)
If you are installing DCV on a supported X server, you must configure the X server with 24-bit color depth. To do so, specify `-depth 24` on the **vncserver** command. In addition, you must also use the **startx** command with `-depth 24` specified. If the pixel format is not `rgb888` (meaning 8 bits for red, green, and blue components), you must also specify `-pixelformat rgb888` on the **vncserver** command. Otherwise, the system will not display the correct colors.

Notes:

1. After you install DCV but before you start RVN for the first time, either the XF86 configuration file `/etc/X11/XF86Config` (RHEL3) or the xorg configuration file `/etc/X11/xorg.conf` (RHEL4) must be updated with 24-bit color depth.
2. After you reconfigure the X server, either the XF86 configuration file `/etc/X11/XF86Config` (RHEL3) or the xorg configuration file `/etc/X11/xorg.conf` (RHEL4) must be updated with 24-bit color depth.

The following example illustrates a pragma from an XF86 configuration file with the default 24-bit color depth:

```
Section "Screen"
        Identifier "Screen0"
        Device     "Videocard0"
        Monitor    "Benq"
        DefaultDepth     24
        SubSection "Display"
                Depth     24
                Modes    "1920x1200_60" "1600x1200"
                         "1400x1050" "1280x1024" "1280x960"
                         "1152x864" "1024x768" "800x600" "640x480"
        EndSubSection
        SubSection "Display"
                Depth     16
                Modes    "800x600" "640x480"
        EndSubSection
EndSection
```

SVN prerequisites

If you are going to use **rsh** for remote access, the installation should enable remote shell, and include all the rendering servers and the application host in the authorization files for **rsh** *(/etc/hosts or $HOME/.rhosts)* on each node (including the application host).

If you are going to use **ssh**, the public key of the user on the host needs to be generated and made accessible to the user on each rendering server and to the application host itself. Typically this information is found in `~/.ssh/authorized_keys`. This allows remote access without requiring passwords to be supplied by the user.

If you are using SVN with an IntelliStation® workstation or System x™ server, install an InfiniBand high-performance network to interconnect the application host node and the rendering server nodes.

For SVN, ensure that IP works correctly between the application host node and the rendering server nodes, and that both the Direct Access Transport and MPI libraries from the vendor are installed on each node.

Note: Direct Access Transport is required only with IntelliStation application host nodes.

SVN requires a shared MPI library, named *libmpich.so*, on the application host and rendering server nodes. Most vendors (such as Cisco/TopSpin) provide this shared library or allow you to compile one yourself. In some cases, the MPI library might be called something other than `libmpich.so` (for example, the Scali MPI shared library is called `libmpi.so`). If the vendor does not name their library `libmpich.so`, you must copy or link the vendor supplied libraries and create the needed `libmpich.so` library for SVN to run properly. For example, in the case of Scali MPI, running the command **/bin/ln -s libmpi.so libmpich.so** would create the required SVN library.

Create a shared file system that is accessible to the application host and the rendering servers for SVN. The full path-name of a directory in this file system must be specified as the `SVN_HOME` environment variable.

To run SVN, you must install one of the following MPI implementations. For details about the required level, refer to Table 4 on page 54.

Scali MPI Connect

The Scali Web site lists a Technical FAQs section that includes the following information:

- Tech FAQ Installation
- Tech FAQ Scali MPI

Cisco/TopSpin

Cisco/TopSpin provides a switch, adapters, drivers, an InfiniBand library, and an MPI library.

MPICH

MPICH is a freely available implementation of the MPI Standard. This MPI implementation was developed jointly with Mississippi State University.

Notes:

1. MPICH is not customized to run over InfiniBand. (This is different from the Scali and Cisco/TopSpin MPI implementations.) As a result, there is a drop in performance when using MPICH in comparison to using Scali or Cisco/TopSpin.
2. To work without passwords, MPICH requires **rsh** (set up the `~/.rhosts` file).
3. MPICH can use **ssh** instead of **rsh** if you set up **ssh** to work without passwords.

Table 3 lists the hardware requirements for SVN implementation.

Table 3. Hardware requirements for SVN

Device	Requirement
Application hosts	One of the following machines: • IBM IntelliStation A Pro or Z Pro workstation, 32-bit or 64-bit • IBM System x 3950 or 3755 system

Table 3. Hardware requirements for SVN (continued)

Device	Requirement
Rendering servers	IBM IntelliStation A Pro or Z Pro workstation, 32-bit or 64-bit
Cisco/TopSpin switch	Cisco/TopSpin 120 InfiniBand
Graphics card	NVIDIA Quadro FX family of ultra-high-end graphics adapters

Table 4 lists the software requirements for SVN implementation.

Table 4. Software requirements for SVN

Software	Level	Notes
DCV	1.3	
Operating system	Red Hat Enterprise Linux (RHEL) Workstation Edition using either: • Release 3 Update 7 or later (with availability of appropriate drivers) • Release 4 Update 3 (with availability of appropriate drivers)	
NVIDIA driver	1.0-8776	To download the latest NVIDIA graphic adapter driver version for 32-bit or 64-bit versions of Linux and the adapters installed, go to the IBM support Web site at http://www.ibm.com/products/finder (use the links for Support and downloads). Configure the device driver by modifying */etc/X11/XF86Config* to specify which of the adapter ports are associated with which X Windows servers.
MPI implementation	One of the following products: • **Cisco/TopSpin driver:** 3.1.0-113 or later update • **Scali MPI Connect:** 4.4.0 + Fix SSP44 or later update • **MPICH:** 1.2.7	• **Cisco/TopSpin driver:** http://www.topspin.com/index.html • **Scali MPI Connect:** http://www.scali.com • **MPICH:** http://www-unix.mcs.anl.gov/mpi/mpich/download.html
OpenGL library	From device driver or RHEL distribution DCV supports OpenGL up through specification level 2.0, including support for shader programming	
DMX	RPM for i386 systems	http://dmx.sourceforge.net/

Requirements for 32-bit and 64-bit SVN implementations

For SVN, the application host node and rendering server nodes must have the same version of Linux installed (all 32-bit or all 64-bit), and the address mode of the application must be the same as the version of Linux. In particular, SVN does not support 32-bit applications running on 64-bit Linux. Therefore, SVN has the following requirements for 32-bit and 64-bit implementations:

- Nodes running 32-bit and 64-bit kernels cannot be mixed.

- The application host and the rendering servers must either all run 32-bit kernels or all run 64-bit kernels.
- If the application host is running a 64-bit kernel, 32-bit applications cannot be run there.
  - **Note:** This limitation is at the MPI transport layer. Refer to your MPI documentation to see if this limitation has been removed.

Attention: The DCV 64-bit RPM is a delta to the 32-bit RPM. Therefore, when you are installing DCV on a 64-bit system, you must install the 32-bit RPM before you install the 64-bit RPM.

RVN prerequisites

Table 5 lists the hardware requirements for RVN implementation.

Table 5. Hardware requirements for RVN

Device	Requirement
Application servers	One of the following machines: • IBM IntelliStation A Pro or Z Pro workstation, 32-bit or 64-bit • IBM System x 3950 or 3755 system
End stations	Most systems (including laptop computers) are supported, because RVN uses that hardware only as a display
Graphics card (application server only)	NVIDIA Quadro FX family of ultra-high-end graphics adapters

Table 6 lists the software requirements for RVN.

Table 6. Software requirements for RVN

Software	Level	Notes
DCV	1.3	
Operating system	One of the following products: • Red Hat Enterprise Linux (RHEL) Workstation Edition – Release 3 Update 7, or later (with availability of appropriate drivers) – Release 4 Update 3, or later (with availability of appropriate drivers) • Windows XP Professional, Service Pack 2 (32-bit only)	For RVN, both 32-bit and 64-bit executable files are supported on 64-bit Linux. The end station can run either 32-bit or 64-bit Linux, but the server for the accelerated graphics function of RVN is a 32-bit executable file.
VNC • vncserver on the application server • vncviewer on the end stations	VNC Visualization Edition	• For details about how to obtain VNC Visualization Edition, see the README for DCV 1.3. • Using any other VNC product (whether from RealVNC or another company) on the application server or end stations is not supported. • To obtain maximum stability, performance, and function, install DCV 1.3 and VNC Visualization Edition on the application server and the end stations.

Table 6. Software requirements for RVN (continued)

Software	Level	Notes
OpenGL library (application server only)	From device driver or RHEL distribution DCV supports OpenGL up through specification level 2.0, including support for shader programming	
NVIDIA video driver (application server only)	• **Linux:** 1.0-8776 • **Windows:** 91.36	• **Linux:** To download the latest NVIDIA graphic adapter driver version for 32-bit or 64-bit versions of Linux and the adapters installed, go to the IBM support Web site at http://www.ibm.com/products/finder (use the links for Support and downloads). Configure the device driver by modifying */etc/X11/XF86Config* to specify which of the adapter ports are associated with which X Windows servers. • **Windows:** Download the NVIDIA video driver from http://www.ibm.com/support/docview.wss?uid=psg1MIGR-66220

Note: For sessions that use X11 DISPLAY export mode, an X server must be running on the application server. You can use one of the following software packages for this purpose:

- Exceed X server, available at www.hummingbird.com
- Cygwin X server, available at www.cygwin.com
- `vncserver` from RealVNC (for details, refer to Table 6 on page 55)

Chapter 7. Installing Deep Computing Visualization

Refer to the following sections for hardware-specific installation instructions:

- "Installing Deep Computing Visualization on Linux application hosts"
- "Installing Deep Computing Visualization RVN on Linux end stations" on page 58
- "Installing Deep Computing Visualization RVN on Microsoft Windows application hosts" on page 59
- "Installing Deep Computing Visualization RVN on Microsoft Windows end stations" on page 59

Notes:

1. DCV 1.2 and DCV 1.3 cannot coexist on the same machine.
2. For RVN, DCV 1.2 end stations are compatible with DCV 1.3 application servers. However, DCV 1.3 end stations are not compatible with DCV 1.2 application servers.
3. Check the following Web sites:
   - For the latest information about DCV, including FAQs, go to http://www.ibm.com/servers/deepcomputing/visualization.
   - For the latest updates to DCV, go to http://techsupport.services.ibm.com/server/cluster/home.html.
4. The version-release numbers shown here relate to Version 1 Release 3. If you are installing a subsequent version or release, or both, use the appropriate v-r numbers from the installation medium.

Installing Deep Computing Visualization on Linux application hosts

To install DCV on a machine running Linux that will be used as an SVN application host or as an RVN application server and end station, use the following procedure.

Note: For information about installing DCV on a machine running Linux that will be used only as an RVN end station, refer to "Installing Deep Computing Visualization RVN on Linux end stations" on page 58.

1. If you are migrating from DCV 1.2 to DCV 1.3, use the **rpm -e** command to uninstall DCV 1.2, then proceed to the next step to install DCV 1.3.
2. Copy the following DCV RPMs to a convenient directory:
   - dcv.license_acceptance-1-3.i686.rpm
     - This RPM is required only the first time you install DCV on each node.
   - dcv-1.3.0-0.i686.rpm
   - If you require 64-bit enabled DCV, you must also copy:
     - dcv64-1.3.0-0.x86_64.rpm

   Note: If you are using the product CD, you only have to mount the CD media in a suitable drive.
3. Log in as the root user.
4. Change directory (**cd**) to the directory containing the RPMs.
5. If you want to pre-accept the license agreement and avoid the license prompt, set the environment variable `DCV_LICENSE_PROMPT` using the command `export DCV_LICENSE_PROMPT=0`.

Notes:

a. You should not pre-accept the license agreement during the original DCV installation.

b. If you do not pre-accept the license agreement, make sure the `DISPLAY` environment variable is set to an appropriate value using the following command: `export DISPLAY=:0`

6. Run the following command:

```
rpm –i dcv.license_acceptance-1-3.i686.rpm dcv-1.3.0-0.i686.rpm
```

Notes:

a. If you did not pre-accept `DCV_LICENSE_PROMPT`, this command displays the license terms and prompts you for acceptance.
   - If you accept the license, the system continues with the installation.
   - If you do not accept the license, installation stops.

b. On successful completion, DCV installs:
   - The operational files in */opt/IBM/dcv*
   - The license agreements in */etc/opt/IBM/dcv/license*

   In addition, an IBM JRE is installed along with the code for the RVN application host. This is for use by the RVN application launcher and dashboard, which are Java™ applications.

7. If you are installing 64-bit enabled DCV, you must also run the following command:

```
rpm –i dcv64-1.3.0-0.x86_64.rpm
```

8. Refer to Chapter 9, “Installation verification for Deep Computing Visualization,” on page 65 to complete the installation.

Installing Deep Computing Visualization RVN on Linux end stations

To install RVN on a machine running Linux that will be used as an RVN end station, use the following procedure.

Note: For information about installing DCV on application servers that are running under Linux, refer to “Installing Deep Computing Visualization on Linux application hosts” on page 57.

1. If you are migrating from DCV 1.2 to DCV 1.3, use the **rpm -e** command to uninstall DCV 1.2, then proceed to the next step to install DCV 1.3.
2. Copy the following DCV RPMs to a convenient directory:
   - dcv.endstation-1.3.0-0.i686.rpm

   Note: If you are using the product CD, you only have to mount the CD media in a suitable drive.
3. Log in as the root user.
4. Change directory (**cd**) to the directory containing the RPMs.
5. Run the following command:

```
rpm -i --nodeps dcv.endstation-1.3.0-0.i686.rpm
```

6. Refer to Chapter 9, “Installation verification for Deep Computing Visualization,” on page 65 to complete the installation.

Installing Deep Computing Visualization RVN on Microsoft Windows application hosts

To install DCV RVN on a machine running Windows that will be used as an RVN application server (or application host), use the following procedure.

Note: For information about installing DCV on a machine running Windows that will be used only as an RVN end station, refer to "Installing Deep Computing Visualization RVN on Microsoft Windows end stations."

To install DCV RVN on an application server that is running under Windows, run **Setup.exe**. The DCV code is installed in `C:\Program Files\IBM\IBM Deep Computing Visualization`.

In addition to installing the DCV RVN code and the *Deep Computing Installation and User Guide* manual, **Setup.exe** completes these steps:

- Deletes the previous release of DCV.
- Installs IBM JRE. This is for use by the RVN application launcher and dashboard, which are Java applications.
- Modifies the environment path to make the DCV files available to VNC.
- Creates a shortcut to **rvn_sender.bat** called **RVN Sender** in **Start > Programs > IBM > IBM Deep Computing Visualization**. This shortcut does not have any arguments and will therefore start the RVN application launcher. For more information about using the launcher, refer to "The RVN application launcher" on page 28.

Installing Deep Computing Visualization RVN on Microsoft Windows end stations

Follow this procedure for each Windows workstation that will be used as an RVN end station.

Note: For information about installing RVN on application servers, refer to "Installing Deep Computing Visualization RVN on Microsoft Windows application hosts."

To install DCV RVN on an end station that is running under Windows, run **End station setup.exe**. The DCV code is installed in `C:\Program Files\IBM\IBM Deep Computing Visualization`.

In addition to installing the DCV RVN code and the *Deep Computing Installation and User Guide* manual, **End station setup.exe** completes these steps:

- Deletes the previous release of DCV.
- Modifies the environment path to make the DCV files available to VNC.

Chapter 8. RVN setup

RVN assumes that at least one receiver is able to run applications on the application server. This can be done using one of the following methods:

- The VNC desktop
- Through remote login
- A remote shell
- OpenSSH

As part of the system setup:

- A `vncviewer` must be installed on each end station. For more information about DCV requirements for VNC, refer to "RVN prerequisites" on page 55 and Chapter 7, "Installing Deep Computing Visualization," on page 57.
- Linux only: You must configure the Linux application server to run a remotely viewed X server.
  - This requires modifying the X server configuration file (such as */etc/X11/XF86Config*).
- Linux only: If you are using OpenSSH, each end station should create a public/private key pair, and you must verify that OpenSSH added the public key to the *$HOME/.ssh/authorized_keys* files on the application server.

For details about additional RVN setup tasks, refer to:

- "Configuring the RVN coordinator (Linux only)"
- "Configuring RVN with VNC for remote desktopping" on page 62

Configuring the RVN coordinator (Linux only)

The `rvn_coordinator` process handles the initial setup for connecting an RVN sender to an RVN receiver that is participating in the same conference. `rvn_coordinator` is designed to be started automatically by `xinetd` on the same system as the RVN sender (the application server).

The DCV code supplies two sample files to help you use the RVN coordinator: `/opt/IBM/dcv/rvn/etc/services` and `/opt/IBM/dcv/rvn/etc/dcv.xinetd`. The code also includes instructions for adding a low-privilege user ID for running the coordinator. In order to set up `rvn_coordinator`, you need to make changes to `/etc/services` and create a new file, `dcv-rvn` in the `/etc/xinetd.d` directory.

To do this, add the following entries to the `/etc/services` file:

```
dcv-rvn 7200/tcp # IBM dcv rvn coordinator
port dcv-rvn-top 7220/tcp # ports from dcv-rvn to dcv-rvn-top reserved by IBM
```

Notes:

1. The `dcv-rvn` service specifies the default port number used by RVN when establishing a RVN conference.
2. The `dcv-rvn-top` service:
   - Specifies the highest port number that the `rvn_coordinator` process will use when allocating ports used in RVN conferences.
   - Must specify a port number higher than `dcv-rvn` and be large enough to support the maximum number of simultaneous RVN conferences allowed on your system.

3. You can change the dcv-rvn and dcv-rvn-top values to avoid port conflicts with other applications. However, you must also transmit those values to any users who will be participating in an RVN conference on this system.
4. You can also reserve additional ports between dcv-rvn and dcv-rvn-top for RVN usage by specifying them in the /etc/services file.

In addition to updating dcv-rvn and dcv-rvn-top in the /etc/services file, you must also create a dcv-rvn file in the /etc/xinetd.d directory. This file uses the following format:

```
    service dcv-rvn
{
          socket_type       = stream
          wait              = yes
          user              = pdk
          log_on_success    += USERID
          log_on_failure    += USERID
          server            = /opt/IBM/dcv/rvn/bin/rvn_coordinator
           server_args       = daemon
          disable           = no
}
```

Notes:

1. When you create the dcv-rvn file, make sure that user specifies a user ID with general user privileges that will be used to start the rvn_coordinator process. Do not specify a user with root privileges.
2. After you have made these changes, you must restart xinetd to enable the rvn_coordinator process. You can do this by issuing the following command:

   ```
   /sbin/service xinetd restart
   ```
3. An example /etc/services file fragment (/opt/IBM/dcv/rvn/etc/services) and xinetd.d file (/opt/IBM/dcv/rvn/etc/dcv.xinetd) are provided.

Configuring RVN with VNC for remote desktopping

The use of VNC for remote desktopping along with RVN provides a powerful mechanism for collaboration in 3-D graphics environments. VNC provides a mechanism for distributing the 2-D application interface to multiple collaborators, and RVN provides a mechanism for distributing the graphically intensive 3-D images. VNC involves:

- A vncserver running on the RVN sender
- One or more vncviewer instances running on the RVN receivers

Configuring VNC under Linux

In the X11 environment under Linux, VNC can run as a runtime-loaded extension to the existing XFree86 or Xorg X server, or as a software desktop using its own X server. To enable the existing X server for VNC, you must make several entries in the configuration file for the X server, typically /etc/X11/XF86Config or /etc/X11/xorg.conf. For detailed information about configuring the authentication and security features of VNC, go to http://www.realvnc.com/products/enterprise/4.2/x0.htm and http://www.realvnc.com/products/enterprise/4.2/unixconfig.html.

Notes:

1. DCV requires a specific level of VNC. For more information about DCV requirements for VNC, refer to “RVN prerequisites” on page 55 and Chapter 7, “Installing Deep Computing Visualization,” on page 57.
2. For more information about vncserver and vncviewer, go to www.realvnc.com.

Configuring VNC under Windows

Most of the defaults do not need to be changed. However, you should define the following settings:

- Select full-color.
- Set 32-bit or 24-bit color 32-bit or 24-bit color for Windows desktop properties.
- Select shared connection for collaboration.

Chapter 9. Installation verification for Deep Computing Visualization

Follow these steps to verify DCV installation:

1. Log on to the application host as the user for which DCV is being installed.

   Note: Do not log in as root during installation verification.
2. On the application host, make a subdirectory *dcv_ivt* in the file system shared by the application host and the rendering servers.

   Notes:

   a. Make sure that the subdirectory is writable and searchable by the application host.

   b. Make sure that the subdirectory is readable and searchable by the rendering servers.

   c. If you are only going to use RVN, you do not have to make the subdirectory shared.
3. Copy all the files in the directory */opt/IBM/dcv/svn/test* to *dcv_ivt*
4. Change directory (**cd**) to *dcv_ivt*

Depending on your application, select the appropriate procedure and continue:

- For SVN
- For RVN

For SVN, also perform these steps:

1. Set the appropriate environment variables for building MPI executable files such as MPI location and others.
2. "**make**" the `rotvp` executable file.

   Note: Use the executable file that was provided. Do not **make** `bcast_bandwidth` or `bcast_bandwidth64`.
3. Create a wall configuration file *wall.cfg* in *dcv_ivt* that describes each rendering server.

   Notes:

   a. For the geometry specifications, use vwidth = width = horizontal display size, vheight = height = vertical display size, and x_offset = y_offset = 0.

   b. Refer to Chapter 2, “Scalable Visual Networking,” on page 5 for additional information.
4. Set the following environment variables:

   a. `export SVN_MPICOMM=mpicomm.ib.rsh` (or `.ssh` if using OpenSSH)

   b. `export SVN_HOME=full-path` to *dcv_ivt*

   c. `export SVN_DISPLAY=`*wall.cfg*

   d. `export SVN_MPIBIN=`full path to the **mpirun** command for the MPI implementation

   e. `export SVN_MPILIB=`full path to the *libmpich.so* shared object for the MPI implementation
5. Run the **svn_sender** command with the `mpitest` option set:

   ```
   svn_sender –mpitest 1 ./bcast_bandwidth 100000 100
   ```

This should produce output similar to the following (shown for four rendering servers):

```
SVN: bcast_bandwidth: Task 1 running on node deepview3.pok.ibm.com
SVN: bcast_bandwidth: Task 3 running on node deepview5.pok.ibm.com
SVN: bcast_bandwidth: Task 2 running on node deepview4.pok.ibm.com
SVN: bcast_bandwidth: Task 0 running on node deepview2.pok.ibm.com
SVN: broadcasting 100000 doublewords, 100 iterations
SVN: bcast_bandwidth: Task 4 running on node deepview6.pok.ibm.com
Start timer value = 850000.0 us
End timer value = 1220000.0 us
Average Elapsed time = 3700.0 useconds per loop
BROADCAST BANDWIDTH TO OTHERS = 864.865*10**6 Bytes/sec
BROADCAST BANDWIDTH WITH SELF = 1081.081*10**6 Bytes/sec
```

Note: Successful execution of this step ensures that the MPI environment is set up correctly and that the application host can communicate with the application servers.

6. Run the **svn_sender** command for the **rotvp** application:

   ```
   svn_sender ./rotvp 1000
   ```

   Each server screen should display a blue background with copyright information (the SVN server banner) for five seconds. Each server should then display a copy of the application host screen, which is two rotating squares. In the window in which the **svn_sender** command was run, each server and the application host should display messages indicating that they have started.

   Note: Successful performance of this step ensures that the OpenGL intercept library and servers have been installed correctly.

For an RVN sender running under Linux and using X11 DISPLAY EXPORT, also perform these steps:

1. Make sure that you are in the *dcv_ivt* directory as described in the initial steps.
2. "**make**" the `rotvp` executable file on the application host.

   Note: Use the executable file that was provided. Do not **make** `bcast_bandwidth` or `bcast_bandwidth64`.
3. On the application server, set up and run the `rotvp` application:
   a. `export DISPLAY=endstat0:0.0`
   b. `rvn_sender –conf <myconf> -key <mkey>`
4. Start RVN on the end station (assume name is `endstat0`):
   a. `export DISPLAY=:0.0`
   b. `rvn_receiver <host> <myconf> <mkey> <nickname>`
5. Verify that the desktop for the RVN sender displays the RVN dashboard, and that the end-station screen displays the rotating squares.

Note: Successful performance of this step ensures that the RVN component and its OpenGL intercept library are installed correctly.

For an RVN sender running under Windows, also perform these steps:

1. From the Windows end station, launch the `vncviewer`.
2. In the **Sender** field of the viewer, enter the name of the RVN sender for the application. Then click **OK**.

3. Enter your password to authenticate to the RVN sender.
4. When you see the desktop for the RVN sender, follow these steps:
   a. From the **Start** menu, launch `rvn_sender.bat`.
   b. In the **Select an application** field of the RVN "Application launcher" window, enter the path to any OpenGL 3-D rendering test application, then click **Launch**.
   c. Verify that the desktop for the RVN sender displays the RVN dashboard, and that the end-station screen displays the appropriate graphic.
   d. Close the test application.
5. Close the `vncviewer` (if you are in full-screen mode, press F8).

Chapter 10. Problem determination

Refer to the following sections for diagnostic information:

Notes:

1. If the diagnostic procedures indicate that device-specific procedures are required, the information in this book will direct you to any additional manuals you might need.
2. Before contacting IBM for service, refer to the README files provided on the DCV page listed under "Related information" on page xii.
3. If you cannot determine the cause of failure, request the assistance of the IBM Field Support Center.
4. Appendix A, "Messages" provides additional diagnostic information, including "Product installation messages" on page 73.

SVN problems

If you encounter any problems using the SVN component, set `SVN_VERBOSE=`*1* to echo the script lines issued and to preserve any work files generated.

If the `svn_sender` script exits with an error, examine the error message and follow the suggested action. If that does not solve the problem, use local problem reporting procedures. In addition, here are some additional things to check:

- If the problem seems to be on the server side, check that the required libraries and executable files are installed on the node.
  - If needed, reinstall the libraries and executable files, and follow local problem reporting procedures if the system requires additional diagnostics.
- If **rsh** is used to start the remote MPI tasks, remember that it does not run the user profile *$HOME/.bash_profile*. However, **rsh** does run the *$HOME/.bashrc* file. This can affect which library paths are actually searched.
  - If needed, update the libraries path defined in the *$HOME/.bashrc* file, and follow local problem reporting procedures if the system requires additional diagnostics.

If the application runs but nothing is displayed on the server after the initial banner:

- Try enabling the `-windowselector` option on the **svn_sender** command and selecting the display window several times.
- Try the test application (**rotvp**) that is distributed with DCV.
- If the problem persists and the system requires additional diagnostics, follow local problem reporting procedures.

If performance is slow, make sure that a high-performance network is being used by both the application host and the servers. Run an MPI bandwidth test using the same configuration to see if MPI performance is acceptable. If MPI performance is not acceptable, verify that MPI and the supporting network hardware are properly

installed and configured as described in the vendor documentation. If system performance is still slow, follow local problem reporting procedures.

If the server or application halts or generates a core dump:

1. Verify that the application runs correctly as a stand-alone application on the node.
2. Collect the failure information and the core dump.
3. Contact IBM Service.

If one of the servers reports that it cannot open the display, make sure that the server has enabled all X connections by running the **xhost +** command. If you receive an application message indicating that it cannot open the display:

- Verify that you have enabled the application to access the display through the **xhost +** command (or equivalent authorization commands).
- If you cannot get the application to open the display, follow local problem reporting procedures.

If you receive a `permission denied` message:

- Review the appropriate `rsh` or `ssh` documentation and ensure that the access for `rsh` or `OpenSSH` has been set up.
- Make sure that the pass-phrases have been supplied to the SSH agents on the application host and on all of the rendering servers, as described in the `ssh` documentation.
- If you are still receiving the `permission denied` message, follow local problem reporting procedures.

Attention: If you cannot determine the cause for any of the problems listed above, contact IBM Service.

RVN problems

If one of the servers reports that it cannot open the display, make sure that the server has enabled all X connections by running the **xhost +** command. If you receive an application message indicating that it cannot open the display:

- Verify that you have enabled the application to access the display through the **xhost +** command (or equivalent authorization commands).
- If you cannot get the application to open the display, follow local problem reporting procedures.

If you encounter any problems using RVN, isolate the component causing the problem by varying the configuration as described in the following list. If any of these actions causes the system to issue a message, follow the actions suggested for the message. If a command is not working correctly, verify that the proper command-line options and environment variables are set (refer to Chapter 3, “Remote Visual Networking,” on page 21). If these actions do not solve the problem, follow local problem reporting procedures.

1. Use only a single end station.
2. Run without VNC.
3. Run with VNC but without enabling the accelerated graphics function.
4. Run with and without OpenSSH tunneling.
5. Run the installation verification procedure.

X server display problems

If you are having problems related to X server display, check the following:

- `/var/log/XFree86.0.log`
  - This file contains information about X.
- **xdpyinfo**
  - This display utility provides information for X.
- **xhost + vis80**
  - This command allows `vis80` to show X windows on the display.
- Run **glxgears**
  - If you are using the correct 3-D accelerated drivers, you should get more than 1000 frames per second.

Display problems

If you are having display problems, check the following:

- Confirm that the `DISPLAY` variable is correctly set.
- Confirm that you can invoke `xclock` and that it appears on the intended display.
  - If this fails, the display owner might need to issue the **xhost + localhost** command.
- From the server, verify that you can you access the local `DISPLAY (:0)`.
  - If this fails, the display owner might need to issue the **xhost + localhost** command.
  - The display owner might also need to issue the **chmod a+rw /dev/nv*** command.

NVIDIA display problems

If you are having NVIDIA display problems, check the following directories and take the suggested corrective actions if needed:

- cat /proc/driver/nvidia/version
  - Verify the correct driver level is installed, and reinstall if necessary. For information about the correct driver level, refer to “SVN prerequisites” on page 52.
- cat /proc/driver/nvidia/cards/0
  - Verify that the reported card is a supported display adapter. If the adapter is not supported, install a supported card and its associated driver.
- cat /proc/driver/nvidia/agp/card
  - Verify that you are receiving the expected output. If not, check the NVIDIA documentation for steps to resolve the problem.
- cat /proc/driver/nvidia/agp/host-bridge
  - Verify that you are receiving the expected output. If not, check the NVIDIA documentation for steps to resolve the problem.
- cat /proc/driver/nvidia/agp/status
  - Verify that you are receiving the expected output. If not, check the NVIDIA documentation for steps to resolve the problem.
- If these actions do not solve the problem, follow local problem reporting procedures.

Appendix A. Messages

Under special circumstances, DCV displays messages related to specific activities. These messages include:

Product installation messages

DCV License only requested. Product install is not attempted.

Explanation: You specified the environment variable `DCV_LICENSE_ONLY` as `1` so only the license acceptance process is run. This is appropriate if you are installing the executable files on a shared file system and the user only needs to license additional nodes for execution.

User response: If you intended to install DCV, reset the `DCV_LICENSE_ONLY` environment variable and rerun the install command.

DCV License agreement not accepted. Product install is abandoned.

Explanation: You have not responded yes to the prompt asking if the DCV license is to be accepted.

User response: If you intended to install DCV, you must rerun the install command and accept the license agreement when prompted.

RVN messages

DCV returns the following messages from the **rvn_receiver** command.

RVN: keys different length incoming = *key* **local=***key*

Explanation: The conference key given to **rvn_sender** is a different length than the conference key given to **rvn_receiver**.

User response: Specify the same conference key to **rvn_sender** and **rvn_receiver**.

RVN: keys do not match

Explanation: The conference key given to **rvn_sender** does not match the conference key given to **rvn_receiver**.

User response: Specify the same conference key to **rvn_sender** and **rvn_receiver**.

RVN: keys different length incoming longer than local=*key*

Explanation: The conference key given to **rvn_sender** is a different length than the conference key given to **rvn_receiver**.

User response: Specify the same conference key to **rvn_sender** and **rvn_receiver**.

RVN failed to detach shared memory

Explanation: Internal error.

User response: Contact IBM Service.

RVN: error loading libraries

Explanation: RVN was unable to load required libraries.

User response: Ensure that the system `libGL.so` is accessible at the default location or that `RVN_SYSTEM_OPENGL_LIB` is correctly set.

RVN: dashboard execv returns *return_code* **errno** *hex_errno decimal_errno*

Explanation: Failure when attempting to exec the dashboard.

User response: Contact IBM Service.

RVN: fork of dashboard fails

Explanation: Maximum number of user processes running.

User response: Reset maximum number of processes.

RVN: Initialization failure: RVN_SYSTEM_OPENGL_LIB not set

Explanation: RVN was unable to load the system OpenGL library.

User response: Ensure that `RVN_SYSTEM_OPENGL_LIB` is properly set.

RVN: dlopen of *library_name* **fails with:** *dlerror()*

Explanation: RVN was unable to load the specified library.

User response: Ensure that `RVN_SYSTEM_OPENGL_LIB` is properly set.

RVN: Fatal error. No FBConfigs returned from OpenGL library.

Explanation: RVN was unable to find a visual with the required attributes.

User response: Ensure that the X server is configured with the required attributes.

RVN: Memory allocation fails

Explanation: RVN was unable to allocate needed memory.

User response: Examine system memory usage. If unable to diagnose the problem, contact IBM Service.

RVN: LoadLibrary of *library_name*: **error** *dlerror()*, **...**

Explanation: An error occurred attempting to load the Intel® Performance Primitives libraries.

User response: Ensure that the Intel Performance Primitives are properly installed.

RVN: LoadLibrary of *library_name*: *dlerror()*, **aborting**

Explanation: An error occurred attempting to load the Intel Performance Primitives libraries.

User response: Ensure that the Intel Performance Primitives are properly installed.

RVN: Error: Compression failed to initialize

Explanation: Internal error.

User response: Contact IBM Service.

RVN: unable to create shared memory segment: *bytes_per_line* **bpl, h** *height*

Explanation: RVN was unable to create a shared memory segment.

User response: Ensure that the system is configured with sufficient shared memory.

RVN: specify an executable

Explanation: There was no executable file specified on the **rvn_sender** command.

User response: Reissue the **rvn_sender** command and specify an executable file for the graphics application.

DCV/RVN License agreement file not found on node *hostname*

Explanation: A copy of the license agreement file could not be found on the application host.

User response: Ensure that RVN is correctly installed.

RVN: unable to open display: *display_name*

Explanation: RVN was unable to open the specified display.

User response: Check the value of the `DISPLAY` environment variable and ensure that access has been granted through a mechanism such as `xhost` or `Xauthority`.

RVN: Compression encoder returns error

Explanation: An error was returned by the RVN internal compression encoding mechanism.

User response: Contact IBM Service.

RVN: sender error writing handshake

Explanation: An error was returned when writing from the sender to a receiver.

User response: Ensure that no network issues exist, then contact IBM Service.

RVN: sender error reading handshake

Explanation: An error was returned when attempting to read from a receiver.

User response: Ensure that no network issues exist, then contact IBM Service.

RVN: Compression decoder returns error

Explanation: An error was returned by the RVN internal compression decoder.

User response: Contact IBM Service.

RVN: error *errno* writing to receiver

Explanation: An error was detected when trying to write to the receiver.

User response: Ensure that no network issues exist, then contact IBM Service.

RVN: unable to create shared memory segment

Explanation: An error was returned when attempting to create a shared memory segment.

User response: Ensure that the system has sufficient memory and that old shared memory instances have been correctly cleaned up. If there are no memory issues, contact IBM Service.

RVN: unable to attach shared memory segment

Explanation: An error was returned when attempting to attach a shared memory segment.

User response: Contact IBM Service.

RVN: unable to XShmAttach

Explanation: An error was returned when attempting to call XShmAttach.

User response: Contact IBM Service.

RVN: glxChooseVisual returns error

Explanation: An error was returned from a call to `glxChooseVisual`.

User response: Contact IBM Service.

RVN: glxCreateContext returns error

Explanation: An error was returned from a call to `glxCreateContext`.

User response: Contact IBM Service.

RVN: receiver error reading handshake

Explanation: An error was detected when trying to read from the sender.

User response: Ensure that no network issues exist, then contact IBM Service.

RVN: receiver error writing handshake

Explanation: An error was detected when trying to write to the sender.

User response: Ensure that network connectivity exists between the sender and receiver. If no network issues exist, contact IBM Service.

RVN: receiver error in select

Explanation: RVN encountered an error when calling the select system call.

User response: Contact IBM Service.

RVN: Memory allocation fails

Explanation: RVN was unable to allocate needed memory.

User response: Examine system memory usage. If unable to diagnose the problem, contact IBM Service.

RVN: compression decoder returns error

Explanation: The RVN internal compression decoder returned an error.

User response: Contact IBM Service.

RVN: listen call returns error *err*

Explanation: Calling listen for the socket returns error *err*.

User response: Contact IBM Service.

RVN: read call returns error *err*

Explanation: Attempting to read from socket returns error *err*.

User response: Contact IBM Service.

RVN: Bad magic number *number*

Explanation: RVN received a corrupt packet.

User response: Contact IBM Service.

RVN: unable to open display *display*

Explanation: An attempt to open display *display* failed.

User response: Check settings of environment variables.

RVN: sender (*client_id***:***rvn_component_id***): error writing handshake**

Explanation: Internal error.

User response: Contact IBM Service.

RVN: sender (*client_id***:***rvn_component_id***): error reading handshake fd** *file_descriptor*

Explanation: Internal error.

User response: Contact IBM Service.

RVN: bind returns error *return_code*, **errno:** *errno*

Explanation: Internal error.

User response: Contact IBM Service.

RVN: read numBytes *num* **calls returns error** *errno*

Explanation: Internal error.

User response: Contact IBM Service.

RVN: receiver read error

Explanation: Internal error.

User response: Contact IBM Service.

RVN: receiver setup failure

Explanation: Internal error.

User response: Contact IBM Service.

SVN script messages

Error SVN: Path SVN_ROOT=*svn_root* does not exist

Explanation: The specified file path does not point to the installed version of DCV.

User response: Verify the location of DCV, usually in */opt/IBM/dcv*, and set the correct value of `SVN_ROOT` if necessary.

Error SVN: Path SVN_HOME=*svn_home* does not exist

Explanation: The specified file path to a shared directory does not exist.

User response: Verify that the indicated directory is accessible from each application and server node.

Attention: Path SVN_BIN=*svn_bin* does not exist

Explanation: The specified file path to the executable files of DCV does not exist, or perhaps is a concatenation of two or more file paths.

User response: If the command does not run, verify that the executable file can be found using the value of `$SVN_ROOT/bin` or `$SVN_BIN`.

Note: If the command runs normally, you can ignore this message.

Attention: Path SVN_SHELL=*svn_shell* does not exist

Explanation: The specified file path to the service shell scripts of DCV does not exist, or perhaps is a concatenation of two or more file paths.

User response: If the command does not run, verify that the shell script can be found using the value of `$SVN_ROOT/bin`, `$SVN_BIN`, or `$SVN_SHELL`.

Note: If the command runs normally, you can ignore this message.

Attention: SVN_MPICOMM = *svn_shell/svn_mpicomm* does not exist or is not executable

Explanation: The specified file for the SVN setup script does not exist, or is not an executable file.

User response: Verify that the MPI startup scripts reside in the path described in the message and that the scripts are executable.

Note: If the command runs normally, you can ignore this message.

Attention: Path SVN_MPILIB=*svn_mpilib* does not exist

Explanation: The specified file path to the MPI library does not exist, or perhaps is a concatenation of two or more file paths.

User response: Verify that the MPI library can be found in the path described in the message.

Note: If the command runs normally, you can ignore this message.

Attention: Path SVN_MPIBIN=*svn_mpibin* does not exist

Explanation: The specified file path to the MPI startup command (**mpirun**) does not exist, or perhaps is a concatenation of two or more file paths.

User response: Verify that the MPI startup command can be found in the path described in the message.

Note: If the command runs normally, you can ignore this message.

Error SVN: unable to create directory *svn_home* **/.dv**

Explanation: SVN creates the indicated working directory in a shared file system to store commands to the application host and to the rendering servers.

User response: Verify that the top-level directory is shared and writable by the application host.

Error SVN: unable to open file SVN_DISPLAY: *svn_display*

Explanation: SVN is unable to find the indicated wall configuration file.

User response: Correct the environment value or command-line option and retry.

Attention SVN: executable program *program* **not found**

Explanation: SVN is unable to verify that the indicated program exists or is executable.

User response: Verify that the executable program can be found using the `PATH` environment variable, or specify the executable program using a fully qualified path.

Error SVN: Executable program is mode *program_addrmode***, address mode** *addrmode* **specified**

Explanation: SVN has determined that the address mode of the executable program does not match the address mode supported on this version of Linux.

User response: Specify an executable program with the correct address mode.

Error SVN: Path SVN_LIB=*svn_lib* **does not exist**

Explanation: The file path to the SVN intercept library does not exist.

User response: Verify that DCV is installed correctly and retry.

Error SVN: DCV/SVN License agreement file not found on node *hostname*

Explanation: DCV is unable to locate the files indicating that the DCV license agreement has been accepted on this node.

User response: Install the license on this node and retry.

SVN application host (client) messages

Attention SVN: ** UNKNOWN FORMAT IN GLxxxxxxx ******
Attention SVN: ** UNKNOWN TYPE IN GLxxxxxxxx ******
Attention SVN: Unknown type in glxxxxxxxx....ignoring call

Explanation: The current version of DCV is intended to support a specific level of OpenGL, plus some extensions. (For information about the supported level of OpenGL, refer to "SVN prerequisites" on page 52.) However, not all options of every call are supported. The application has probably used an option in an OpenGL call that is not supported by SVN.

User response: Contact IBM Service.

Error SVN: XQueryTree failed
Error SVN: XGetGeometry failed
Error SVN: unable to open display
Error SVN: unable to create window
Error SVN: select error in selector

Explanation: The SVN client has encountered an internal error trying to create the selector window (`-windowselector = 1`).

User response: Verify that the `DISPLAY` environment variable is set correctly. If `DISPLAY` is correct and the message still occurs, you might be able to work around the problem by disabling the selector window. If that does not help, contact IBM Service.

Attention SVN: typeSize using default statement: type = %d We are returning 4 bytes

Explanation: An OpenGL datatype was not recognized. The application has probably used an option in an OpenGL call that is not supported by SVN.

User response: If your application does not encounter any problems, the 4 byte guess was probably good. However, if you are having trouble with your application after encountering this message, contact IBM Service.

SVN rendering server messages

Attention SVN: unresolved OpenGL op called: *OpenGL_call*

Explanation: The current version of DCV is intended to support a specific level of OpenGL, plus some extensions. (For information about the supported level of OpenGL, refer to “SVN prerequisites” on page 52.) However, not all calls are supported. The application has probably used an OpenGL call that is not supported by SVN.

User response: Contact IBM Service.

SVN: glBitmap does not support GL_UNPACK_ALIGNMENT != 1

Explanation: SVN does not support this feature.

User response: Modify the application to avoid use of this feature.

Attention SVN: X Color *color* **not found. Using white**

Explanation: This color is not available on the SVN display node.

User response: Modify the rgb.txt file to include the missing color.

SVN: No conforming visual exists on server!
==========
Visual Attributes Requested
==========

Explanation: This is a header that indicates some visual was not available in the server. The specific visual attributes will be listed.

User response: Ensure that the X server on the display node is configured with the missing capabilities.

SVN: Texture ID conflict.

Explanation: A texture with the same ID has been defined previously.

User response: Correct the application to use unique texture IDs.

Attention SVN: Unable to allocate cell for color *color*

Explanation: Be aware that the selector window colors might be affected.

User response: The selector window will continue to work.

SVN: Texture ID *texture_id* **could not be found for deletion.**

Explanation: The indicated texture_ID could not be found.

User response: Ensure the application is attempting to delete an existing texture.

Error: The SVN option *option* **is not recognized.**

Explanation: An unrecognized option was specified on the **svn_sender** command.

User response: Check the **svn_sender** command syntax and reissue the command.

Error: No SVN program name specified.

Explanation: No application name was specified on the **svn_sender** command.

User response: Rerun the **svn_sender** command and specify the application name.

Error SVN: Host addrmode *svn_addrmode* **does not match server addrmode** *srvr_addrmode* **on server** *server*

Explanation: The application host and rendering server are not running compatible versions of the operating system. One is 32-bit and one is 64-bit.

User response: Install compatible versions of Linux and retry.

Error SVN: Host version *version* **does not match server version** *srvr_version* **on server** *server*

Explanation: The application host and rendering server are not running compatible versions of DCV.

User response: Install compatible versions of DCV on the nodes and retry.

Error SVN: unable to create decode thread

Explanation: A server was unable to create one of the required service threads.

User response: Contact IBM Service.

Error SVN: unable to open wall configuration file *configfile*

Explanation: The file specified by the message cannot be opened by the server.

User response: Check that the file exists in a shared file system and that the file system is mounted on the server node.

Error SVN: not enough entries in wall configuration file *configfile*

Explanation: The **svn_sender** command counts the number of active entries in the file, and tells each server which entry to process. The server has not found the entry it is looking for.

User response: Contact IBM Service.

Error SVN: invalid X display string in wall configuration file *configfile*

Explanation: The server name in the wall configuration file should be in the form `server:d.s`.

User response: Correct the entry and retry the command.

Error SVN: unable to open display: *display*

Explanation: The indicated display is unknown or cannot be initialized.

User response: Check that the command `xhost +` has been issued on the node.

Error SVN: BAD OPCODE (nnn)!!

Explanation: The indicated number is not in the range of the numbers associated with the supported OpenGL calls. However, because the client and server use the same set of numbers, this probably means that a prior call was incorrectly processed by the server.

User response: Contact IBM Service.

Error SVN: something bad happened at line *line* **in routine** *routine*

Explanation: Internal error at indicated location.

User response: Contact IBM Service.

Attention SVN: NVIDIA framelock is not supported on the graphics hardware on *node*

Explanation: SVN is attempting to use framelock between graphic adapters to synchronize screen updates but the hardware on the specified node does not support framelock.

User response: Set the environment variable `SVN_SWAP_ON_RETRACE` to 0, or do not set it at all and allow it to default to 0.

Error SVN: Error querying MaxSwapGroupsNV on *node*

Explanation: Internal error trying to get information about the framelock capabilities of the hardware.

User response: Set the environment variable `SVN_SWAP_ON_RETRACE` to 0, or do not set it at all and allow it to default to 0. If this is not acceptable, contact IBM Service.

Error SVN: Insufficient number of swap groups *(groups)* **and/or swap barriers** *(barriers)*

Explanation: Probable internal error in trying to use the framelock capabilities of the hardware.

User response: Set the environment variable `SVN_SWAP_ON_RETRACE` to 0, or do not set it at all and allow it to default to 0. If this is not acceptable, contact IBM Service.

Appendix B. Environment variables

In addition to options that you can specify on SVN and RVN commands, and in the launcher and dashboard for RVN, you can set environment variables to further customize the processing that is done. For more information, refer to these sections:

- "Environment variables for svn_sender"
- "Environment variables for rvn_sender" on page 88
- "Environment variables for rvn_receiver" on page 92
- "Environment variables for rvn_coordinator" on page 93

Environment variables for svn_sender

In addition to the options listed in "Command-line options for svn_sender" on page 8, you can also define the following environment variables to control **svn_sender** processing.

Notes:

1. Unless you have specified the corresponding command-line option, you must define all environmental variables marked with an asterisk (*).
2. The following definitions assume that you are using the Bash shell.
3. The environment variables listed in this section should be exported.
4. If you set both the environment variable and its corresponding command-line option, the command-line option takes precedence.
5. Where applicable, the underlined value is the default.

SVN_BANNER_COLOR=*X_display_color_name*
This environment variable specifies the color used for the initial server banner display. The default is `SkyBlue`.

SVN_BANNER_FONT=*X_display_font_name*
This environment variable specifies the font used by the rendering X server to display the banner text. The default is `variable`.

SVN_BANNER_TIME=*display_time_in_seconds*
When the rendering server is started, it displays a banner showing the overall geometry and its portion of it. This environment variable controls how long the banner will be displayed. The default is 5 seconds.

SVN_BIN=*installation_binary_path*
If the SVN rendering servers are installed in a directory path other than `$SVN_ROOT/bin`, this environment variable should point to the alternate path.

SVN_CLIENT_OVERLOAD_FILE=*overload-file-path*
This environment variable identifies SVN overloads that will be loaded in the client application process. It overrides the `SVN_OVERLOAD_FILE` setting for the specified process. For more information about overloads, refer to "OpenGL overloads with SVN" on page 12.

*** SVN_DISPLAY=***wall-configuration-file-path*
The wall configuration file lists the names of the rendering servers, the virtual size of the display wall, and the portion of the geometry to be shown by each server. If this environment variable is not set, you must supply the `–svndpy` option. For more information about the wall configuration file, including display groups, refer to Chapter 2, "Scalable Visual Networking," on page 5.

SVN_DMX_OVERLOAD_FILE=*overload-file-path*
This environment variable identifies SVN overloads that will be loaded in the DMX server processes on the DMX display nodes. It overrides the `SVN_OVERLOAD_FILE` setting for the specified process. For more information about overloads, refer to “OpenGL overloads with SVN” on page 12.

SVN_FIRST_WINDOW_ONLY={0 | 1}
This environment variable provides a solution for an NVIDIA graphics driver limitation. When this variable is set to 1, it instructs SVN to send information to the rendering servers that relates only to the first window that was opened by the application. Refer to the `-firstwindowonly` command-line option for additional information.

SVN_HOME=*shared_file_home_directory*
SVN requires a shared file system directory into which the host node can write startup files and from which the rendering servers can read them. If this environment variable is not set, SVN assumes that `$HOME` is shared.

SVN_INITIAL_SCALED_WINDOW=*n*
When you set this variable, it instructs SVN to initially display the nth OpenGL window that was opened by the application, rather than the first window that was opened (which is the default behavior). The window selector can subsequently be used to change which window is shown on the display wall.

Note: If you set this environment variable, the `-firstwindowonly` command-line option and the `SVN_FIRST_WINDOW_ONLY` environment variable are ignored.

SVN_LIB=*installation_library_path*
If the SVN intercept library (`libGL.so`) is installed in a directory path other than `$SVN_ROOT/lib`, this environment variable should point to the alternate path. The **svn_sender** command assumes that 64-bit libraries, if installed and required, are in the corresponding `lib64` directory. This is the standard Linux convention. You should not supply the "64" suffix.

*** SVN_MPIBIN=***MPI_binary_path*
This variable is used to pick up the appropriate variant of the **mpirun** command for your MPI implementation. You must set this environment variable to the directory containing the appropriate MPI startup command.

Note: Setting PATH in `$HOME/.bash_profile` is not an effective substitute, because that script is not run when **rsh** is used by MPI to start task 0 on the host node.

*** SVN_MPICOMM=***SVN_startup_script_base_name*
If you do not set the `–mpicomm` command-line option, you must use this environment variable to identify the SVN startup script that selects the MPI startup option. These scripts are normally found in the same directory as the **svn_sender** command.

Note: Refer to the `–mpicomm` description for a list of supported MPI options.

*** SVN_MPILIB=***MPI_library_path*
This environment variable is added to the `LD_LIBRARY_PATH` environment variable on all nodes. It is used to pick up the shared MPI library (`libmpich.so`) on each node. You must set this variable to the directory containing the MPI library.

Notes:

1. Setting `LD_LIBRARY_PATH` in `$HOME/.bash_profile` is not an effective substitute, because that script is not run when **rsh** is used by MPI to start jobs on the rendering servers.
2. If `libmpich.so` is not in the same directory as the other shared objects required for MPI implementation, this environment variable can provide a search path.

SVN_OVERLOAD_FILE=*overload-file-path*
This environment variable identifies a set of overloads that will be loaded into all SVN processes, on both the application host and the rendering server. For more information about overloads, refer to “OpenGL overloads with SVN” on page 12.

SVN_ROOT=*installation_path*
If the SVN component is installed in a directory other than `/opt/IBM/dcv/svn`, this environment variable should point to the alternate path.

SVN_SERVER_OVERLOAD_FILE=*overload-file-path*
This environment variable identifies SVN overloads that will be loaded in the SVN server processes. It overrides the `SVN_OVERLOAD_FILE` setting for the specified process. For more information about overloads, refer to “OpenGL overloads with SVN” on page 12.

SVN_SHELL=*installation_SVN_shell_path*
If the SVN startup scripts are installed in a directory other than `$SVN_BIN`, this environment variable should point to the alternate path.

SVN_SVNRVN_OVERLOAD_FILE=*overload-file-path*
This environment variable identifies SVN overloads that will be loaded in the SVNRVN server process. It overrides the `SVN_OVERLOAD_FILE` setting for the specified process. For more information about overloads, refer to “OpenGL overloads with SVN” on page 12.

SVN_SWAP_ON_RETRACE ={ 0 | 1 }
This environment variable delays synchronization until the displays retrace on the server.

Note: If you are using NVIDIA graphics G Series adapters, setting `SVN_SWAP_ON_RETRACE` to `1` enables the framelock function.

SVN_VERBOSE={ 0 | 1 }
This environment variable performs the following functions:

- Turns on echoing of each line in the **svn_sender** script for debugging purposes
- Saves intermediate work files for analysis and problem determination

SVN_WINDOW_SELECTOR={0 | 1}
This environment variable sets the default value for the window selector function described in the `–windowselector` command option. This variable can be overridden by the value set for the `-windowselector` command-line option.

Environment variables for rvn_sender

In addition to the options listed in "The RVN application launcher" on page 28 and in "Command-line options for rvn_sender" on page 37, you can also define the following environment variables to control **rvn_sender** processing.

Two environment variables are commonly used with **rvn_sender**:

- `DISPLAY`
- `RVN_USE_VNC=1`

If you are using the RVN application launcher under Windows or Linux, you generally do not need to set these two environment variables yourself. The launcher sets them for you. However, if you are using the `X11` export mechanism, you must set the `DISPLAY` variable before starting the launcher. For more information, refer to "The RVN application launcher" on page 28.

If you are running the **rvn_sender** command and bypassing the launcher, set these two variables before invoking **rvn_sender**.

- Set the `DISPLAY` variable to correspond to the X server on which the 2-D graphical application interface should be displayed.
  - If you are using VNC, the 2-D interface is technically being displayed on the application host, and so the `DISPLAY` should be set to `:0`.
  - Linux only: If you are using the `X11` export mechanism (refer to Figure 4 on page 24), you should set the `DISPLAY` to the X server running on the receiver (`receiver:0`).
- If you are using VNC, set `RVN_USE_VNC=1`. Both the sender and receiver require this information to define how pixels are written to the application window.

  Note: You do not need to set the `RVN_USE_VNC=1` variable on the receiver. The sender passes the setting information to the receiver upon connecting.

Here is the complete list of environment variables that you can set for the RVN sender.

RVN_ALTERNATE_VISUALS=1
Set this environment variable only if **rvn_sender** fails when this variable is not set.

Note: This environment variable is not normally used, and it might be required only when you are using an X server with limited function. It asks for a 24-bit true-color non-stereo visual, irrespective of what the application tried to request.

RVN_CONFERENCE_ID=*confID*
(X11 export mode only) Specifies the name of the RVN conference session.

Notes:

1. This information is needed only for X11 export mode. When you are using VNC, this information is automatically generated during VNC authentication.
2. This information must be conveyed to each conference participant before the start of the conference.

RVN_CONFERENCE_KEY=*confkey*
(X11 export mode only) Specifies the key required to access the specified RVN conference session.

Notes:

1. This information is needed only for X11 export mode. When you are using VNC, this information is automatically generated during VNC authentication.
2. This information must be conveyed to each conference participant before the start of the conference.

RVN_COORDINATOR_PORT=*port*
Specifies the port on the application host used by `rvn_sender`, `rvn_receiver`, and `rvn_coordinator` to establish an RVN conference session. This environment variable must have the same value for all three processes (`rvn_sender`, `rvn_receiver`, and `rvn_coordinator`) participating in the RVN conference.

Notes:

1. The default port is 7200.
2. This information must be conveyed to each conference participant before the start of the conference.

RVN_DASHBOARD_DISPLAY=*display*
(Linux only) Causes the RVN dashboard to open on the specified display.

RVN_EXTERNAL_TRANSPORT=[1|0]
When this environment variable is set to 1, the RVN pixel backchannel is enabled, and OpenGL output is transferred to the viewer much more quickly. RVN will write data to a local window and not send it out across an accelerated graphics link.

Note: The default value for this environment variable is 0 (RVN does not rely on the pixel backchannel).

RVN_HOST_SHOW_PIXELS=[1|0]
Enables or disables displaying pixels on the local display of the application server in addition to sending it to the RVN end station.

Note: The default value for this environment variable is 0 (RVN does not write the image to the local display).

RVN_IMAGE_QUALITY=[1...100]
Controls the balance between image compression and image quality when the image is changing. The range is a percent from 1 to 100. Choosing a high quality value (a large number) improves the visual quality of the dynamic images, but requires more data to be sent for each image, which can reduce responsiveness or lower the frame rate. If you select 100% compression, lossless compression will be used.

Note: The default value for this environment variable is 80 (100 is the highest quality).

RVN_INTERACT_MODE=[1|0]
Specifies the mode for sending frames from the application to the end-station display.

- If you specify 1 (interactive mode), RVN attempts to allow the application to run as fast as possible (with the least RVN delay). It always transmits the most recently generated frame to the end-station display. In doing so, it might drop stale images before trying to compress and send them, in

favor of sending newer images. Select this mode if you are most interested in always viewing the most recent frame, and not concerned with seeing every generated frame.

- If you specify 0 (pacing mode), RVN sends every frame that the application generates and, in so doing, can often significantly slow the application. Pacing mode allows the users at the end stations to see all of the rendered data. It also addresses the issue of frame-rate jitter, where image frames are not displayed at regular intervals. This variability in the frame-to-frame interval is typically the result of network jitter or a variability in computation loads on the sender or viewer. To "smooth out" the jitter, paced mode introduces interval delays, causing the frames to be displayed at regular intervals. This creates a regular paced viewing experience.

Note: The default value for this environment variable is 1.

RVN_LOCAL_DISPLAY=*display*
The default hardware rendering adapter is :0. Use this environment variable to specify non-default hardware.

RVN_PACING_TIME=[0...1000]
Specifies the length of delay to be used for pacing mode, which is selected by setting `RVN_INTERACT_MODE=0`. The time is specified in milliseconds. Increase the value until the frame rate becomes sufficiently smooth and pleasing to the eye.

Note: The default value for this environment variable is 0.

RVN_QUALITY_UPDATE=[1|0]
Sends a higher quality image when the image is not being rapidly updated by the application. When this option is selected, RVN rapidly transfers lower-quality images when the image is changing, then updates the image with a high-quality update, thereby enhancing the quality of the static image.

Notes:

1. The default value for this environment variable is 0.
2. In some circumstances, rapid flickering occurs when `RVN_QUALITY_UPDATE` is enabled. To avoid this condition, do not enable this variable.

RVN_SUBSAMPLING=[1|2|4]
Set this environment variable to adjust image quality through pixel subsampling.

- Specify `1` for 411 subsampling, which uses every other x and every other y pixel. 411 subsampling creates a compressed image by reducing the number of bits to approximately half, and so is likely to perceptibly decrease image quality.
- Specify `2` for 422 subsampling, which uses every other pixel in x. 422 subsampling generally provides sufficient quality for human perception.
- Specify `4` for 444 subsampling, which uses all pixels. 444 subsampling approximately doubles the data requirements.

Note: The default value for this variable is 2.

RVN_SUPPRESS_DASHBOARD=[1|0]
Set this environment variable if you do not want the RVN dashboard to be displayed.

Note: The default value for this variable is 0 (RVN does not prevent the dashboard from being enabled).

RVN_SYSTEM_OPENGL_LIB=*complete_path_to_system_openGL_library*
Specifies the path to the `System_OpenGL` library.

RVN_UDP=[0|1|2]
Determines how pixels are sent when latency is high. Specifying 1 or 2 for the `RVN_UDP` environment variable provides a more responsive view when interactive mode is being used (`RVN_INTERACT_MODE` is set to 1) and a sequence of dynamic frames is being sent over a low bandwidth. Specify one of the following options:

0 Indicates that the low bandwidth option will not be used.

1 Indicates that the low bandwidth option will be used.

2 Allows RVN to automatically determine whether to use the low bandwidth option, based on throughput, latency, and packet-loss threshold.

Note: The default value for this environment variable is `2`.

RVN_UDP_QUALITY=[1...100]
Adjusts the image quality when `RVN_UDP` is set to `ON` or `AUTO`. Because fewer frames are transmitted when a low bandwidth is used, you might want to reduce the quality of the image to increase the compression and, therefore, speed. Low values mean high compression and poor image quality; high values mean little compression and high image quality.

Note: The default value for this environment variable is 10.

RVN_UDP_SHOW_PARTIAL=[1|0]
Indicates whether to display partially updated frames. Set this variable to 1 to see a "frameless" display in the event of network losses, and to use old data to fill occasional gaps in the image when some data segments are lost.

Note: The default value for this environment variable is 1.

RVN_UNIQIFY_CONFERENCE_ID=[1|0]
(X11 export mode only) Set this environment variable to 1 if you are running an OpenGL application that forks multiple processes that each load the OpenGL library, libGL.so. This ensures that each application process that is forked after the initial application process is assigned a unique conference ID, based on the conference ID for the session followed by a sequential number. For example, if the specified conference ID is `abc123`, the initial application process will use that conference ID, the second application process will use the conference ID `abc123_1`, the third will use `abc123_2`, and so on.

Notes:

1. This variable is relevant only for sessions that use X11 export. It is not needed for sessions that use VNC.
2. The default value for this variable is 0.

RVN_UPDATE_QUALITY=[1...100]
Controls the balance between image compression and image quality when the image is not changing. The range is a percent from the current dynamic compression quality to 100. If you select 100% compression, lossless compression will be used.

Note: The default value for this environment variable is 95 (100 is the highest quality).

RVN_USE_VNC=[1|0]
When set to 1, tells RVN to look for a VNC window when writing application images.

Notes:

1. On a Linux application server, the default value for this environment variable is 0 (RVN does not look for a VNC window, and uses X11 export mode instead).
2. On a Windows application server, this environment variable must be set to 1.

Environment variables for rvn_receiver

One primary environment variable is used with the **rvn_receiver** command:

- DISPLAY

The DISPLAY environment variable is used in the standard method for X applications. In most cases, you should set the *sender* variable to DISPLAY=":0" before calling rvn_receiver. In addition, it is also necessary to ensure that you set the X access control to allow the required connections. For example, when using RVN in X11 export mode, the sender must have access to the X server running on the receiver.

Here is the complete list of environment variables that you can set to control **rvn_receiver** processing.

DISPLAY=*standard_X_DISPLAY_syntax*
Describes where 2-D receiver-side images will be displayed.

RVN_CONFERENCE_ID=*confID*
Specifies the name of the conference session.

Note: If RVN_CONFERENCE_ID and RVN_CONFERENCE_KEY are not specified, default values are used. As a result, the connection is not secure.

RVN_CONFERENCE_KEY=*confkey*
Specifies the key required to access the specified conference session.

Note: If RVN_CONFERENCE_ID and RVN_CONFERENCE_KEY are not specified, default values are used. As a result, the connection is not secure.

RVN_COORDINATOR_PORT=*port*
Specifies the port used by rvn_sender, rvn_receiver, and rvn_coordinator to establish an RVN session. This environment variable must have the same value for all three processes (rvn_sender, rvn_receiver, and rvn_coordinator) participating in the RVN conference.

Note: If RVN_COORDINATOR_PORT is not specified, the default port is 7200.

RVN_VIEWER_TITLE=*window_title*
Causes the RVN receiver to look for a specific window title (for example, a vncviewer) as the destination window. By default, if you set `RVN_USE_VNC` on the sender and do not set both `RVN_VIEWER_TITLE` and `WINDOWID`, this environment variable uses some known vncviewer such as `VNC: x11`.

RVN_VIEWER_WINDOWID=*window_ID*
Causes the RVN receiver to use a specific X-window ID as the pixel destination.

Note: The default value is none.

Environment variables for rvn_coordinator

You can set the following environment variables to control **rvn_coordinator** processing.

RVN_COORDINATOR_PORT=*port*
Specifies the port used by `rvn_sender`, `rvn_receiver`, and `rvn_coordinator` to establish an RVN session. This environment variable must have the same value for all three processes (`rvn_sender`, `rvn_receiver`, and `rvn_coordinator`) participating in the RVN conference.

Notes:

1. The default port is 7200.
2. This information must be conveyed to each conference participant before the start of the conference.

RVN_TOP_PORT=*top-port*
Specifies the highest port number used by `rvn_coordinator` when allocating ports for use in an RVN conference. This value must be higher than the value of `RVN_COORDINATOR_PORT`.

Note: The default value is 7220.

Appendix C. Programming considerations for SVN and RVN

You must consider the following information when you are programming applications that will run under SVN and RVN:

Front-buffer programs

If a program renders to the front buffer, graphics commands might be delayed reaching the SVN rendering servers. This delay occurs while the SVN client fills its output buffers with rendering commands. SVN requires that an explicit or implicit flush occur in order to guarantee that the SVN rendering servers are updated on a timely basis. OpenGL commands that explicitly or implicitly cause a flush are:

- **glFlush()**
- **glXSwapBuffers()**
- **glFinish()**
- **glXMakeCurrent()**

Offscreen buffer and pixmap rendering programs

Programs that render to an offscreen buffer and then retrieve the rendered data might not work correctly. The same is true of programs that render pixmaps.

Non-OpenGL programs

Programs that do not use OpenGL to produce their graphic images will not work correctly in SVN or RVN.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept LJEB/P905
2455 South Rd.
Poughkeepsie, NY 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States or other countries or both:

- IBM
- IBMLink™
- IntelliStation
- System x
- xSeries

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product and service names may be the trademarks or service marks of others.

Glossary

B

backchannel. A second data path between the application host and an end station, which uses data compression to efficiently transport pixel images.

Bash shell (Borne Again SHell). A command interpreter based on the Borne shell from AT®&T that is commonly used in Linux to run command scripts.

C

coincident. Occurring or operating at the same time (or at the same location).

convenience script. A command script that encapsulates several commands to avoid having to type them individually.

D

display wall. A large display surface, typically hung from or installed in a wall. Typically, display walls are assembled from individual components, each of which shows a portion of the picture. For example, a display wall might be a screen on which several projectors are focused, each projector pointing to a different portion of the screen. It is usually convenient to organize a display wall as a set of tiles, which might overlap slightly to avoid the visual impact of a sharp edge between them.

E

end station. A node and associated display which is used to actually display graphics.

F

framelock. The ability of the several graphic adapters driving a display wall or multiadapter monitor to synchronize their screen updates so that all portions of the display change simultaneously. This is usually accomplished by having one of the adapters send the update signal to cause all the adapters to update their screens.

G

geometry coverage. The location and dimensions of a portion of a three-dimensional graphic display.

I

InfiniBand. A specification and products for communication adapters and switches for high-performance communication within a limited area. The specification includes a description and standards for software interfaces as well as for hardware.

intercept library. A library that is loaded as an alternate to the shared library intended by the application programmer. The intercept library provides the same functions to the calling program, but often performs additional functions, such as capturing debugging information, or routing data to alternate locations.

M

Message Passing Interface (MPI). An industry-standard application programming interface for message exchange between two or more cooperating tasks. Widely used in high-performance and scientific computing applications.

MPI. See *Message Passing Interface*.

O

OpenGL. An application programming interface for representing three-dimensional objects and models. Originally developed by Silicon Graphics, Inc. (now SGI, Inc.), it has become a de facto standard for high-performance, three-dimensional graphics programming.

P

pass-phrase. An extension of the concept of password, but allowing a phrase, rather than a single word, to be handled. Pass-phrase is used and recommended by Secure Shell.

port forwarding. See *tunneling*.

R

remote desktopping. A software feature whereby a display's entire desktop is shown and can be manipulated on a remote display.

remote visual networking (RVN). A component of Deep Computing Visualization (DCV) in which the image of a three-dimensional graphic object is efficiently transmitted to a remote display running on an end station.

render. To create an image or visual display from data that describes the scene.

rendering server. A computer with a display which receives a set of graphic elements and a description of the desired geometry coverage, and creates (renders) a corresponding image.

RVN. See *remote visual networking*.

S

scalable visual networking (SVN). A component of Deep Computing Visualization (DCV) in which the elements of a three-dimensional graphic object are broadcast to a cluster of displays, such as a display wall.

scene geometry. The location, dimensions and orientation of all the objects being represented by a three-dimensional application.

Secure Shell (OpenSSH). A method of authenticating and encrypting data between two computing systems that provides a high level of security. OpenSSH is a version of the Secure Shell (SSH) protocol that is distributed with Linux systems.

shuffle algorithm. An algorithm for broadcasting data in which the data buffer is broken into pieces by the originator of the broadcast, and each piece is given first to a different member of the broadcast recipients, who then have the responsibility to exchange the pieces with each other until each recipient has the entire message.

single program, multiple data (SPMD). A form of parallel computing in which several copies of the same program are started simultaneously, but each copy works with different data, and might be executing different instructions.

SPMD. See *single program, multiple data*.

SVN. See *scalable visual networking*.

T

tar. Acronym for tape archive. A method for compressing and packing files and directories so that they can be moved to another computing system.

tarred. A file that is in tar format.

tile (n.). One of several non-overlapping, rectangular divisions of a display screen.

tile (v.). To arrange multiple windows so that they appear side by side and top to bottom.

tunneling. A technique provided by SSH which allows secure access to a remote computer by a variety of applications. Also called *port forwarding*.

U

untar. To decompress files, similar to unzipping in Microsoft Windows environments.

V

virtualize. To provide the appearance of a function while actually performing the function in a different placed or time.

visualization. The act or process of putting into or interpreting in visual terms or in visual form.

W

wall configuration file. A computer file that describes the components of a display wall. An entry in the wall configuration file identifies the component (computer) responsible for updating a particular tile of the display, and tells that component which portion it is responsible for.

wall display. See *display wall*.

Index

R

S

T

U

V

W

X

Reader's comments – We'd like to hear from you

Deep Computing Visualization
Installation and User Guide
Version 1 Release 3

Publication No. G224-9183-02

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	E-mail address

Readers' Comments — We'd Like to Hear from You
G224-9183-02

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P181
2455 South Road
Poughkeepsie NY 12601-5400

Fold and Tape | **Please do not staple** | Fold and Tape

G224-9183-02

Cut or Fold
Along Line

Printed in USA

G224-9183-02