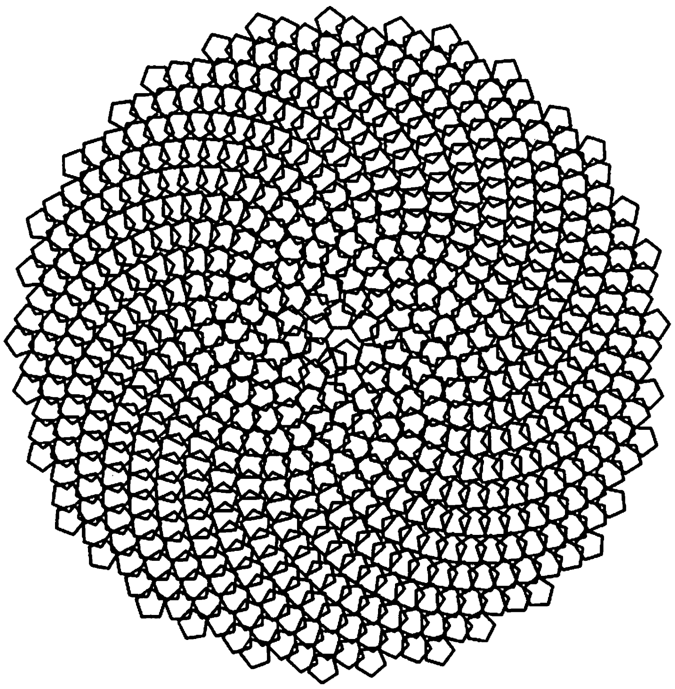
GDDM

Performance Guide

IBM



Front Cover Pattern: Electronic Sunflower

The pattern on the front cover was produced by a GDDM program. The program to produce this pattern, and many variations of the pattern, is published in:

- *GDDM Application Programming Guide*
- *GDDM Base Programming Reference*

SC33-0324-0
File No. S370/4300-34

GDDM

Performance Guide

Program Numbers
GDDM/MVS 5665-356
GDDM/VM 5664-200
GDDM/VSE 5666-328
GDDM/Interactive Map Definition 5668-801
GDDM-PGF 5668-812

Version 2 Release 1

Licensed Programs

First Edition (September 1986)

This edition applies to Version 2, Release 1, Modification 0 of the following members of the IBM GDDM Series of licensed programs:

GDDM/MVS 5665-356
GDDM/VM 5664-200
GDDM/VSE 5666-328
GDDM Interactive Map Definition 5668-801
GDDM-PGF 5668-812

Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Preface

What this book is about

This book provides information about performance aspects of Version 2 Release 1 of GDDM, the Graphical Data Display Manager. It will help you understand the background to GDDM performance, and show you how to run GDDM in the most efficient way.

Who this book is for

This book is for:

System designers and programmers

Application designers and programmers.

What you need to know

Some of the chapters of this book require familiarity with the subsystem under which the application programs are to run. For example:

CICS/VS

IMS/VS

TSO

VM/SP.

Other chapters require some experience of computer applications.

How to use this book

You can read this book sequentially, or just read the chapters that concern you. An overview of the book structure is shown on page viii and detailed in the table of contents. You can use the index at the back of the book for reference.

Bibliography

GDDM library

Introduction

- General Information
  GBOF-0058*
- Release Guide
  GC33-0320

*Includes the GDDM brochures.
For the General Information manual only, use order number GC33-0319

General

- Installation and System Management for MVS
  SC33-0321
- Installation and System Management for VM
  SC33-0323
- Installation and System Management for VSE
  SC33-0322
- Performance Guide
  SC33-0324
  this book
- Messages
  SC33-0325
- Diagnosis and Problem Determination Guide
  SC33-0326

Programming

- Application Programming Guide
  (Two volumes)
  SC33-0337
- Base Programming Reference
  (Two volumes)
  SC33-0332
- GDDM-PGF Programming Reference
  SC33-0333
- Base Programming Summary (Booklet)
  SX33-6053
- GDDM-PGF Programming Summary (Booklet)
  SX33-6054

User's Guides

- Guide for Users
  SC33-0327
- Interactive Chart Utility (ICU)
  SC33-0328
- Image Symbol Editor
  SC33-0329
- Vector Symbol Editor
  SC33-0330
- Interactive Map Definition (GDDM-IMD)
  SC33-0338

Books from related libraries

In addition to the GDDM library, you may need to refer to some of the following manuals:

APL

VS APL for CMS: Terminal User's Guide, SH20-9067
APL2 Installation and Customization under CMS, SH20-9221.

CICS

CICS/VS Version 1 Release 6 Resource Definition Guide, SC33-0149
CICS/VS Version 1 Release 6 IBM 3270/8775 Guide , SC33-0096
CICS/OS/VS Version 1 Release 6 Modification 1 Installation and Operations Guide, SC33-0172
CICS/DOS/VS Version 1 Release 6 Installation and Operations Guide, SC33-0070
VS APL for CICS/VS: Terminal User's Guide, SH20-9167.

IMS/VS

IMS/VS Installation Guide, SH20-9081
IMS/VS Messages and Codes Reference Manual, SH20-9030
IMS/VS Utilities Reference Manual, SH20-9029.

MVS

OS/VS2 MVS Initialization and Tuning Guide, GC28-0681
OS/VS2 MVS Performance Notebook, GC28-0886.

SMP/E (SMP)

OS/VS SMP System Programmer's Guide, GC28-0673
SMP/E Terminal User's Guide, SC28-1109
SMP/E User's Guide, SC28-1302.

TSO

VS APL for TSO: Terminal User's Guide, SH20-9180
APL2 Installation and Customization under TSO, SH20-9222
JES/328X Print Facility — Version 2 Program Offering, Program Description and Operator's Manual, SB11-6444.

VM

VM/SP Installation Guide, SC24-5237
VM/SP Planning Guide and Reference, SC19-6201
VM/SP Operator's Guide, SC19-6202
VM/SP System Programmer's Guide, SC19-6203
VM/VCNA General Information, GC27-0501
VM/VCNA Installation, Operation and Terminal Use, SC27-0502.

VSE

VSE Advanced Function: MSHP Reference, SC33-6199
VSE Advanced Function System Control Statements, SC33-6198
VSE/SP Installation, SC33-6178.

3179-G

3179-G Color Graphics Display Station Description, GA18-2261.

3270-Family Devices

3270 Information Display System Configurator, GA27-2849
3270 Information Display System Data Stream Programmer's Reference, GA23-0059
8775 Display Terminal: Component Description, GA33-3044.

3270-PC/G and 3270-PC/GX Work Stations

Introducing the IBM 3270 Personal Computer/G and /GX Ranges of Work Stations, GA33-3157
3270-PC/G Personal Computer/G and /GX Ranges of Work Stations; Planning Guide, GA33-3158
3270-PC/G Guide to Operations, GA33-3140
3270-PC/GX Guide to Operations, GA33-3139
Graphics Control Program User's Guide, (supplied with program).

3274

3274 Control Unit Description and Programmer's Guide, GA23-0061
3274 Control Unit Planning, Setup and Customization Guide, GA23-2827.

Image Devices

4224 Printer
  Printer Product and Programming Description Manual, GC31-2551
  Operating Instructions, GC31-2546
  Guide to Operations, GC31-3621.

3117 Scanner
  IBM 3117 Scanner and IBM 3117 PC Adapter Guide to Operations, GA18-2477
  IBM 3117 Scanner and Extension Unit Guide to Operations, GA18-2478
  IBM 3117 Scanner Hardware Maintenance and Service, SY18-2159
  IBM 3117 Scanner Technical Reference, SC18-2105.
3118 Scanner
  Scanner Guide to Operations, GA18-2475
  High Speed Adapter Guide to Operations, GA18-2476
  IBM 3118 Scanner Hardware Maintenance and Service, SY18-2158
  High Speed Adapter Hardware Maintenance and Service, SY18-2167
  Scanner Technical Reference, SC18-2104
  High Speed Adapter Technical Reference, SC18-2117.

3193 Display station
  Description, GA18-2364

Setup Instruction, GA18-2366
Operator's Guide, GA18-2365
Problem Solving Quick Check Guide, GA18-2443
Problem Solving Guide, GA18-2444.

Other printers

4234 Printer (supported in alphanumeric mode only)
Operation Instructions for Model 1, GC31-2556.

5550 Multistation

(Available in Japanese only)

5550 Japanese 3270-PC User's Guide, N:SC18-2059
How To Use 5550 Japanese 3270-PC, N:SC18-2060
5550 Japanese 3270-PC/G User's Guide, N:SC18-2071
How To Use 5550 Japanese 3270-PC/G, N:SC18-2072
5550 Small Cluster User's Guide, N:SC18-2092
How To Use 5550 Small Cluster, N:SC18-2091
5550 Small Cluster/Graphics User's Guide, N:SC18-2107
How To Use 5550 Small Cluster/Graphics, N:SC18-2108
5550 3270 Kanji Emulation Description, N:SC18-2020
5550 3270 Kanji Emulation Operator's Guide, N:SC18-2021.

GDDM/graPHIGS manuals

Installing GDDM/graPHIGS, SC33-8101
Licensed Program Specifications, GH23-0001
Introducing graPHIGS, SC33-8100
Understanding graPHIGS, SC33-8102
Writing Applications with graPHIGS, SC33-8103
Programmer's Reference for graPHIGS, SC33-8104
Messages and Error Codes for graPHIGS, SC33-8105
Programmer's Pocket Reference for graPHIGS, SC33-8107
Problem Diagnosis for graPHIGS, SC33-8108.

Networking

Network Program Products Samples: VM SNA, SC30-3309.

Book structure

Contents

Chapter 1. Performance background

This chapter explains how GDDM works, and what the performance implications are. It should give you sufficient background to help you make informed judgments about the subsystem-dependent customizing suggestions discussed in Chapter 2.

What governs GDDM resource usage

Two major factors influence the amount of system resources that GDDM requires to display a picture on a graphics device or an image device.

1. The type of device

   When GDDM produces a picture containing graphics for printing on a 4250 composed-page printer, it may require ten times or more host-processor resource than when it displays the same picture on a 7372 plotter. Similarly, the 3279 picture data stream generated by GDDM may be two to five times as large as that for a 3270-PC/G. Graphics hardware has a significant influence on the resource requirements of GDDM. We will examine this in more detail later.

   Similarly, the host processor and data stream requirements of a picture containing an image will be different for an image device and a non-image device, because image devices can perform several image processing tasks that have to be carried out by the host for non-image devices.

2. Picture complexity

   For graphics, we can make a start by saying that picture complexity is related to the number of elements (lines, arcs, areas, and so on) that define the picture. Clearly, GDDM is going to do more work to draw six lines than it is to draw one.

   For images, complexity is related to the total number of pixels in the picture, how many are on or off, and how often they alternate between the two states.

   Complexity is also related to the device on which the picture is displayed. Graphics pictures that appear fairly modest to GDDM when it is producing them for a 3287 may well appear very complex when the device is a plotter. The resource relationships mentioned earlier apply to specific pictures and will vary considerably.

It is well worth spending some time understanding how the GDDM-supported graphics devices and image devices work.

Figures

GDDM hardware – how it works

A device can work in one of two ways to produce graphics or image. These are known as the **raster-scan** and **random-scan** technologies.

For the raster-scan devices, the picture area is defined as a series of regularly spaced dots. The drawing implement (like an electron beam or the print head of a matrix printer) moves along each row, marking those dots that form the picture.

For random-scan devices (plotters) the path of the pen that actually draws the picture moves along the lines that make up the picture.

All the devices that GDDM supports rely on one of those two basic technologies to produce graphics or images. These technologies in themselves have implications for GDDM performance, but there are other device characteristics, the prime one being device programmability, which play an even larger role.

In terms of performance you can consider GDDM devices as belonging to one of six classes:

- Order-driven devices – programmable
- Order-driven devices – nonprogrammable with random scan
- Order-driven devices – nonprogrammable with raster scan
- Image devices
- Character-driven devices
- Composed-page printers.

These six classes comprise all the devices that GDDM supports. However, no two devices are exactly alike. Devices in different classes may have similar characteristics, but each class has its own unique hardware features that govern GDDM's picture display resource requirements.

Generally speaking, the device classes, as listed above, reflect the programmable capacity that is in the device itself. This reflects the resource that GDDM uses to display pictures. Programmable order-driven devices require least, composed-page devices most.

Of the six classes, only the nonprogrammable order-driven devices use the random-scan technique to produce pictures. All the others use raster-scan technology.

Order-driven devices

All GDDM graphics calls that a program issues eventually get translated to commands like "draw a line from A to B" or "future items will be colored blue." The order-driven devices obey just such orders. All GDDM has to do is encode the program calls into the language that the device recognizes and to send them.

At the device, the orders are decoded and the drawing is made.

Programmable devices

The 3270-PC/G and /GX family (including the AT versions), the 5550 family, and the 5080 graphics system are the programmable order-driven devices that GDDM supports. One of the key features of these devices is that the work of activating the dots that represent the picture is done in the terminal itself. We call that process **vector-to-raster conversion**. Besides converting lines or vectors to dots, the devices can also do other things like fill areas.

GDDM has two modes of operation for communicating with the 3270-PC/G and 3270-PC/GX work stations. These modes are known as **retained** and **non-retained** modes.

In retained mode the vector definition of the picture is written to **segment storage** as the terminal starts the vector-to-raster process onto an **all-points-addressable** (APA) bit map of the screen. In non-retained mode, each vector is drawn on the bit map as it is received and then discarded.

You can see that in non-retained mode it is possible to display very complex pictures, regardless of the number of vector definitions that they contain. The 5550 family supports non-retained mode only.

Retained mode has important implications. We will look at these now.

Besides being able to do the vector-to-raster conversion itself, the 3270-PC/G and /GX can perform other tasks. This gives them distinct performance advantages over other devices that GDDM supports.

Work-station features that GDDM takes advantage of include:

- Segment manipulation

  The work stations can perform certain actions on pieces of a picture. These pieces of the picture are properly known as **segments** and are defined by the graphics application. The application program or work-station user can impose meaning on a picture by ensuring that each segment contains logically connected elements of the picture. For example, in a picture of a nut and bolt it would make sense to put all the lines defining the nut into one segment and all the lines defining the bolt into another. The two objects could then be manipulated separately.

Work-station functions performed on segments include:

- Dragging

  The ability to move a segment around the screen under graphics cursor control. Its new position can be signalled back to the host.

- Picture update

  Specific segments can be deleted by sending a "delete segment" order to the work station. When adding a segment to a picture, only the new segment has to be sent, not the entire picture. Similarly, highlighting and visibility changes can be made by sending a small amount of data.

- Character expansion

  Besides containing things like lines, arcs and areas, many pictures also contain graphics characters. These can vary in size; they can be angled or sheared and can be in different styles.

  The work station contains a set of graphics character definitions, more properly known as a **symbol set** or **font**. When a picture is sent from the host, it contains references to characters in the symbol set. The work station will take the required character definitions from the set, perform any scale, shear, or rotation necessary and add them to the picture definition.

  Besides the default fonts, GDDM supplies other fonts as image or vector symbol sets. If these fonts are not to your liking, you can define your own using the GDDM Image Symbol Editor or Vector Symbol Editor.

  (For the 5550 family, default single-byte character set (SBCS) and double-byte character set (DBCS) vector symbol sets are supported by the Japanese 3270 PC/G program, version 6.0.)

  If you use any of the non-default symbol sets, GDDM can send them to the work station before sending the picture. They will then get treated in exactly the same way as the default sets.

- Panning and zooming

  Under user control, the work stations can locally scroll pictures up and down or left and right (panning), or enlarge pictures (zooming). (For other graphics displays, and for the work stations if the local capability is not activated, GDDM performs the panning and zooming in the host.) This avoids the necessity of sending a new picture for every new viewing operation. See 17 for more details.

- Clipping

  Those areas of a picture that lie outside the graphics field are not displayed. On devices like the 3270-PC/G and /GX, the terminal **clips** the picture to the visible area. When the page window is changed, or the operator changes the page window, the picture does not have to be resent. On the 5550 terminal, the picture is clipped to the **segment viewing limits**.

All the above functions depend on the ability of the work station to hold the vector definition of the picture, and the fonts associated with it, in segment storage in the work station. Segment storage is allocated to host sessions at work-station customization time. Chapter 3, "Storage requirements and capacity planning" on page 61 gives guidance on how much you will need to allocate to your host session for GDDM to take advantage of the work-station functions described above.

Nonprogrammable devices

IBM 3179 Color Display Models G1 and G2 (3179-G): In many ways, the 3179-G is equivalent to the 3270-PC/G and /GX in non-retained mode with no segment storage. Like 3270-PC/G and /GX, GDDM sends the vector definitions to the 3179-G, and the work of activating the pixels that represent the picture (the vector-to-raster conversion) is done in the terminal itself.

The 3179-G, however, has only one default vector-symbol set and has only four image symbol sets.

It also has no local clipping capability, so GDDM may have to do extra clipping in the host, for example, when a page is scrolled or an operator window is sized.

Arcs are split into short vectors, that are sent to the 3179-G to be drawn. (This is also true of the 3270-PC/G and /GX when used with the LCLMODE,NO processing option.)

The 3179-G does not store graphic data, and so cannot offload any manipulation function from GDDM. In particular, with user control, each new viewing operation means that the data has to be regenerated.

4224 printers: These dot-matrix printers are also in the nonprogrammable device category. The vector-to-raster conversion process is performed in the printer, and so a 4224 printer generally requires less host-processor resource than, for example, a 3287 printer.

Because the paper moves through the printer in only one direction, all the GDDM data that describes the picture must be sent to the printer before the rastering process can begin. 4224 printers therefore have their own storage to contain the data. Models 101 and 102 have 64K bytes, of which 35K bytes is available for picture data. Models 1E2 and 1C2 have 512K bytes, of which 448K bytes is available for picture data.

Alphanumeric and image data are printed on a row-by-row basis, from the top to the bottom of the page. About 5K bytes of the available storage is needed for processing these data. The remainder can hold the graphics data and, for models 1C2 and 1E2 up to six alphanumeric symbol sets.

The 4224 has no resident vector symbol sets, so all graphics text referencing such symbol sets must be expanded into its constituent lines and arcs by GDDM.

GDDM does not send arcs to the device. They are split into short vectors, resulting in more host processing and longer data streams.

The size of the picture data that GDDM sends to the 4224 is increased when stored in the printer. This is because printer specific information is associated with each primitive received. The effect of this is to reduce the amount of available storage for GDDM data, and, for the 64K byte models in particular, you should be careful of the nature of pictures that you send.

GDDM pre-calculates whether all the picture data will fit in the available storage. If an overflow would occur, GDDM issues a warning message, and sends to the printer only that amount of data that can be stored without causing an overflow. You can use the GDDM call FSCHEK in your program to find out whether a picture will be too complex for the printer.

Considering GDDM's use of the printer to support graphics data in isolation, the printer-dependent data appended to each drawing order significantly increases the storage requirements for a GDDM picture – typically by a factor of 2 to 3.

Plotters: Also in the nonprogrammable order-driven class are the 737x and 6180 plotters. **Reverse clipping**, which is described here, applies only to GDDM support of these plotters.

They are similar to the programmable devices in that they can translate orders into pictures. However, the set of orders that they can draw is limited:

- Area fill is done by sending vector definitions of each of the lines that make up the shading pattern.
- Arcs are drawn as several very short vectors. Each vector is drawn by moving the pen by no less than half a pen width.
- Symbol sets are expanded into vectors in the host.

The other main feature of these devices is that, as they are basically moving a drawing implement around, anything that gets drawn stays drawn. This is not so with raster devices.

Raster devices operate by ORing and ANDing into a bit plane so that things that are drawn first can get erased by subsequent objects that overlay them. This is known as **overpaint mode**. Objects can be overpainted (and therefore obscured) by putting other objects on top of them.

This can be very useful. Suppose you want to draw a picture of a square overlapping a circle. Look at Figure 1 on page 7.

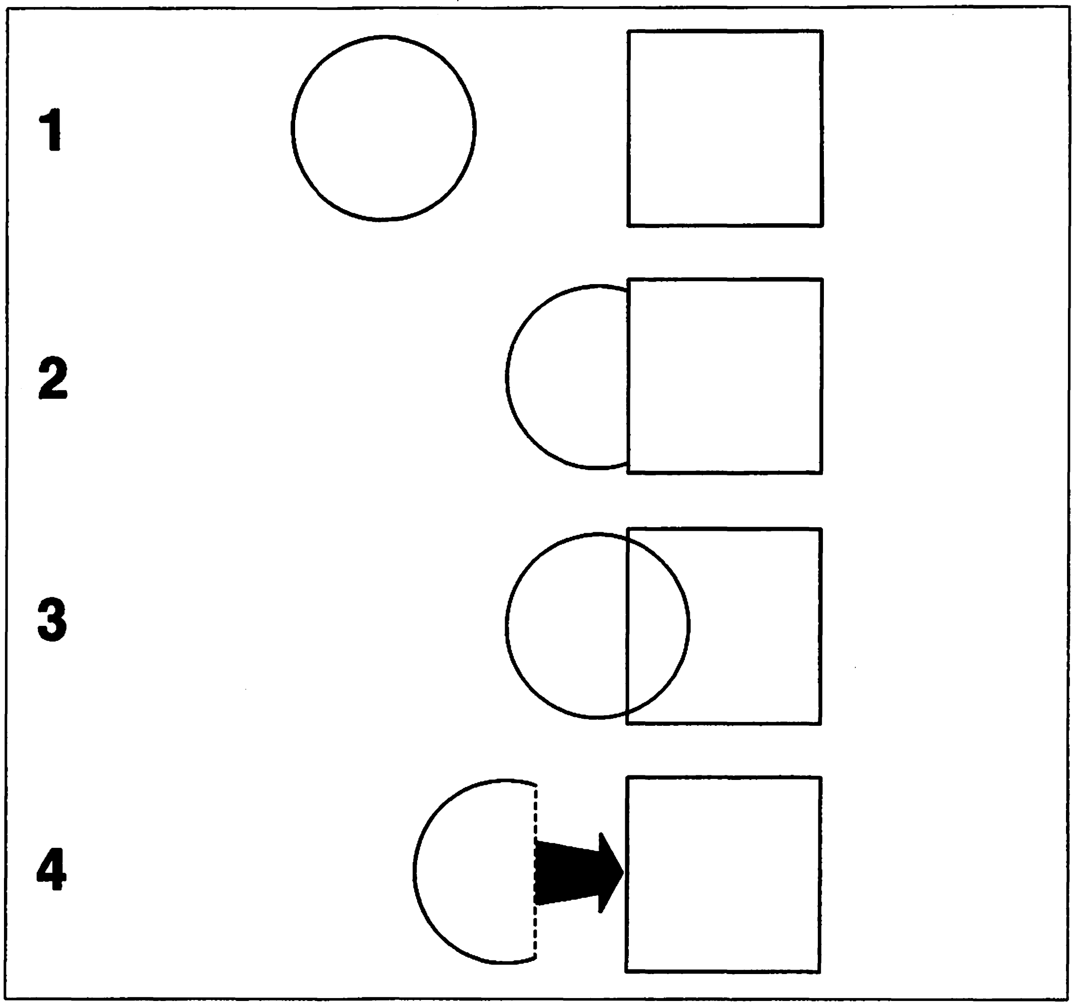


Figure 1. Reverse clipping

1. You could start by drawing a circle and a square.

2. Then you could put the square in front of the circle.

   What we haven't mentioned is that when you drew the square you filled it with **background color**, which is just like any other color except you cannot see it. When you're sitting at a raster device, the background color in the square will obscure bits of the circle automatically if you are in overpaint mode.

3. The square would not obscure the circle on a pen plotters, because there is no built-in eraser. So, to draw the picture properly on a plotter, you would have to make sure that the lines you cannot see do not get drawn. You can do this yourself, which might be quite a time-consuming job, or you can leave it to GDDM.

4. GDDM will work out which lines have to be removed to produce the same picture on a plotter as you see at your raster device. This process is known as **reverse clipping**.

Image devices

The 3117 and 3118 scanners and 3193 display station are desk-top devices that cater specifically for **image processing** – the electronic capture, storage, and display of documents – often called the "paperless office." Graphics devices not specifically designed for image processing, such as the 3179-G, the 3279, the 5550 family, and the 3270-PC/G and /GX family, support it by emulation, using GDDM's graphics facilities.

GDDM provides a set of calls to read and store images from the 3117 and 3118 scanners, manipulate those images if required, and output them to 3193 display stations and other displays and graphics printers.

An image is a picture, for example a photograph or a legal document, held as an array of dots, called pixels. GDDM supports bi-level images. These are images consisting of pixels that are either on or off. When uncompressed images are stored or transmitted electronically, each pixel is represented by one bit. The 3117 and 3118 scanners, which attach to a 3193 display station, will convert a document into image data, at a resolution of either 120 or 240 pixels to the inch, horizontally and vertically. It will convert the image into compressed format using a compression algorithm that is particularly suited to text and line art, and may reduce the size of the image by as much as a factor of 20.

The 3193 display station is a raster-scan device that displays images on a screen at 100 pixels to the inch and can perform a certain amount of image processing without reference to the host. For example, the 3193 can decompress an image that has been compressed using the modified modified read (MMR) compression algorithm. (This is the compression algorithm supported by the IBM Scanmaster scanner, and available when transferring images.) The 3193 can also do certain transforms, so offloading processing from the host (host offload) during transfer operations that have a 3193 as their target. When IMAPTx calls are used to send data to a 3193, host offload allows **direct transmission**. Host offload and direct transmission are further described later in this chapter. The various conditions that have to be satisfied to achieve direct transmission are described in "Image devices" on page 24.

Character devices

We mentioned earlier that the 3270-PC/G and /GX have an APA bit buffer, which is a representation of the screen. Character-oriented devices are different. Although the picture is still made up of a series of dots, it has to be decomposed into character-size units and the dot pattern of each has to be defined. The character definitions are then placed in the appropriate positions on the screen to make the picture. Figure 2 shows this. Notice that a character definition can be used in more than one screen position.

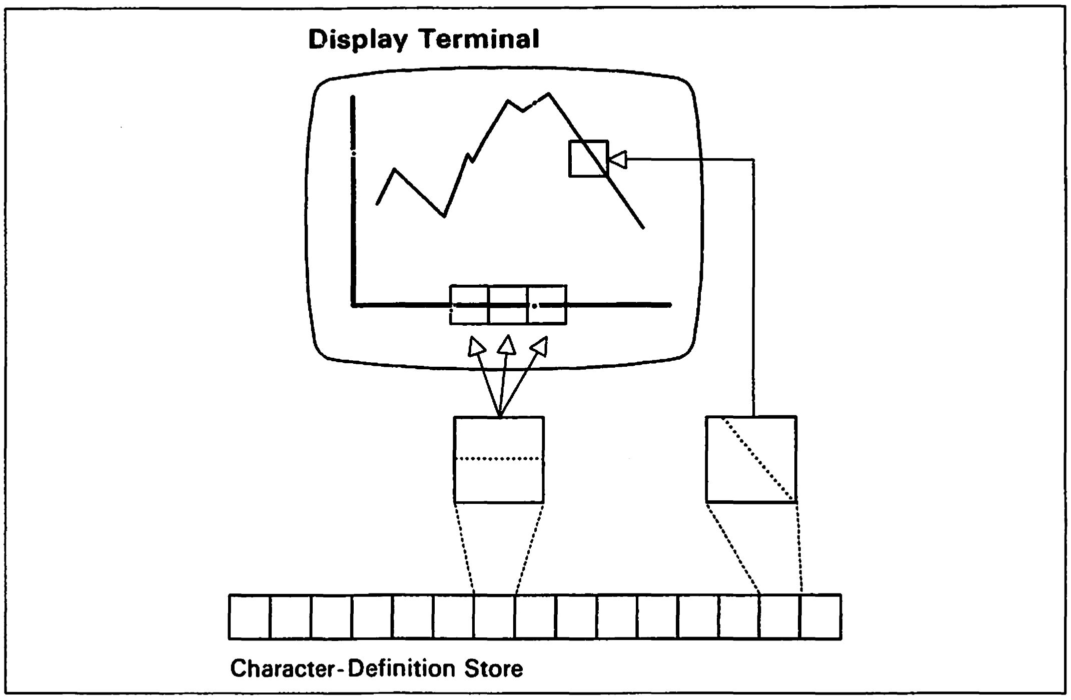


Figure 2. Character graphics in action

All the work to produce the dot patterns that make up the character definitions is done by GDDM in the host.

The 3278, 3279, 3290, and 8775 display terminals and the 3268 and 3287 matrix printers are all character-oriented devices.

It's worth understanding a major difference between character devices and vector devices at this point. This relates to the length of the data stream that each device needs to display a picture.

For vector devices, data stream is related to the number of graphics primitives (lines, arcs, and so on) in the picture. The more there are, the longer it gets. The relationship is not so direct for character devices. Here, data stream depends on the number of program symbol definitions used. Each program symbol (PS) definition can contain the representation of one or many vectors. Character devices tend to have longer data streams than vector devices for simple pictures like business charts, but as the number of graphics primitives in the picture increases, the vector data streams become longer.

You should know, however, that there are a finite number of PS definition slots in the 3279 which limits the data stream to a maximum size of around 50K bytes. Very complex pictures cannot be shown properly as they cause **PS overflow**.

There is a much higher limit to the number of PS definitions that can be used to print a picture on the 3287. There are fewer PS stores in the device, but GDDM constructs the picture using up to 32 "logical" PS stores. The picture is printed in pieces, one logical PS store at a time, the character definitions overwriting those in the device's programmable store that were sent for the previous piece. In this way it is possible to print pictures that would cause PS overflow when shown on a display terminal.

One final point about PS definitions. Because they are basically image definitions, they are suitable candidates for compression. GDDM can do this; the result is that the data-stream length for the picture is generally halved. There are costs associated with compression, so it is recommended only for remotely-attached devices.

Composed-page printers

Composed-page printers have two important characteristics:

1. For the 3800 and 3820 printers the resolutions are 120 or 240 pixels to the linear inch. The 4250 printer has 600 pixels to the linear inch. For a U.S. standard 8.5 x 11.0 inches piece of paper, or a European standard 210 x 297 millimeters A4 piece of paper (8.3 x 11.7 inches), that's more than 31 million pixels on a piece of paper. As you can imagine, looking after that number of pixels can be very costly.

2. GDDM does not communicate directly with these devices. It creates a print file in disk storage that is later processed by another program to produce the printed output. (For example, for the 4250, this is done by the Composed Document Print Facility (CDPF), and for the 3800-3, 3800-8, and 3820, by the Print Services Facility (PSF).)

GDDM's support of these devices is very similar to character devices, but with special considerations because of the high resolution. GDDM rasters the vectors onto an APA bit map, which is broken up into logical cells that are 32 * 32 pixels. Cells that are empty are either not put into the file (3800 and 3820 operation), or are subjected to compression (4250 operation).

How GDDM draws pictures

GDDM is made up of several layers of function. Some of the layers use other layers. The Interactive Chart Utility, for example, uses the Presentation Graphics Routines (known as the PG Routines). If you draw a picture using the Interactive Chart Utility, the panel options that you select are translated into GDDM PG Routine calls, like CHPIE to produce a pie chart. The PG Routine calls themselves get converted into GDDM Base calls, such as GSLINE to draw a line, or GSARC to produce an arc.

The GDDM Base calls are converted into the device-dependent data stream necessary to display the picture. For devices such as the 3270-PC/G and /GX, little work is necessary to do this. Devices like the 4250, on the other hand, require a very significant amount of effort from GDDM.

GDDM operates on several subsystems: CICS, VM/SP, TSO, and IMS/VS. The way it produces graphics is the same in all of them. Therefore, most of the GDDM code is independent of the subsystem that it runs under. All requests for subsystem services (such as device reads), are channelled through an environmental layer that is different for each subsystem.

Figure 3 shows the structure of GDDM.

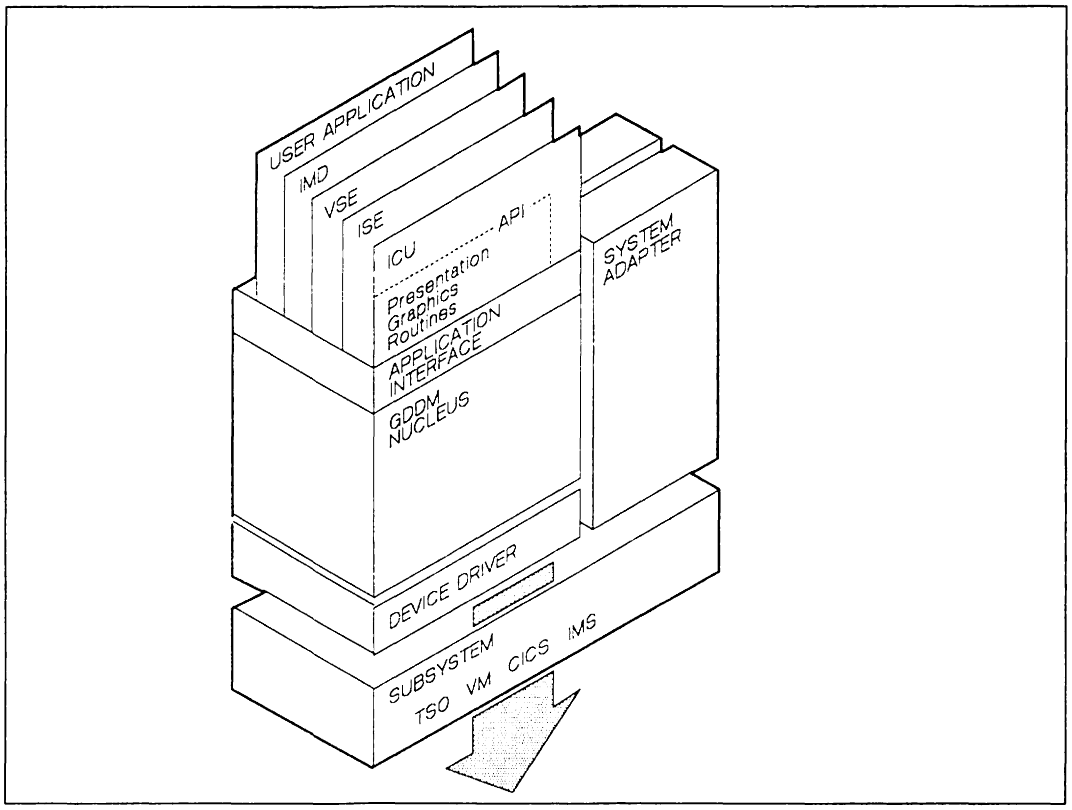


Figure 3. The structure of GDDM

What happens in a GDDM program

Understanding the sequence of events in a GDDM program will help to identify some of the areas where performance problems can occur.

1. The first GDDM call must be FSINIT. The primary purpose of this call is to establish the environment in which GDDM will operate. It starts GDDM processing by:

    a. Initializing GDDM default values.

    b. Resolving routine addresses.

    GDDM consists of many hundreds of routines, which, for convenience, are packaged together into several dozen load modules. As GDDM processes each call, it determines which routines are needed and checks if the routine entry-point addresses are known. The addresses are kept in a table and are resolved by finding the routines in shared storage or by loading the modules containing them into the application address space. There are various options open to you as to which GDDM modules get loaded, and when. These have important performance implications and will be discussed later.

2. The next calls concern the picture that you are about to draw, for example:

    - Which device you want it drawn on
    - The device characteristics
    - Which part of the display area you are going to use to draw on
    - What coordinate system you are going to use.

    One of the important things that happens now is that GDDM finds out about the device that you are going to draw the picture on. This happens through an application call, DSOPEN. At this time, GDDM may explicitly query the device and get information back, or it may use values obtained from the program or from auxiliary files, for example, a nickname file.

    The things that DSOPEN can tell GDDM that have a bearing on performance include:

    - Whether user control functions are available
    - Whether GDDM is to send high-precision vectors to the 3270-PC/G or /GX for panning and zooming under user control
    - The number of swathes (or horizontal slices) that control processing of the picture for the 4250, 3800, and 3820 printers
    - What level of ICU defaults is to be used
    - Whether GDDM is to use draft draw or full draw picture update mode for 3270-PC/G and /GX, 3179-G, and 5550 family displays

- Whether GDDM is to operate in retained or non-retained mode for the 3270-PC/G or /GX.

FSQURY, the call that queries device characteristics, will tell an image application whether the device being used supports the offload of certain image functions.

3. For graphics, further calls, which describe the picture, are translated into instructions that define the picture in terms of the elements that it is made of: lines, arcs, or areas. These elements are kept in a table in main storage in **graphics data format (GDF)**.

   Note that these calls do not cause the picture to be sent to the device.

   GDDM constructs two versions of the GDF file, either of which can be accessed by the application program:

   a. Floating point GDF

      - Corresponds almost directly with the API calls issued.
      - Is kept in short floating-point notation corresponding to user-defined coordinates
      - Has not had any transformations applied. These are kept separately in a **transformation matrix**.
      - Is device-independent.

   b. Fixed point GDF

      This form is device-dependent. It takes its name from the fact that the coordinates are held in fixed point form. They have also been converted to a GDDM-defined coordinate system that is device-related.

   The advantages of using floating-point GDF, rather than fixed point GDF, as the method for storing pictures are discussed in the *GDDM Application Programming Guide.*

   For image, direct transmission does not store the image if the entire projection is within the capabilities of the 3193, and projections are performed by the device. Where the device does not support direct transmission, the images are held in a GDDM-internal format in dynamic storage.

4. The picture is drawn and sent to the device when an input/output call is issued, for example, FSFRCE, ASREAD, GSREAD, MSREAD, or WSIO.

   For graphics, these calls cause the device-dependent data stream to be generated from the fixed point GDF file that was created by earlier calls.

   In all cases except direct transmission on the 3193, images are converted from a GDDM-internal format to uncompressed format.

   On all devices, overlapping partitions and operator windows are clipped/merged. Different things happen for the different device classes.

a. Programmable order-driven devices

   Generally speaking, the GDF to be sent will have already been generated. However, a regeneration of some of the data may be necessary in the following cases:

   - Applying any segment transforms
   - Sending any referenced symbol sets that are not currently in the work station and, for 5550 terminals,
   - Clipping the picture to the segment viewing limits
   - Expanding characters from symbol sets that could not be stored in the device.

   Having created the picture, if GDDM finds that it or the associated symbol sets are too large to fit into the available segment storage in the work station it will degrade into non-retained mode. This degradation is a multistage process. GDDM will switch back through the stages if a subsequent picture display within the session is small enough to permit it. The stages are:

   1) Everything is held in the work station.

   2) Only symbol sets are held.

   3) Nothing is held.

   When multiple partitions or operator windows are used, all device resource, such as symbol-set storage is allocated based on window priority. When more than one graphics field is defined, segments are not stored in the device.

   The advantage of the second step (active segment + symbol sets) is that it permits local dragging of a segment even if there is insufficient storage to hold the entire picture.

   If GDDM is in non-retained mode, then on a 3270-PC/G, area boundaries have to be sent separately from the definition of the area.

b. Nonprogrammable order-driven devices

   GDDM has to prepare the data stream under the device limitations discussed earlier; regeneration as discussed for order-driven devices may occur.

   For plotters, area-fill patterns are expanded into simple vectors, and reverse clipping is carried out to remove objects that will be obscured by opaque areas.

   For 3179-G, in some cases characters are expanded into simple vectors or images.

c. Image devices

   In some cases (direct transmission, for example) the data may have been sent to the device as part of the transfer operation, before the ASREAD.

   In other cases, GDDM holds the image data and sends it to the device when the ASREAD or similar call is issued.

d. Character devices

   For graphics, GDDM converts each graphics primitive in the GDF file (line, arc, and so on) into dots and assigns the dots to character definitions. Some checking is carried out to eliminate most of the identical definitions to reduce the data stream.

   When operator windows are used, device resources are first allocated to the highest priority window. Remaining resources (if any) are allocated to lower priority windows.

   For image, the image data is rastered by GDDM into programmed symbols.

   Doing this work takes 60 to 80% of the total GDDM processing time used to display a picture on a character device.

e. Composed-page printers

   GDDM converts each graphic primitive in the GDF file into pixels. These pixels are stored in the relevant cells. Cell data is reworked into the format required by the printer that is to print the picture. This formatted data is stored as an ADMIMG file. If color separation has been requested, several files are produced, each one corresponding to one of the color masters.

5. The complete picture is transmitted to the device by a series of calls to the subsystem. The device could be a display terminal, a printer, or disk storage for a saved picture or print data set.

   Each call transmits a piece of the picture. Each piece is by default 1536 bytes (the default transmission buffer size).

   You can change the default transmission buffer size by specifying the ADMMDFT IOBFSZ defaults, which are described in the *GDDM Installation and System Management* manual appropriate to the system that you are running under.

   Data stream compression, which can be specified for character devices like the 3279, happens as the character definitions are moved into the GDDM output buffer.

   The flow of GDDM terminal transmissions through the subsystem is discussed later.

   For direct transmission of image data to the 3193 terminal, each buffer of image data is transmitted to the 3193 as it is passed to GDDM.

6. The final call in a GDDM program is FSTERM. This destroys the GDDM environment by releasing all the storage that GDDM has acquired for the application and deleting all the loaded modules.

GDDM resource usage in picture production

By now, you should understand that GDDM's resource requirements for picture display depend on two main things:

1. The characteristics of the display device
2. Picture complexity.

In this section we are going to try to define, approximately, what these resource requirements are. You should be aware, however, that whatever numbers we come up with are likely to be vague. If you want some numbers that have more substance to them you should see "Capacity planning – first-pass method" on page 68.

The main areas of resource usage in picture production are:

1. Network time needed to handle the data stream.

   We will also include 3270-PC/G or /GX segment storage in this category, and the amount of disk space required for composed-page printer picture generation. Both are related to data-stream size.

2. Host-processor time.

3. Host-processor storage required (a) for referenced code and (b) for temporary storage to hold information relating to the current transaction.

   Where there is insufficient real storage in the host processor to meet these requirements, paging will take place. This will increase the host-processor time and cause I/O operations to storage.

Data-stream size

Programmable order-driven devices

The main determining factor of data-stream size for these devices is the number of graphic primitives in the picture definition. The following table suggests data-stream ranges for "typical" pictures in various graphics application areas.

GDDM picture	*Data-stream size*
Business charts	1 – 10K bytes
General graphics	1 – 10K bytes
Scientific and engineering	5 – 30K bytes
Data-driven pictures	20 – 100K bytes
Images	20 – 100K bytes

"Data-driven" pictures include any that are generated automatically from some sort of data-collection process; there are many graphics elements in the picture by implication. Typically, this class would include such areas as seismology, cartographics, and civil engineering.

All numbers assume retained-mode operation and exclude any character fonts that may be required. Retained-mode operation is the default, if there is sufficient segment storage allocated to the host session in the work station.

There is no limit to the complexity of pictures that can be displayed. Unlike the 3279, PS overflow cannot occur on programmable order-driven devices. GDDM and the work station may have to operate in non-retained mode to display very complex pictures.

Items that affect data-stream length, in order of importance, include:

- Picture complexity, that is, the number of graphics elements (lines, arcs, areas) or image size and content
- Number of different font styles used
- Use of retained mode
- Use of high-precision coordinates for pan/zoom
- Number of segments in the picture.

Customizing the work-station session with sufficient storage to enable applications to run in retained mode can give significant performance improvements for reduced data streams even for output-only applications. There are two reasons for this:

1. Areas

   For the 3270-PC/G only, the work station has to make two passes of the area definition. First to draw the fill, and second to draw the boundary. In non-retained mode, the work station discards each vector when it is processed, so it cannot make a second pass to draw the boundary. So, to draw it properly, GDDM has to send the area definition twice, thus doubling the data-stream length.

2. Non-default character fonts

   If there is sufficient symbol-set storage, GDDM will send fonts to the storage in the work station before sending the picture data stream. The picture data stream will contain references to the characters within the fonts. The fonts will remain in storage in the work station to be used by subsequent picture displays in this GDDM session. Typical GDDM-supplied fonts are 10K bytes each.

   If there is no room in segment storage for the font, GDDM will expand character definitions in the host and include them in the picture data stream. For mode 3 (vector) characters, this adds 50 to 100 bytes to the data stream for every character in the picture. Typical business charts will contain 50 to 200 characters from two fonts.

See "3270-PC/G and /GX segment storage requirement for GDDM" on page 67 for some suggested figures for segment storage allocation.

The GDDM LCLMODE processing option, specified either explicitly with the DSOPEN call or through nicknames, offloads some of the panning and zooming function of user control to the work station. The overall total data stream for a user session should reduce if local panning/zooming is used rather than relying on GDDM to perform the operation in the host. However, this may cause individual picture data streams to increase.

In non-LCLMODE operation, the vector definitions that GDDM sends down to the device use pixel coordinates. This means that the locations defined by the x and y coordinates correspond directly with dot positions on the screen. This has performance advantages over using a coordinate system that doesn't correspond to pixels:

1. GDDM is able to convert more of the x,y pairs into "relative" definitions. A relative coordinate occupies one byte rather than two and can be used if it defines a point on the screen that is within plus or minus 128 units (in both x and y) from the current position. For example, the 3270-PC/G has a screen size of 720 x 512 pixels. Thus, many vectors will be relative if pixel coordinates are used.

2. GDDM does not send any very short vectors which do not activate any new pixels in the device.

A coordinate system of greater precision is needed if the picture is to be zoomed. Typically, GDDM uses a coordinate range that is 16 times the number of pixels on the screen.

With this system, few relative vectors will occur, and more vectors will be sent, even those that are currently too small to be seen. Data-stream size will therefore increase. For complicated pictures made up largely of lines, the "zoomable" data stream is likely to be twice as large as the "nonzoomable" one.

This increase in data stream can be offset when the picture is panned or zoomed locally. If the operation was performed by GDDM in the host, a new picture definition would have to be computed and sent to the work station for each new view.

When you make changes to a picture displayed on a 3270-PC/G or /GX, the work station may draw just the update, or it may need to redraw the entire picture. This can be controlled by setting the **update mode** which can be set to draft-draw mode or full-draw mode, either:

- By the user in user control
- By a system or application programmer in a nickname file
- By an application programmer using a procopt to DSOPEN in a program
- By an application programmer using the call FSUPDM.

In draft-draw mode, you can avoid complete redraws. Faster updates can be obtained in most cases, because GDDM updates only the changed segments. There is a cost in possible drawing inaccuracies, where primitives overlap either before or after the update, because GDDM degrades the color mixing. Overlapped sections might therefore be missing or in the wrong colors.

In full-draw mode, additions to the picture generally cause partial redraw. Deleting segments or changing segment attributes (visibility, priority, and so on) require a total picture redraw; deleting an object may make other objects visible, for example. The entire picture is redrawn to make them appear correctly. It may therefore be sensible to "batch up" operations that need total picture redraw.

For retained mode, full draw does not mean that the entire picture is being retransmitted by GDDM; only the changes are sent. The work station incorporates these into the picture definition that it is holding in segment storage.

Nonprogrammable order-driven devices

3179-G: Data streams are always sent in pixel coordinates. Data streams will be greater than for the 3270-PC/G or /GX because:

- Characters from non-default fonts are expanded to vectors in the host, adding 20 to 50 bytes per character.

  For example, the 3179-G has storage for four image symbol sets, but no vector symbol sets (other than the default). This means that, where a non-default vector symbol set is used, as in the ICU quality defaults, GDDM has to expand the vector characters into primitives before sending them to the terminal. Typically, an ICU chart will have 50 to 200 characters, which must each be expanded to 20 to 50 bytes of data stream. So 3179-G data streams can be from 1K to 10K bytes longer than equivalent data streams for the 3270-PC/G and /GX. This could have a significant effect on remote terminals.

- Arcs are represented by several very short vectors.

- As the 3179-G has no segment storage, interactive-graphics data streams are generally greater than on a 3270-PC/G and /GX, because the entire picture has to be retransmitted for any redraw. Draft draw mode, see above, can significantly reduce the amount of data sent when segments are changed or deleted.

4224 printers: Like the 3179-G, the 4224 has no segment storage.

Plotters: As well as the reasons quoted above for the 3179-G, plotter data streams will be longer because:

- Relative vectors are not used, so the more complex pictures will be twice as large as for the 3270-PC/G or /GX.

- These devices do not "fill" areas. Any shading pattern has to be broken into vectors in the host. For a picture consisting totally of shaded areas this can increase the data stream by a factor of 10 over the 3270-PC/G or /GX.

Image devices

If you consider, for example, the kind of newspaper photograph that is made up of dots, and consider the total number of dots there are, you will appreciate the number of bits that can be involved in image data streams. As another example, a U.S. standard 8.5 x 11.0 inches document is around 670K bytes at 240 pixels per inch (ppi), or 168K bytes at 120 ppi, of uncompressed data. A European standard 210 x 297 millimeters A4 document (8.3 x 11.7 inches) is around 700K bytes at 240 pixels per inch (ppi), or 175K bytes at 120 ppi, of uncompressed data. Factors affecting the size of image datastreams are:

- Compression
- Size
- Resolution

Compression can significantly reduce the amount of image data. For example, if an 8.5 x 11.0 inches document scanned by an image scanner contains text, MMR compression could reduce it by a factor of around 20. It would then require around 20K to 30K bytes. That's about the size of a fairly large ADMGDF file. Equivalent ADMIMG files may be 60K to 150K bytes. You can use either uncompressed data, or data that has been compressed using one of the following algorithms:

- Modified modified read (MMR)
- 4250
- 3800.

The above compression types use different algorithms to compress the data. Consequently certain compression methods are suited to certain types of image data. For example, MMR compression gives good results for regular data such as text and line art, but may actually expand an irregular image such as a photographic image. Image compression uses a lot of host-processor time. Sending compressed images will also increase the time taken by the device to display the image, as it has to uncompress the data first. See Figure 4 for details of the supported compression algorithms.

You can reduce the size of an image by excluding the data that is irrelevant or not required. For example, you might need only the signature from a letter. You could extract it when the document is scanned, or later, either by just extracting the sub-image containing the signature or by trimming off the irrelevant data so that only the signature remains.

If an image is only going to be displayed at a terminal or on a low-resolution printer, it need only be stored or sent at low resolution.

A feature of the 3193 that allows reduced data stream is that it can decompress image data.

Only certain combinations of format and compression can be used on IMAGTx and IMAPTx calls. Acceptable combinations are shown by a Yes in Figure 4. Acceptable combinations that allow direct transmission are shown by **Yes – direct**.

	Unformatted	**3193 data stream**	**CPDS**
Uncompressed	**Yes**	**Yes – direct**	No
MMR	**Yes**	**Yes – direct**	**Yes**
4250	No	No	**Yes**
3800	No	No	**Yes**

Figure 4. Acceptable combinations of format and compression

For details of application programming techniques for image performance, see "Image" on page 121. For full details of GDDM image support, see the image processing chapters of the *GDDM Application Programming Guide, Volume 1.*

Character devices

It is important to remember that the main factor influencing data stream size for these devices is the number of unique program symbol definitions used, **not** the number of graphics elements in the picture.

Because of the finite size of the Program Symbol store, the maximum data stream size for a 3279 picture is around 50K bytes. This can be compressed, typically for remote transmission. Compression usually halves data-stream length.

The following table suggests uncompressed data-stream ranges for "typical" pictures in various graphics application areas.

GDDM picture	*Data-stream size*
Business charts	10 – 25K bytes
General graphics	5 – 30K bytes
Scientific and engineering	10 – 40K bytes
Data-driven pictures	20 – 50K bytes
Images	20 – 50K bytes

Factors affecting data-stream length, in order of importance, include:

- Picture area
- Number of graphics primitives used
- Use of different colors
- Amount of area shading
- Length of lines and arcs.

Data streams for the 3287 printer are usually similar to those of the 3279 display terminal. However, they may be shorter, because the 3287 supports only four colors.

Data-stream compression: GDDM can compress the character definitions before it transmits them to remote terminals. This can be a significant advantage when the transmission speed of the link is low. For example, a 9 600-bit per second link takes 20 seconds to send a 24K byte picture. This can be reduced to about 10 seconds by using data-stream compression.

Data-stream compression can occur if the 3274 controller is configured for decompression. GDDM will normally generate compressed PS load data streams if the 3274 can decompress them. This is discovered from the "query device" request. GDDM will not compress if it can determine that the device is local, either from the subsystem (on VM) or from GDDM external default settings (the ADMMDFT AM3270 default).

Compression will generally halve data-stream length although there are some costs associated with it.

- Host-processor time to compress is significant, although it is partially offset by the reduction in terminal access method overhead because of the smaller data stream.

- The 3274 time needed to decompress the data stream causes a significant reduction in the rate at which the 3274 can process the data.

There is a restriction, in the remote non-SNA environment, concerning the size of the block that can be sent to the 3274 when character definitions are present in the transmission. In this environment, you should not change the GDDM default transmission size, the ADMMDFT IOBFSZ default.

Compression/decompression is recommended for devices attached to teleprocessing lines where the line speed is less than the rate at which the 3274 can decompress and process the data stream. Typically, data compression is an advantage at line speeds of 19 200 bits per second or less. Using compression does not affect the amount of main storage space used while the picture is being created.

Composed-page printers

GDDM does not generate a data stream for these devices, but stores the picture data in a file. It is the responsibility of other programs to cause the file to be printed.

Processor time requirement

All numbers quoted in this section are in seconds of 4341-2 host-processing time. You should be aware that the elapsed time in the host processor depends on, among other things, how busy the processor is, and relative job priorities. Elapsed time will be at least twice as long as processing time.

Usually, the factors influencing host-processor requirements are the same as those that affect data-stream size. Only variations will be discussed here.

You should bear in mind that the non-GDDM part of an application can add considerably to the host-processor requirement. This is particularly true where processor-bound jobs such as data sorting or regression analysis are involved.

Programmable order-driven devices

The following table suggests the amount of processing time required by a 4341-2 Processor to display a typical picture in various application areas:

GDDM picture	*Processing time*
Business charts	0.5 – 2.0 seconds
General graphics	0.2 – 3.0 seconds
Scientific and engineering	0.2 – 5.0 seconds
Data-driven pictures	5 – 20.0 seconds
Image (stored as MMR-compressed)	5 – 40.0 seconds

Besides the factors that influence data-stream size, an influence on the host-processor time required for these devices is the switch from retained to non-retained mode. This happens if, when GDDM has constructed the picture, it finds that it is too large to fit into the segment storage in the work station. Here GDDM will switch to non-retained mode. For the 3270-PC/G, this means that it will then have to rebuild the picture, inserting area boundaries as separate definitions. If a picture contains many areas, the processor time required may increase by as much as 50%.

GDDM will automatically switch back to retained mode should a subsequent picture in the session be small enough to fit into segment storage. If every other picture is too large to fit into segment storage, GDDM has to alternate between retained and non-retained modes. If these pictures also contain many areas, the total processor time required may increase by approximately 25%.

To prevent this happening, you can:

1. Allocate more segment storage to your host session

2. Run in non-retained mode from the beginning

   a. Using the SEGSTORE processing option, specified with the DSOPEN call or in GDDM external defaults

b. By selecting output-only mode for the session during work station customization.

Nonprogrammable order-driven devices

For the nonprogrammable device class in general, the processor requirement is similar to the programmable device class. Exceptions are plotters, where area fill is used, and reverse clipping has to be done.

For the ICU, the chart defaults available use non-default vector symbol sets, to improve the visual appearance of charts. However, this can significantly increase both the data stream and host-processor requirements for this class of device. You can choose to avoid this overhead by using a nickname to select the Version 1 Release 4 defaults. This is described in "Hints and tips on efficient usage for all subsystems" on page 41.

When you use the 3179-G for ICU business charts, GDDM processor utilization is up to 10% greater than the 3270 PC/G and /GX requirement for the Version 1 Release 4 defaults.

The cost of reverse clipping is related to the number of opaque areas that overlap other objects, the number of vectors that define the areas, and the objects that are overlapped.

Although reverse clipping costs will usually be quite small, it is possible for some pictures for host processor costs to be more than 10 times greater than 3179-G requirement.

Image devices

The 3193 can lighten the load on the host by performing some local image processing. GDDM takes advantage of these features to save on processor usage. The following transforms are done by the 3193, if they are within the capabilities of the device:

- Extraction/placing
- Rotation/reflection/negation
- Scaling
- Resolution conversion.

The biggest factor affecting CPU requirement for the 3193 is whether GDDM can use direct transmission on a PUT operation (IMAPTx calls). When direct transmission is used, a GDDM application passes a buffer of image data (either uncompressed or MMR compressed) to GDDM, and the buffer is sent to the device, preceded by any transforms.

Direct transmission to the 3193 is used by default, if the data is in 3193 data-stream format, if the 3193 can perform the entire projection, and if GDDM does not otherwise need to perform the projection itself (for example, to maintain a read-write image field). If the preceding conditions are satisfied, GDDM sends the data directly to the 3193 without keeping a copy of the entire image.

If the 3193 cannot do the entire projection, GDDM performs the functions in the host as necessary. GDDM compresses each buffer of the PUT sequence into a GDDM-internal format, and stores the entire image. This can take from 4 to 20 seconds. The transforms that the 3193 cannot do are applied in the host. When an input/output command is issued, the data is converted into a device-acceptable format (either MMR or uncompressed) and sent to the device. This can take a further 5 to 30 seconds for transforms like extract and scale. More complex transforms, such as orientate, can take up to 70 seconds of processor time. The figures assume resolution conversion is set on. Setting resolution conversion off, or displaying data at the correct resolution, significantly reduces the figures.

For direct transmission, the host-processor requirement depends on the size of buffer passed through GDDM. The smaller the buffer, the more GDDM calls are required and the higher the host-processor requirement. However, storage requirements may be less. The following table gives some typical figures for host-processor time taken by direct transmission to a 3193 of MMR compressed text or line art:

Buffer size	**Processing time**
400 bytes	1.0 – 2.0 seconds
2K bytes	0.2 – 0.5 seconds
4K bytes	0.1 – 0.2 seconds

For details of application programming techniques for image performance, see "Image" on page 121 in Chapter 5, "Application programming for performance." For full details of GDDM image support, see the image processing chapters of the *GDDM Application Programming Guide.*

Character devices

For simple pictures, like business charts, the processor requirement for picture generation is related to the length of the data stream produced. This is not true when the picture becomes more complex and the data stream approaches the maximum size.

The following table suggests the amount of processing time required by a 4341-2 Processor to display a typical picture in various application areas:

GDDM picture	***Processing time***
Business charts	1.5 – 5.0 seconds
General graphics	1.0 – 10.0 seconds
Scientific and engineering	1.0 – 15.0 seconds
Data-driven pictures	10.0 – 50.0 seconds
Images	10.0 – 50.0 seconds

One of the major factors affecting processor requirement that does not affect the data stream is the number of vector (mode 3) characters used.

For user control, the host processor requirement for each new viewing operation is effectively the cost of regenerating the screen.

Composed-page printers

There is no easy guide here, because the picture generation process depends on a significant number of variables, such as the final picture size, the number of vectors making up character text, the amount of storage space available, the swathing option, the spill file option, and device resolution. The range can be expected to be from 5 seconds to 5 minutes. The average time for a 6 x 4 inches (15 x 10 centimeters) picture of medium complexity is between 30 seconds and 1 minute.

Processor-storage requirement

There are various terms used to describe processor-storage requirements and it is as well to understand what each of them means. This section discusses processor storage under three headings:

- Virtual-storage requirement
- Reference set
- Working set.

Virtual-storage requirement

The virtual-storage requirement can be divided into two discrete pieces:

1. The amount of virtual storage required to load the GDDM code and objects necessary for program execution. This is usually known as the **code size.**
2. The amount of virtual storage required to satisfy GDDM's storage requests. This is sometimes called the "user dynamic storage requirement".

The two added together are sometimes known as the "main storage requirement." Let's look at code size first.

There are two factors under your control that influence the GDDM code virtual-storage requirement:

1. Where GDDM is kept.

   You have the choice between:

   a. Keeping one copy of GDDM in a shareable area, like the VM Discontiguous Saved Segment (DCSS) or OS Pageable Link Pack Area (PLPA), which can be executed by many users simultaneously.

   b. Letting users have individual copies of the code in their own address spaces or virtual machines.

   All but around 10K bytes of GDDM code (mainly entry point addresses) will go in a shareable area. Keeping it there reduces the GDDM virtual-storage requirement if you have more than one concurrently active user. There are other performance implications associated with each choice, and these are discussed further in "Finding the optimum loading and packaging combinations" on page 35.

2. What GDDM code is loaded.

   The total size of all GDDM code is over 3 megabytes, but it is unlikely that applications in your environment will need to execute all of it. For example, you may not want the code for Program Symbol generation if you have 3270-PC/Gs rather than 3279s. There are two ways that you can prevent unnecessary code getting loaded into main storage:

   a. You can leave GDDM to load modules on an "as-needed" basis.

b. You can select the GDDM functions that you want and choose when to load them.

There are advantages and disadvantages in both approaches. These are discussed further in "Finding the optimum loading and packaging combinations" on page 35.

The other contributor to virtual storage use is the "user dynamic storage" requirement.

GDDM keeps all the information about what you are doing in an area of storage special to you. It will dynamically acquire and release storage to meet your needs, so the size of the area varies.

The sort of things that GDDM keeps in user dynamic storage are:

- Control blocks
- The definition of the picture being created. This is held in various forms:

  1. Graphics Data Format in both

     Floating-point form, and
     Fixed-point form.

  2. Data stream

     Program symbol definitions for character devices
     Device-dependent GDF files (expanded arcs, area fill, and so on).

Because of the additional functions in GDDM Version 2 Release 1, applications that use the functions may require more user dynamic storage than Version 1 Release 4. For example, when using operator window support, the user dynamic storage requirement will be significantly increased because GDDM needs to monitor such things as window size, priority, and overlapping areas. Also, functions such as chart-by-example, and non-direct transmission of image data, will use large amounts of dynamic storage. See Figure 14 on page 66 for figures. If you are changing to Version 2 Release 1 from Version 1 Release 4, or from an earlier release, you should consider this.

Users of order-driven devices will always have a smaller requirement than those on character devices. The difference, again, depends on the type of picture being displayed. For simple business charts the requirement will be halved, whereas for very complex cartographics the reduction may be as little as 5%.

The following factors also contribute to the user dynamic storage requirement:

- Use of multiple graphics pages
- For character devices, the size of the graphics field
- Use of PG Routines/ICU.

Use of multiple graphics pages can cause significant increases as the GDF file and data stream are held for each graphics page. The storage overhead is reduced if the application deletes the graphics field for non-current pages, and redefines it where necessary.

The size of the virtual machine/address-space/partition needed to execute a GDDM program is determined by the user's dynamic-storage requirement and the virtual-storage size of that part of the program that is not in a shared area.

The virtual-storage requirements (code and dynamic storage per user) of the major GDDM functions are shown in Chapter 3, "Storage requirements and capacity planning" on page 61.

As an example, the optimum virtual-storage requirement of the Interactive Chart Utility is approximately 1100K bytes for the code plus 30K to 50K bytes of dynamic storage per 3270-PC/G or /GX user, and 60K to 90K bytes per 3279 user. These numbers assume that GDDM is allowed to load code as required. All but about 10K bytes of code could be placed in a shared area.

One final point, for MVS/XA users. Most of the GDDM code can be placed above the 16-megabyte line. Some user dynamic storage, however, is still obtained from below the line.

Special considerations for composed-page printing

The amount of dynamic storage needed for composed-page printing depends on the size and complexity of the picture. It can be several megabytes.

You can use two methods to reduce storage requirements at the expense of increased processing time:

1. Break up the picture generation into several horizontal slices, or swathes. GDDM then needs to hold picture data as swathes, instead of as entire pictures. Swathe storage can be reused for subsequent swathes. Processor time increases because the coded picture definition, the GDF, is interpreted from start to finish for each swathe, to determine which vectors appear within the current swathe. Swathing is controlled by the HRISWATH processing option.

2. Cause the GDF orders to be held in a spill file rather than in main storage. The GDF file that becomes the input to the image-generation process is held in its expanded form. This means, for example, that arcs have been expanded into a series of line orders, vector symbols have been expanded into a series of line orders, and that line orders, where necessary, have been expanded into area definitions. So, expanded GDF for composed-page printers can become large, and it becomes useful to be able to store it away on disk. Processor time increases because of the additional input/output activity need to write and read the GDF orders to and from the disk. Spilling is controlled by the HRISPILL processing option.

The reference set

The reference set is the amount of storage that will be referenced or changed during the execution of an application, and is generally much smaller than the virtual-storage requirement. It is smaller because the GDDM code that gets loaded contains routines that will not necessarily be used during the invocation of a particular program. These routines may be things like error handlers or calls that are not being used by this program.

So, the GDDM reference set in a typical graphics program is probably around 50% to 70% of the code size.

The reference set may contain some code that is used infrequently. GDDM initialization/termination routines, for example, are only executed once by each application.

Code that is seldom used, like the initialization routines, is part of the reference set but is not present in real storage most of the time. Another measure – the working set – is used in assessing how much real storage is actually required for satisfactory program execution.

The working set

The working set is the amount of real storage that has to be available to enable the application to run without excessive paging. The definition of "excessive" cannot be precise. It might well be that no page faults are desirable, in which case working set would equal reference set. In general terms, the working set of an application that uses GDDM is probably around 50% to 70% of the reference set.

Understanding and managing GDDM performance

The preceding discussion shows that there are two significant factors that affect GDDM performance. The first is the potential for high peaks of resource usage caused by the generation of graphics. The second is the size of GDDM. By understanding these, you can minimize their effect both on GDDM users and on other users of the system.

The next two sections give you the necessary background information to manage them.

Smoothing out the peaks of resource usage

When GDDM displays a picture, it can use a lot of system resources. It all depends on the type of picture and the type of device that is being used to display it. The use of a lot of system resources has two implications for performance:

1. Increased use. As more system resources are used, transaction response times tend to get longer. This is particularly true for transactions that have a low priority in the system. There comes a point when even small increases in resources used cause very large jumps in response times. Figure 5 on page 32 illustrates this clearly. The chapter on capacity planning will help you estimate whether you are going to exceed your resource-utilization threshold by installing GDDM.

2. Peaks of resource use. The average resources required by a GDDM application may be no greater than for any of your other non-graphics applications and you may be well below the utilization threshold. However, while your other applications make small but frequent requests, GDDM may make large, but generally infrequent, use of resources. People who display pictures are likely to want to look at them for a long time.

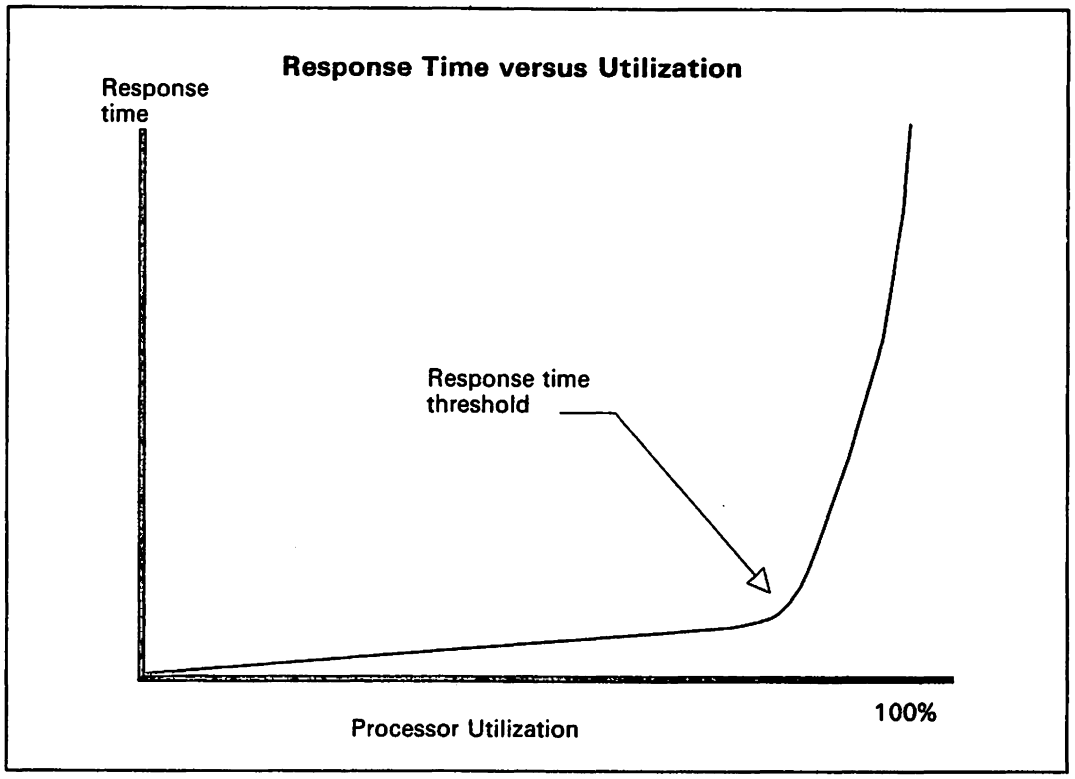


Figure 5. Response time versus processor utilization

The hold up at the bank – a fable

Demands for large amounts of resource cause problems that are best illustrated by thinking about a "typical" bank. When you walk in to make a transaction, you might have to join a queue or you might get served straight away. If you join a queue, how long you are kept waiting depends on what the people in front of you are doing. You will probably get served quicker if there are ten people in front of you making simple withdrawals than if, say, the owner of a chain of 21 shops is paying in the takings from all the shops. It doesn't happen very often, but when it does, everyone behind that person has to wait.

There are several ways that your delay in the bank could be minimized. There might be a special queue for people who want to make lengthy transactions, or the teller might occasionally interrupt the lengthy transaction to deal with smaller transactions.

Exactly the same sort of procedure takes place in a processor. We can specify that transactions have priorities, so that those needing a good response time get processed first. Most of the subsystems also operate a time-slicing technique so that transactions that consume large amounts of processor time at a gulp get interrupted, and other people have a bite. We can usually decide the size of the slice that we want each user to have. CICS systems do not practice time-sharing in this way, but GDDM hands back control to CICS periodically so that transactions with a higher priority can be dispatched.

The same problem — how to stop large infrequent resource requirements affecting other users — exists in remote communications networks.

We can see a partial solution by looking at the flow of the GDDM-produced data stream through the different terminal access methods. In practice, there are only two to consider, SNA and non-SNA. As a starting point, remember that GDDM breaks the data stream into separate transmissions. The GDDM default buffer size for each transmission is 1536 bytes. Except for IMS/VS, which puts all the transmissions back together, all the subsystems treat each GDDM transmission as a separate, independent message.

Non-SNA: The GDDM transmission gets passed to the terminal in one SEND regardless of size. For IMS/VS, this means the entire data stream. Figure 6 shows what happens.

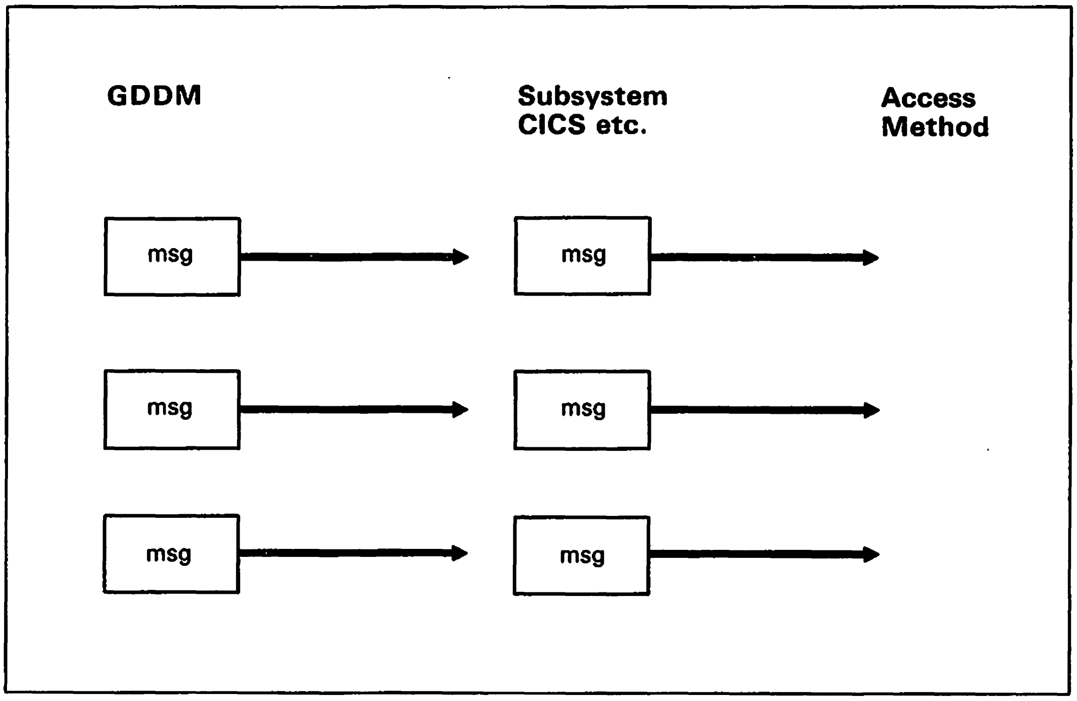


Figure 6. Data flow through a non-SNA system

SNA: The GDDM IOBFSZ defines the amount of data sent for each definite response. So IOBFSZ should be set to a high value, say 4K. When sent across the network, the data will be segmented into chained **request units (RUs)**. The RU size can be individually defined for each terminal.

By manipulating the GDDM IOBFSZ size for non-SNA, or the IOBFSZ and the RU for SNA, we can generally prevent large messages causing problems in the remote network. Some sort of balancing act will always be necessary to reach a compromise between the GDDM users and the other users of the network. Figure 7 on page 34 shows what happens.

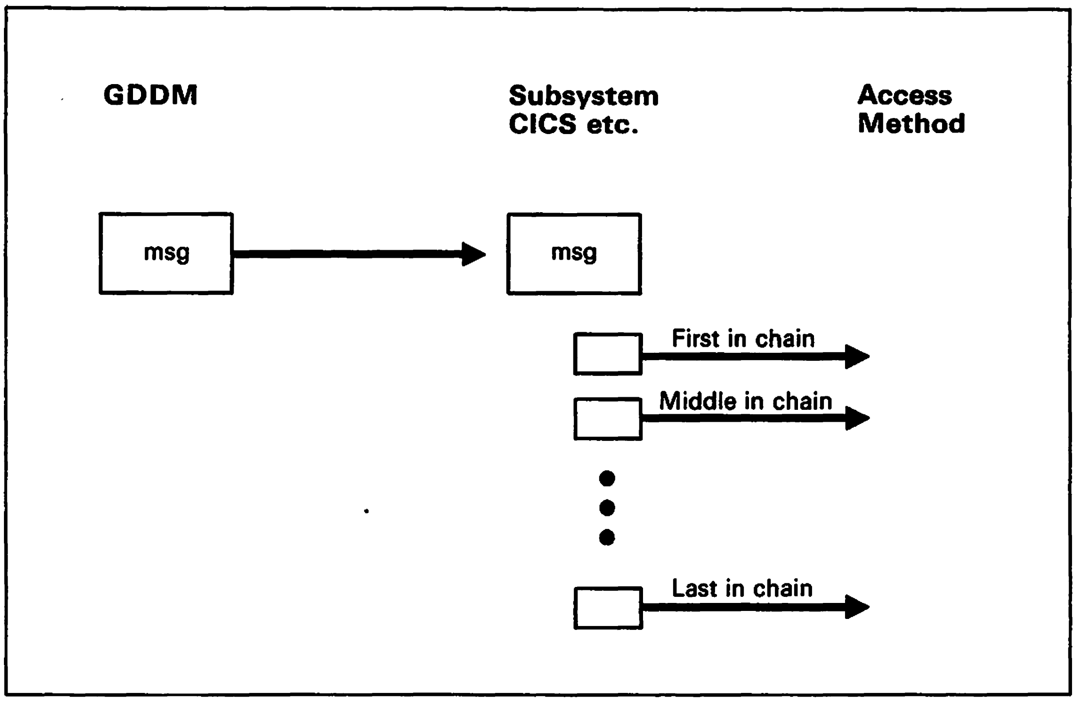


Figure 7. Data flow through an SNA system

By manipulating either the GDDM transmission buffer size (for non-SNA), or the subsystem buffer size (for SNA), we can generally prevent large messages causing problems in the remote network, although we have to perform some sort of balancing act because each message, regardless of size, has processor and link overheads associated with it.

Large messages are not such a problem for channel-attached terminals, so you should change the default to a large size (such as 4K bytes) to minimize terminal-access-method processing costs. You can do this by specifying the ADMMDFT IOBFSZ default.

The considerations that apply to IMS/VS non-SNA are special, and are discussed in "IMS/VS tuning – background information" on page 48.

Let's go back to the bank. The owner of the chain of shops realized that getting the teller to count all the money causes problems. So next time, the owner took **all** the shop managers to the bank with their respective takings. Have you ever waited behind 21 people paying in shop takings? It takes a long time to get served.

GDDM has the same problem. The data stream can be broken up into small messages by use of GDDM and subsystem buffer sizes, but because GDDM is producing them in rapid succession, when they get into the network it is just like the 21 shop managers. Each one takes only a short time to get served, but when they all arrive at the teller's window together, everyone behind them has to wait.

Then, the head teller had a great idea. He took the owner of the shops and the shop managers over to the waiting area. Then he told the owner of the shops to send the shop managers, one at a time, to the teller. Each shop manager would join the back of the queue only after the previous one had been served and had returned to the waiting area. Of course it took longer for all the shop managers to get served, but it pleased all the

Of course it took longer for all the shop managers to get served, but it pleased all the other customers because it meant that there was never more than one shop manager waiting in front of them in the queue.

We can adopt a similar approach to the problem of many GDDM messages arriving together at a node in the network that has a slow service time, such as a remote teleprocessing line.

In SNA networks, the solution is to use the SNA PACING and VPACING parameters. These can prevent GDDM from sending further messages until the first has been processed by the network. VPACING is not supported in the SNA TSO environment, but another, similar, mechanism exists. This is discussed in the TSO-specific performance section.

In the two non-SNA environments where the problem is likely to occur, CICS and TSO, GDDM offers a solution. A GDDM default signifies whether synchronized terminal I/O is to be performed in these two subsystems. This can be changed by specifying the ADMMDFT IOSYNCH=YES default. Like VPACING, this mechanism prevents GDDM from clogging the network; it achieves this by delaying acceptance of a second message for a user until the first has been handled by the network. The use of synchronized terminal I/O is discussed further in the subsystem-specific sections of Chapter 2, "Tuning and customizing by subsystem" on page 41.

Finding the optimum loading and packaging combinations

Which pieces of GDDM to load, and when and where to load them in processor main storage, are probably the most important performance decisions a system programmer has to face. You will have to choose a combination of loading and packaging options that suit both GDDM users and other users of your system.

GDDM consists of many hundreds of routines that are packaged together into several dozen family groups or load modules. Each load module manages one major GDDM function, like alphanumerics support or data stream processing. At execution time, GDDM resolves the addresses of the routines needed to process the calls issued by the application program. It does this in one of two ways:

1. Finding the routine already in storage because it has been previously loaded into the VM discontiguous shared segment (DCSS), OS pageable link pack area (PLPA), or the CICS partition/address space.

2. Loading the relevant GDDM load module into the application address space/partition/virtual machine.

It is possible to make a compromise, having some of the more commonly used GDDM code in a shared area with the others loaded as needed at execution time.

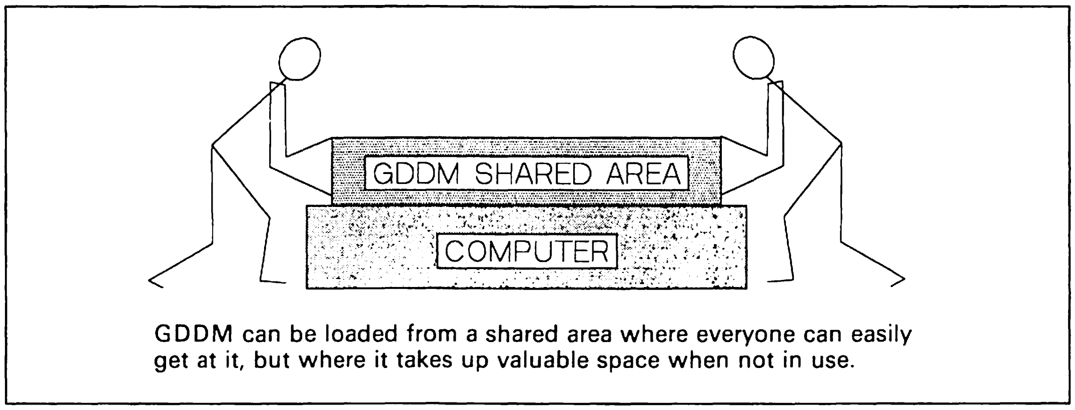
Whichever method is used, three options affect the timing of address resolution. The options are:

1. GDDM is loaded with the application. All addresses can be resolved on the first GDDM call.

2. GDDM is loaded at the first GDDM call in one big package. All addresses can be resolved on the first GDDM call.

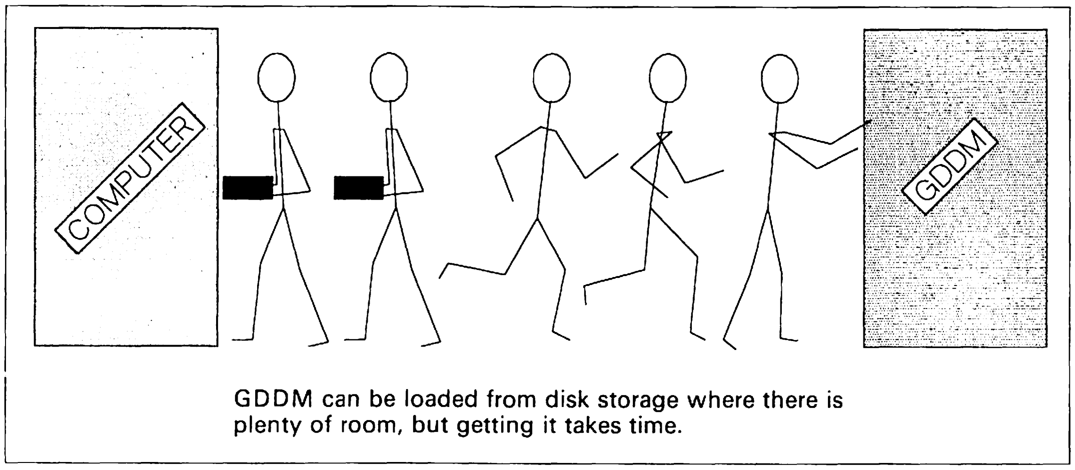
3. GDDM is loaded in several packages as calls are processed. Only the addresses of routines required to manage a particular call are resolved as the call is processed.

The loading and packaging options are illustrated in Figure 8, Figure 9, and Figure 10 on page 37.



GDDM can be loaded from a shared area where everyone can easily get at it, but where it takes up valuable space when not in use.

Figure 8. Loading options – keep GDDM in shared area



GDDM can be loaded from disk storage where there is plenty of room, but getting it takes time.

Figure 9. Loading options – keep GDDM in normal disk storage

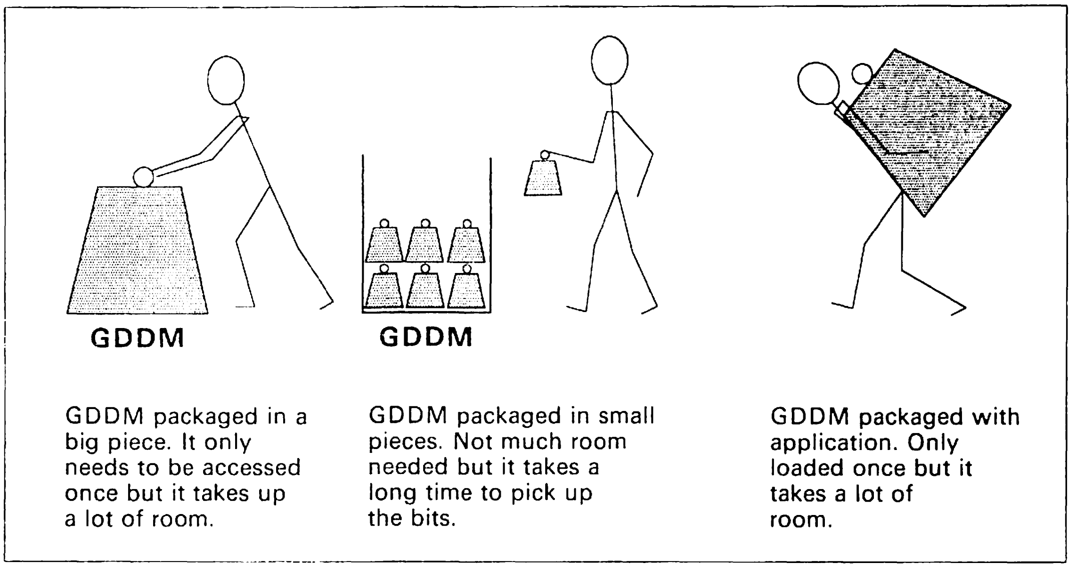

GDDM
GDDM
GDDM packaged in a big piece. It only needs to be accessed once but it takes up a lot of room.
GDDM packaged in small pieces. Not much room needed but it takes a long time to pick up the bits.
GDDM packaged with application. Only loaded once but it takes a lot of room.

Figure 10. Packaging options: Big package, small packages, or with applications

To configure GDDM into a large package requires some action on your part. The appropriate GDDM subsystem initialization module (ADME000x) (where "x" is C for CICS, I for IMS/VS, O for TSO, or V for VM) has to be link-edited with one or more GDDM packaging stubs. These stubs refer to the major functions in GDDM, so by doing this you are constructing a single GDDM load module that refers to all the GDDM routines that you think the application is going to need. Do not worry if you miss one out; the function will get dynamically loaded when required, but you may pay a performance penalty for doing this. It is probably reasonable to omit things like the ICU Help panels which might or might not be required at run time.

The following table summarizes the options you have for loading and packaging GDDM:

Loading options	Applicable packaging options
GDDM preloaded into shared storage	No packaging
	Link-edit GDDM modules together using packaging stubs
GDDM loaded at execution time	No packaging
	Link-edit GDDM modules together using packaging stubs
	Link-edit GDDM modules with the application using packaging stubs

Figure 11. Loading and packaging options

Chapter 4, "Repackaging for performance" on page 93 describes how the packaging stubs work and which ones to use, and shows you how to go about producing composite GDDM modules for different purposes.

Recommendations for loading and packaging combinations

This list gives the advantages and disadvantages of the possible combinations and makes a suggestion for each subsystem. Additional subsystem-dependent points will be discussed later in the performance information for individual subsystems.

1. GDDM preloaded into shared storage, not packaged at execution time: recommended for VM/SP environments.

   Advantages

   - One copy of the code can be shared by many users, thus saving on virtual storage requirements and potential page faults.
   - Because code is preloaded there is little disk fetch overhead at execution time.

   Disadvantages

   - The code occupies virtual storage even when no application is using GDDM.
   - The cost of finding the code in the shared area may be expensive in some environments.
   - If there are few GDDM users, the code may have to be paged in at execution time, causing problems in systems that are short of real storage.

2. GDDM preloaded into shared storage, packaged at execution time: recommended for TSO and IMS/VS environments that have many GDDM users.

   Advantages

   - One copy of the code can be shared by many users thus saving on virtual storage requirements and potential page faults.
   - Because code is preloaded, there is little disk fetch overhead at execution time.
   - The cost of finding the code in the shared area is reduced because only one load is issued.

   Disadvantages

   - The code occupies virtual storage even when no application is using GDDM.
   - If there are few GDDM users, the code may have to be paged in at execution time, causing problems in systems that are short of real storage.

3. GDDM loaded at execution time, module by module: recommended for CICS environments with frequent GDDM use.

   Advantages

   - GDDM does not occupy virtual storage while applications that execute it are not being used.

- Only the GDDM code required for the specific application may be loaded at execution time, again saving virtual storage.

Disadvantages

- There are potentially more page faults and greater use of virtual storage in environments where several copies of GDDM exist.
- Issuing multiple loads may severely degrade program execution time.

4. GDDM loaded at execution time, packaged into one module: recommended for TSO, IMS/VS and CICS environments that have little GDDM usage.

Advantages

- GDDM does not occupy virtual storage while applications that execute it are not being used.
- Only a single load operation takes place so response times are not severely degraded.

Disadvantages

- There are potentially more page faults and greater use of virtual storage in environments where several copies of GDDM exist.
- Unnecessary code might get loaded.

5. GDDM packaged with application.

Another option that is open to you is to link-edit GDDM with your application program so that they both get loaded together. This might be a useful move if you had only one or two programs that used GDDM and you did not want to keep GDDM on-line. But, usually, this technique does not have a great deal in its favor.

Chapter 2. Tuning and customizing by subsystem

This chapter describes what you, a system programmer, can do to tune GDDM. Chapter 1 described why you should do it. Scanning Chapter 1 should help you make sensible decisions about the suggestions that follow. To use this chapter for tuning you should look at the hints and tips for all subsystems, because you may be able to apply this information to ensure that GDDM is used efficiently in your installation. Then you should look at the particular subsystem or subsystems that you are interested in.

Hints and tips on efficient usage for all subsystems

When it comes to performance, computer systems are like automobiles. Simple design and regular tuning will help keep fuel consumption down, but if you want to get that extra 5% you have to drive thoughtfully. Such things as stepping your foot hard down on the gas when the lights turn green are out.

Details of application programming techniques are documented in Chapter 5, "Application programming for performance" on page 115. The following is a list of hints and tips that you might want to think about:

- Customize your 3270-PC/G and /GX work stations with sufficient segment storage for GDDM to run in retained mode. Even for output-only type applications, this may give significant performance advantages. Retained mode usually means shorter data streams and, therefore, less use of the terminal access methods like VTAM. See the table under "3270-PC/G and /GX segment storage requirement for GDDM" on page 67 for suggested values for different types of applications.

- Very complex pictures may take quite some time to draw in the work station. This can cause problems. While drawing, the work station does not communicate with the host operating system, and to the host it may seem that the terminal has gone out of service, and so it disconnects it.

  You can avoid this happening by resetting "timeout" values in your operating system. The "Things You Can Do" section for each subsystem later in this chapter tells you how to go about it.

- There are various GDDM default values that can affect performance. Here is a summary:

Default	*Meaning*
AM3270	Indicates how your 3270 terminals are attached. You can give GDDM a (small) helping hand by letting it know that you only have local SNA devices for example.
ICUFMDF	Determines the ICU defaults.
ICUFMSS	Determines the ICU symbol sets.
IMSUMAX	(IMS only) Specifies the maximum number of concurrent conversations to be supported.
IOBFSZ	Determines the GDDM transmission buffer size.
IOCOMPR	Indicates whether GDDM is to create compressed PS loads for character devices.
IOSYNCH	Indicates whether GDDM is to perform synchronized terminal I/O.
MAPGSTG	Defines the mapgroup storage threshold for GDDM run-time alphanumeric mapping.
SAVBFSZ	Defines the FSSAVE transmission buffer size for saved pictures.

- Processing option values can also be specified using GDDM defaults or in a DSOPEN parameter list. Those which can influence performance include:

Procopt	*Meaning*
FASTUPD	Indicates the picture update mode to be used for 3270-PC/G and /GX, 3179-G, and 5550 family displays. You can shorten data streams by specifying (FASTUPD,YES).
HRISPILL	Defines whether a spill file will be used to hold expanded GDF for composed-page print file generation. You can trade off host-processor time against storage by specifying (HRISPILL,YES).
HRISWATH	Defines the number of slices (swathes) that the picture is divided into for composed-page print file generation. You can again trade off host-processor time against virtual storage. The larger the number of swathes, the less virtual storage is needed. For 3800 or 3820 print file generation, the largest useful swathe count is the picture depth in inches multiplied by 8. For 4250 print file generation, the largest useful swathe count is the picture depth in inches multiplied by 19.
SEGSTORE	Defines whether GDDM operates in retained or non-retained mode for the 3270-PC/G and /GX. Local interactions are only possible if (SEGSTORE,YES) is specified or implied by default.

CTLMODE	Defines whether user control is available to the user. The default is (CTLMODE,YES). If user control is not to be used, specifying (CTLMODE,NO) causes a small decrease in dynamic storage requirement.
LCLMODE	Defines whether GDDM is to use a higher precision in the vectors that it sends to the 3270-PC/G and /GX work stations. Only specify if panning/zooming is to be used in user control.

Suggested settings for some of these are given in the "Things you can do" section for each subsystem later in this chapter.

A more detailed explanation of each default, whether it applies to your specific subsystem, and how to change defaults is in "GDDM Defaults and Nicknames" in the *GDDM Installation and System Management* manual appropriate to your subsystem.

CICS tuning – background information

Loading

CICS operates in a fundamentally different way from that of most of the other subsystems that GDDM runs under. All users execute programs in the same address space/partition. If these programs are reentrant, like GDDM, the same copy can be used by other transactions. Putting GDDM in the shared virtual area or a link-pack area gives no advantage unless you have more than one CICS region running GDDM transactions.

Program load under CICS can occur at two different times:

1. At system initialization, if RES=YES in the PPT entry

2. At program execution otherwise.

Resident programs stay permanently in the CICS address space/partition. Most programs that are nonresident are not deleted from the CICS address space/partition until a short-on-storage condition arises. Frequently used programs are therefore likely to be in main storage if they have been used previously, even if they are nonresident.

If a nonresident program has been deleted from main storage, it has to be reloaded from a library on DASD before it can be executed. This may increase transaction-response time. Declaring GDDM programs as resident will avoid this but will cause short-on-storage conditions to occur more often.

Under CICS 1.6.1, on MVS/XA most GDDM modules can be loaded above the 16-megabyte line. CICS does not delete such modules when they are loaded whether or not RES=YES is specified.

Packaging

In most CICS environments other than MVS/XA, there is little point in packaging GDDM modules together to form large composite CICS programs. In fact, there are two positive disadvantages:

1. No CICS program can be greater than 512K bytes. Some of the composite GDDM modules that you can create will exceed this.

2. At load time, sufficient real storage to contain the program has to be acquired. This may give paging problems where large programs are loaded in storage-constrained environments.

CICS will keep programs that are frequently used in main storage, even if they are nonresident, so packaging is unlikely to reduce loading delays.

These disadvantages do not apply on MVS/XA.

Controlling the data stream

The I/O synchronization option (the ADMMDFT IOSYNCH default) determines whether the WAIT option is put on the EXEC CICS SEND calls that GDDM issues to send data streams to the device.

This is useful for controlling teleprocessor-attached terminals in a non-SNA environment. It prevents a GDDM application from issuing a SEND until the response from a previous SEND has been returned from the access method to CICS. This technique helps to avoid increasing the response times for non-GDDM users on the same teleprocessing line in the non-SNA environment.

In an SNA environment, PACING and VPACING should be used to avoid increasing the response times for non-GDDM users on the same control unit.

The size of the GDDM SEND transmission is determined by specifying the ADMMDFT IOBFSZ default.

The size of each transmission from CICS to the terminal access method is governed by the BUFFER parameter in the CICS TCT. For SNA, the maximum size is 1536 bytes; so if GDDM SENDs a message that is larger than this, segmentation occurs. For non-SNA, the BUFFER parameter is automatically set to the size of the GDDM SEND, so messages are not segmented.

The program processing table

The number of entries in the program processing table (PPT) influences the performance of all transactions, because this table has to be scanned each time a transaction executes a program. IBM supplies four lists of GDDM PPT entries. One of these, the national languages list ADMUNPPT, consists of messages, menus, and help panels in each of the national languages other than US-English. You can delete the national language entries that you do not require. The following list shows the module suffix for each of the languages:

Suffix	*Language*
ADM. . . .B	Brazilian
ADM. . . .D	Danish
ADM. . . .F	French
ADM. . . .G	German
ADM. . . .H	Korean (Hangeul)
ADM. . . .I	Italian
ADM. . . .K	Japanese (Kanji)
ADM. . . .N	Norwegian
ADM. . . .S	Spanish
ADM. . . .V	Swedish

Controlling GDDM in the processor

GDDM will periodically give back control to CICS to allow any transaction with a higher priority to be dispatched. For GDDM transactions, you should specify low TRNPRTY values in the DFHPCT entries; for non-GDDM transactions, specify high ones. This will prevent GDDM from increasing the response times of non-GDDM transactions. There might well be an adverse effect on the GDDM response times, however.

CICS tuning – things you can do

Here are some suggestions and comments about some of the things you can do to tune CICS.

1. Declare GDDM programs as nonresident in the PPT.

2. There is little advantage to be gained in packaging GDDM in most CICS environments.

3. There is no advantage in putting GDDM in the VSE shared virtual area or in a link-pack area unless you have more than one CICS region executing GDDM transactions.

4. If you do package GDDM, remember that composite GDDM programs must not be greater than 512K bytes.

5. Do not put entries for the national language routines into the PPT if you are not going to use them.

6. Configure the 3274 to support data-stream decompression if it is TP attached and the line speed is 19 200 bits per second or less.

7. The size of the GDDM transmission buffer should be increased to 4K bytes for all local and SNA remote users by specifying the ADMMDFT IOBFSZ default.

8. For remote non-SNA GDDM users, consider specifying the GDDM transmission buffer size (the ADMMDFT IOBFSZ default). Do not make it larger than the normal value, which is 1 536. For remote SNA users, changing the BUFFER parameter in the CICS TCT table has the same effect.

   A large size means:

   - The response times of other remotely attached users will be degraded.

   - The response times for remotely attached GDDM users will be better.

   - There will be fewer calls to the terminal access method. Therefore, there will be lower processor utilization.

   A small size means:

   - Less impact on response times of other remotely attached users.

- Worse response times for remotely attached GDDM users.
- More calls to the terminal access method. Therefore, there will be higher processor utilization.

9. Control data streams in remote networks by using:

   a. EXEC CICS SEND WAIT for non-SNA, specified by the ADMMDFT IOSYNCH=YES default.

   b. PACING/VPACING and PASLIM for SNA.

10. Prevent GDDM from impacting non-GDDM transactions in the processor by use of low TRNPRTY values in DFHPCT.

11. If your graphics terminals are attached to a 3274-1D controller:

    a. For CICS/OS/VS: the Prepare To Read feature should be specified in the OS System Generation I/O device macro for each terminal.

    b. For CICS/DOS/VS: the mode parameter should be set to "05" in each terminal ADD statement in the IPL procedure.

12. See "Application programming under CICS" on page 124 for details of CICS pseudoconversational mode.

IMS/VS tuning – background information

The way that GDDM behaves is slightly different in an IMS/VS environment from the way it behaves under other subsystems. The principal differences are:

1. GDDM accesses a user data base, the SYSDEF data base, when it needs to determine the characteristics of devices used by an application. This is done because GDDM is not in direct communication with the display device and cannot, unlike other subsystem environments, issue a "query device." The information on the SYSDEF data base is referenced by using the logical terminal (LTERM) name of the device.

2. GDDM sends the data stream for the picture it has created by doing inserts (ISRTs) to the IMS/VS message queue. Under other subsystems, each of the GDDM output requests is treated as a separate message and is sent to the device as soon as possible by the subsystem. Under IMS/VS, they are treated as segments of a single message which is not sent to the device until GDDM has passed the complete data stream to IMS/VS and issued a PURGE. This difference has important performance implications.

3. Under IMS/VS, application programs cannot use the GDDM Device Input call ASREAD to read input from the display device. This means that GDDM or GDDM Presentation Graphics Facility (PGF) application programs cannot be interactive.

   Under IMS/VS, execution of a message processing program (MPP) is not bound to a specific LTERM. When an MPP associated with a specific transaction is executing, it must (if it is conversational) accept input from any terminal that can input the transaction. IMS/VS transactions can be written to be conversational by using a scratch pad area (SPA) to preserve the status of a conversational transaction for one LTERM. The nature of the architecture of GDDM, however, means that the status of a conversation is reflected in the contents of many tables and control blocks throughout the Message Processing Region (MPR), and this cannot be simply put into the SPA. Therefore, an application can use GDDM only in non-conversational mode. The interactive utilities are implemented under IMS/VS in a way that avoids this restriction. The mechanism is described later.

Output of GDDM data streams

The way that GDDM and IMS/VS interact during the transmission of GDDM-produced data streams has important performance implications. There are three separate operations involved:

1. GDDM sends the data stream to the message queue

   GDDM ISRTs the data stream as segments of a multisegment message to the output message queue. The following points should be noted:

   a. The size of the ISRT is controlled by the GDDM transmission buffer size (specified by the ADMMDFT IOBFSZ default).

b. With remote non-SNA transmission, the 3274 Model C has a restriction that no more than 3K bytes of uncompressed character definitions may be received in one transmission. Only GDDM knows where the data stream may be segmented to avoid this restriction, so it puts no more than 3K bytes of character definitions into a transmission buffer before an ISRT is made. For remote non-SNA, IMS/VS sends each ISRT as a separate transmission.

   It should be noted that only BTAM may be used for remote non-SNA.

c. It is advantageous to increase the GDDM transmission buffer size, to 4K bytes by specifying the ADMMDFT IOBFSZ default. From the preceding information it can be seen that this change will not be used to full advantage in the remote non-SNA environment.

d. Some tuning of the ADMMDFT IOBFSZ default to the long message queue length should be considered. This is because a message-queue element cannot be shared between segments of different messages. It is possible for a GDDM ISRT to span more than one message-queue element. To avoid wasting space in the message queue, make the GDDM transmission buffer size (the ADMMDFT IOBFSZ default) equal to, or a multiple of, the long message-queue logical-record length.

   The arithmetic for this must allow the space for the 4-byte "LLZZ" added by GDDM, and the space for appropriate headers in the message queue. See the description of the RECLNG parameter in the MSGQUEUE macro in the *IMS/VS Installation Guide.*

2. IMS/VS moves the message to buffer(s) in Communications I/O Pool

   The GDDM output data stream is held in the message queues until it is completed, when GDDM issues a PURGE call. It is then considered sendable, and a device-dependent module will be scheduled to convert the data stream into a device-specific form in the Communications I/O Pool (CIOP) OUTBUF buffers. For GDDM, Message Format Services (MFS) is bypassed. The transfer to the CIOP will occur when the device is available. The following points should be noted:

   a. Except with BTAM remote attachment, the blocking of the output data stream sent by IMS/VS is governed by the OUTBUF buffer size. The choice of the OUTBUF size is severely constrained by the protocol of attachment of the output device.

   b. For local non-SNA there can be only one OUTBUF buffer. This must be large enough to hold an entire output message, but must not be more than 32K bytes long. This means that there are GDDM graphic pictures that cannot be shown in the local non-SNA environment.

Unless there is some strong reason against, it is recommended that the maximum value (32K bytes) is used in the non-SNA environment. Violation of the maximum value causes the following messages to be displayed on the IMS/VS master console:

DFS0089I OUTPUT EXCEEDS BUFFER SIZE

LTERM xxxxxxxx NODE yyyyyyyy

DFS9989I VTAM NODE yyyyyyyy IS INOPERABLE

and causes the IMS/VS user terminal to be made inoperable. (Descriptions for the above messages are in the *IMS/VS Messages and Codes Reference Manual.*)

If this occurs, and the device token being used specifies COMPRES=NO, a way of reducing the length of the data stream is to use a different device token, which allows data stream compression (assuming the controller is configured for PS compression), that is, COMPRES=YES. See "Device Characteristics Tokens" in the *GDDM Installation and System Management for MVS* manual.

c. For BTAM remote attachment of GDDM devices, the value of OUTBUF is not used. In this case, IMS/VS uses the ISRT sizes as block sizes to avoid the 3K bytes restriction for the 3274 Model C described earlier.

3. IMS/VS uses the access method to send the message to the device

This is a regular function of IMS/VS. All that is unusual about it with GDDM is the occurrence of large data streams. You should consider the following:

a. With local non-SNA support, the entire GDDM data stream is put into one IMS OUTBUF buffer in the CIOP. When this is transmitted, it is done with one call to the access method (for example, one VTAM SEND). This can cause delays for other users of the channel if large data streams are sent. The channel is held busy while the control unit processes the entire GDDM data stream.

b. With BTAM remote support, IMS/VS does not communicate with any other devices on a BTAM line group during the transmission of a complete GDDM picture. In scheduling the device-dependent module, IMS/VS uses the concept of a communications ITASK, in controlling output to different devices. IMS/VS considers a complete BTAM line group as a resource for the communication task scheduling. This means that when a GDDM picture is being transmitted to one device of a line group, the other devices are locked out during the transmission of the entire picture.

As an example, for an average 6K bytes (compressed) GDDM graphic data stream, sent on a 4 800 bits per second line, the lockout would last approximately 10 seconds. For a complex 17K byte (compressed) data stream it would last approximately 30 seconds. If data-stream compression is not used, these times will double.

Note: This will seriously affect the performance of all terminals on the same BTAM linegroup.

Cross-domain considerations

The IMS/VS parameter OUTBUF defines the largest request unit (RU) that can flow from IMS/VS into the network. For IMS/VS-GDDM local non-SNA terminals, a value of 32K is recommended for OUTBUF.

In a cross-domain environment, the RU is sent through a 3705. The MAXDATA parameter on the network control program (NCP) must be big enough to handle the largest RU that can enter the network; in this case it is 32K bytes.

The product of two other NCP parameters (MAXBFRU * UNITSZ), should be greater than MAXDATA. When MAXDATA = 32 000, for example on local non-SNA terminals, this will lead to increased storage requirements in the host computer.

Use of nonrecoverable transactions

If GDDM transactions are inquiry-only, it is an advantage to make them nonrecoverable. This is done by specifying

INQ=(YES,NORECOVER)

as a parameter to the TRANSACT macro. If this is done, performance will be improved, because:

- The output data stream is not logged.
- In SNA transmission, the message-queue elements and CIOP buffers do not have to be held until the picture display at the device is completed. This can be important, particularly for remote attachment.

Message Processing Region (MPR) priority

Care must be taken with the class scheduling and resource management to ensure that GDDM applications are run in Message Processing Regions (MPRs) of a suitable priority. For example:

- In a processor-constrained system, it would be advisable to run GDDM applications in a low-priority MPR to avoid impacting the response times of other transactions.
- In a storage-constrained system, it may be better to run GDDM applications in medium- or high-priority MPRs to minimize the time for which storage is required.

Message queue and Communications I/O Pool sizes

GDDM data streams are large. This will increase the message queue and the CIOP space requirements. The following points should be considered:

- The message queues will be the bottleneck if there are delays in outputting messages to terminals, because the CIOP buffers will be allocated only when the communication path to the device is available.

- It may be prudent to use the terminals in RESPONSE mode to help regulate the flow of output.

- With local non-SNA terminals, the use of large 32K byte OUTBUFs for GDDM output will affect the CIOP utilization. Non-GDDM transactions that use MFS, however, will not cause the large OUTBUFs to be allocated.

- Message-queue and CIOP utilizations should be carefully monitored during the introduction of GDDM to an installation, and periodically thereafter.

The Interactive Chart Utility and symbol editors under IMS/VS

GDDM applications cannot be run as conversational transactions, because the architecture of GDDM makes it impossible to preserve the status of one conversation between inputs for that conversation. The Interactive Chart Utility and the symbol editors, however, must work in a conversational mode. To solve the problem of preserving status, the following technique is used:

- A utility scheduler program, ADMUTIL, is provided that runs in "wait for input" mode. A user who wants to use the Interactive Chart Utility, or one of the symbol editors, must enter the transaction code for the utility scheduler, with the name of the specific utility as a parameter.

- ADMUTIL then schedules a new "instance" of the utility which will be a subtask of the scheduler. Unique GETMAINed working storage will be obtained for the subtask, and the appropriate GDDM code will be loaded, or located in the LPA.

- Only one copy of the GDDM code will be used by all users.

- DL/I calls cannot be made from within a subtask, so the scheduler issues DL/I calls for the subtask.

- The code of the subtask is executed until a response from the terminal operator is required. It will pass control back to the scheduler, where a GU will be reissued. The next message is read from the message queue, and the LTERM is checked to see if it is either a request for a new instance of a utility, or a response from a terminal operator for an existing instance of a utility.

  If the input message is a response for an existing instance of a utility, the scheduler passes control to the correct instance of the utility. Otherwise, a new instance of a utility will be scheduled.

The following points affecting performance should be noted:

- The GETMAINed storage associated with an "instance" of a utility cannot be shared. This means that only a finite number of parallel conversations can be supported, before the storage capacity of the MPR is exceeded.

- All transactions for the utilities are routed through one MPR. The IMS/VS class scheduling cannot be used to balance the load on the MPRs, which may result in large variations in response time if there are many users of the interactive utilities. You can control the number of concurrent users with the ADMMDFT IMSUMAX default.

- In light use, the MPR is still occupied, because ADMUTIL runs in "wait for input" mode.
- It has been observed that the measurement of processor time by the IMS/VS Monitor program is affected by the tasking structure of the utility scheduler. The processor time recorded appears to be only that associated with the utility scheduler. The processor time for the "instances" of the utilities, which are attached as subtasks, is not counted.

IMS/VS tuning – things you can do

Here are some suggestions and comments about some of the things you can do to tune IMS/VS.

1. Put GDDM into a link-pack area if you have many potential GDDM users or if you are not short of real storage. Be aware that the overhead of finding modules there is governed by the number of user STEPLIBS that are accessed.
2. Package GDDM into a single load module to significantly reduce loading costs, even if the code is in a link-pack area.
3. Configure the 3274 to support data-stream decompression if it is teleprocessor-attached, and the line speed is 19 200 bits per second or less.
4. Carefully consider the performance implications of managing large output data streams in non-SNA environments, before installing GDDM in such environments.
5. Increase the size of the GDDM transmission buffer to 4K bytes, except for remote non-SNA environments, by specifying the ADMMDFT IOBFSZ default.
6. Tune the long message record size to the GDDM transmission/save buffer sizes.
7. Consider message-queue and CIOP space before GDDM is installed, and monitor after installation.
8. Consider running terminals in response mode to regulate the data flow.
9. Use resource-management and class-scheduling facilities to control processor usage by GDDM applications under IMS/VS.
10. Use nonrecoverable transactions when appropriate.
11. Review the teleprocessor-access-method parameters that can affect performance:
    - Choice of buffer sizes and RU sizes
    - PACING/VPACING and PASLIM parameters in SNA.

12. The use of a utility scheduler has enabled the GDDM interactive utilities to run under IMS/VS. However:

    - A dedicated "wait for input" MPR is required.

    - It has been observed that the measurement of processor time by the IMS/VS Monitor program is wrong for the MPR when the ADMUTIL scheduler is run.

13. Review the number of concurrent users that may be expected for the interactive utilities, in light of the performance constraints of the implementation. The number of concurrent users will be limited by:

    a. The storage space within the MPR. This depends on which utilities are used. The GDDM code is shared between instances of the utilities, so only one copy is required. There is also a dynamic storage requirement for each user.

    b. The response time for the MPR running ADMUTIL. This depends on many factors, such as the loading of the system, the relative priority of the MPR, and the hardware used.

14. Vary the ADMMDFT IMSUMAX default to overcome variable response times for the GDDM utilities which run under the ADMUTIL scheduler. IMSUMAX defines the maximum number of concurrent conversations with the scheduler. Its default value is 5.

15. If your graphics terminals are attached to a 3274-1D controller, the Prepare To Read feature should be specified in the OS System Generation I/O device macro.

TSO tuning – background information

This section contains several terms used in MVS/TSO tuning. If you need an explanation, further information is given in the *MVS Initialization and Tuning Guide.*

Controlling the data stream

The I/O synchronization option (the ADMMDFT IOSYNCH default) determines which of two types of TPUT is to be performed:

- TPUT NOHOLD (no synchronization of terminal I/O)
- TPUT HOLD (wait for a response from the terminal before sending any more data; that is, synchronize the I/O).

The TPUT HOLD option therefore regulates the data flow into the network from the GDDM application.

If the HOLD option is selected for a particular user, every TPUT issued will cause the address space to incur an OUTPUT WAIT SWAP until a response has been received from the terminal, when the address space will be swapped back in.

Another method of controlling the data stream is to use the MVS/TSO system options. The size of the TSO output buffer is set by the MAXRU parameter in the LOGMODE table for SNA (maximum 1536 bytes); for non-SNA it is the same size as the application TPUT, that is, no segmenting of data occurs.

The number of buffers available to manage TSO terminal output is defined on a global basis in TSOKEY00 for VTAM; this is a member in SYS1.PARMLIB. The philosophy of output-buffer handling for VTAM is to specify a maximum amount of buffer space that can be in use for output by an address space. This is defined by HIBFREXT, which is the amount of buffer space. When this level is reached, MVS puts the address space into an OUTPUT WAIT SWAP, until the amount of buffer space builds up to a minimum level (LOBFREXT), when a new TSO transaction is started in the first period. Buffer space is returned to the free buffer pool as the positive responses are received from the terminal on receipt of the output.

For example:

Assume:	TSO output buffer size = 1 024 bytes
	HIBFREXT = 4K bytes
	LOBFREXT = 2K bytes

The ADMMDFT IOBFSZ default buffer size = 1 024 bytes.

If GDDM does four TPUTs in succession, 4K bytes of output buffers will be used, and the high buffer extent will be reached. The address space will be swapped out until 2K bytes of buffers have been freed by the receipt of two positive responses.

This mechanism may therefore come into effect if large amounts of data are being given to the access method. It would probably not be seen with locally-attached terminals, because responses would be returned rapidly, but may come into play in the remote environment if the TPUT NOHOLD option is used. Therefore, you can use it to control the amount of output data entering the network from one user, and to produce less swapping than the TPUT HOLD option. However, great caution should be exercised if this type of control is imposed on a TSO system, because the size of output buffers and the high/low extents are global. Other terminal output will be subject to the same constraints, so the high and low buffer extents should be chosen to allow other TSO users to run without incurring excess swapping.

The I/O synchronization option offered by GDDM can be applied to individual users by giving each an External Defaults File. It is likely that these users will incur more SWAP activity than would be the case using the HI/LOBFREXT technique, although total system swaps should be fewer.

For remote GDDM terminals not on dedicated lines it is probably advisable to use the TPUT HOLD option if you want to prevent response times increasing unduly for non-GDDM users.

Swapping

SWAP activity may become significant to the total system utilization, particularly if the GDDM output buffer size (the ADMMDFT IOBFSZ default) is small and the graphics output is complex. This will cause swap instruction processing greater than would usually take place in TSO, as follows:

SWAP OUT/IN PAIR = APPROX. 83K INSTRUCTIONS

(This assumes a swap page rate of 12, and 3 pages per START I/O.
See the *MVS Performance Notebook*, for details.)

For a sample benchmark data stream:

Total compressed data-stream length	= 6 718 bytes
GDDM/TSO output buffer sizes	= 1 024 bytes
Number of transmissions	= 7

TOTAL ADDITIONAL SWAP INSTRUCTIONS/PICTURE = 581K

The increased swapping will produce contention both for processor cycles and for the DASD subsystem which, depending on the existing workload, may be critical.

This swapping can theoretically be reduced by use of logical swapping; however, in a heavily loaded system, most swaps will be physical.

If an address space incurs an "output wait swap" because of buffer utilization, a new transaction will be initiated when processing resumes. The statistics on the number of transactions will therefore be distorted, and there will be greater contention for the MPL in the first period.

TSO performance groups

The addition of GDDM transactions to an existing TSO system should lead you to reevaluate the IPS, particularly if more than one or two graphics terminals are going to be running concurrently with other TSO users. GDDM transactions are not insignificant, and will increase contention in the system, particularly for the active-address-space limits specified in the MPL. Swapping will increase, particularly if TPUT HOLD is specified or if the maximum amount of buffer space is used. This will affect the total processor utilization and the response time. Paging activity may also increase; the contention for real storage may increase the swap and page data-set response.

Two alternatives are available to minimize the impact of graphics on the system:

- Increase the MPL
- Run GDDM in a different PGN and DOMAIN.

The second alternative has several advantages. It allows increases in the number of graphics terminals to be absorbed by the system with minimal impact on other users; the IPS for the GDDM PGN can be tuned separately; and the typical graphics transaction for an installation can be monitored. Both alternatives may have real-storage implications; increasing the MPL to allow more resident TSO transactions may cause the paging rate to increase to unacceptable levels.

TSO tuning – things you can do

Here are some suggestions and comments about some of the things you can do to tune TSO.

1. Put GDDM into a link-pack area if you have many potential GDDM users or if you are not short of real storage. Be aware that the overhead of finding modules there is governed by the number of user STEPLIBs that are accessed.

2. Package GDDM into a single load module, because this significantly reduces loading costs even if the code is in a link-pack area.

3. Configure the 3274 to support data stream decompression if it is teleprocessor-attached, and the line speed is 19 200 bits per second or less.

4. Increase the size of the GDDM transmission buffer, the ADMMDFT IOBFSZ default, to 4K bytes for all local and SNA remote users.

5. For remote non-SNA GDDM users, consider changing the GDDM transmission buffer size, the ADMMDFT IOBFSZ default. Do not make it larger than the normal value which is 1 536. For remote SNA users, changing the MAXRU parameter in the LOGMODE table has the same effect.

   A large size means:

   - The response times of other remotely attached users will be degraded.

   - The response times for remotely attached GDDM users will be better.

- There will be fewer calls to the terminal-access method, so lower processor utilization.

A small size means:

- Less impact on the response times of other remotely attached users
- Worse response times for remotely attached GDDM users
- More calls to the terminal-access-method, so higher processor utilization.

6. Control the data streams in remote networks by using either:

   a. TPUT HOLD, specified by the ADMMDFT IOSYNCH = YES default, which can apply to individual users, or

   b. HI/LOBFREXT mechanism, which is system-wide.

7. Control GDDM in the processor by either:

   a. Increasing the MPL, or

   b. Running GDDM in a different PGN and DOMAIN.

8. If your graphics terminals are attached to a 3274-1D controller, the Prepare To Read feature should be specified in the OS System Generation I/O device macro.

VM/SP tuning – background information

Discontiguous saved segment (DCSS)

The largest potential performance improvement in a VM/SP environment is obtained by putting GDDM into a discontiguous saved segment (DCSS). It has been observed that response times for the initial GDDM display in an application can be reduced by several tens of seconds when a DCSS is used. This is discussed in the previous chapter. See "Recommendations for loading and packaging combinations" on page 38.

There are several reasons GDDM code in a DCSS might not be used even if you intend it to be:

1. The name of the saved segment may be wrong. The GDDM Version 2 Release 1 names are:

GDDM-Base	ADMBA210
GDDM-PGF	ADMPG210
GDDM-IMD	ADMIM210

   or

GDDM-Base	ADMASS00
GDDM-PGF	ADMPG000
GDDM-IMD	ADMIM000

   These are different from the Version 1 Release 4 name ADMASS00, the Version 1 Release 3 name ADMASS30, and the Version 1 Release 2 name ADMASSSV.

2. The starting address of the DCSS may overlap with addresses in the user's virtual machine. This could well occur if virtual machine storage size gets increased for any reason.

3. Explicitly including the GDDM packaging stub ADMUX00V when loading the application prohibits the use of most of the GDDM code in the DCSS.

4. This applies to 43xx Processors only.

   When a DCSS is protected (the default) the VM Command Processor (CP) must scan through all pages of the DCSS each time a virtual machine using it stops being dispatched. This is done to check if the change bit is on in a page. You can avoid this by coding PROTECT=OFF on the NAMESYS.

   You should only use this technique if your applications have all been thoroughly tested. If not, you may find that the saved segment gets corrupted by someone doing wild stores.

On 308x and 3090 machines, CP uses the segment protect hardware rather than the scanning approach.

Time-outs for 3179-G, 3270-PC/G and /GX, 4224, and 5550 devices

While 3179-G color display stations, 3270-PC/G and /GX work stations, or 4224 printers are drawing they have no contact with the host. Although picture draw time is usually very short, for very complex pictures it can take several tens of seconds. Here, VM may act as if the terminal has gone out of service and disconnect it. You can vary the length of time that VM will wait before disconnecting the terminal with the

```
CP SET MITIME GRAF time
```

command. This specifies the number of seconds that VM will wait after a "missing interrupt" before disconnecting the terminal.

The default setting is 30 seconds. If you are drawing very complex pictures, particularly on the 3179-G or 3270-PC/G, which take longer to draw pictures than the 3270-PC/GX, then you should increase this. A figure of 90 seconds is suggested as a good starting point.

Chapter 3. Storage requirements and capacity planning

This chapter tells you how to estimate how much of the various system resources your GDDM users will require. Read it to find out how to calculate:

- The amount of DASD space required for GDDM objects.
- The amount of virtual storage required by the various GDDM functions in the different subsystem environments.
- How much segment storage to allocate to the GDDM host session running in the 3270-PC/G or /GX.
- The processor/link/3274 utilizations caused by GDDM for:
  - Mapped and procedural alphanumerics applications.
  - Graphics applications executed on 3179-G, 3279, 3287, 3270-PC/G and /GX, or 5550 family.

Information on how much DASD space is needed for the GDDM libraries is contained in the installation section for each operating system.

GDDM objects and use of DASD space

GDDM users can produce several objects (such as saved charts) from application programs that use GDDM or from the GDDM interactive utilities. These objects require direct-access storage, and you may find it necessary to produce a system to manage them and to prevent unauthorized access to them. The objects that can be produced are:

- ADMGDF files
- Alphanumeric maps in various forms
- Chart data files
- Chart definition files
- Chart format files
- Composed-page printer files
- Image data files
- Projection definition files
- Saved picture (ADMSAVE) files
- Symbol sets.

Figure 12 on page 62 shows their contents, the features of GDDM with which they are associated, how many you may expect your users to produce, and their likely size. As you can imagine, the last three of these can vary enormously according to the use made of GDDM, and are therefore necessarily approximate, but they should give you a figure that you can use for estimating.

Object type	**Contents**	**Produced by**	**How Many**	**Average Size of each object**
ICU				
Chart data	Data values for ICU Charts	GDDM-PGF ICU	Many: 20 per user	4*400-byte records
Chart format	Formats for ICU charts	GDDM-PGF ICU	Many: 15 per user	4*400-byte records
ICU data definition	Data extraction rules	GDDM-PGF ICU	Many: 5 per user	3*400-byte records
Symbol sets				
Image	Dot-pattern symbols for logos & special characters (See Note 1)	GDDM Base Image Symbol Editor	Not many: 0.5 per user	10*400-byte records
Vector	Line symbols for logos & special characters (See Note 1)	GDDM-PGF Vector Symbol Editor	Not many: 0.5 per user	20*400-byte records
Saved picture (ADMSAVE files)	GDDM pictures held in device-dependent data-stream format	GDDM Base programs	Not many 0.5 per user	40*400-byte records
Saved GDF (ADMGDF) files)	GDDM pictures held in device-independent Graphic Data Format	GDDM Base programs and ICU	Many: 10 per user	20*400-byte records
Composed-page print files	GDDM pictures held in device-dependent data stream format	GDDM Base programs	Not many: 0.5 per user	500*137-byte records (3800/3820) 100*2000-byte records (4250)

Figure 12 (Part 1 of 2). GDDM object contents and size table

Object type	**Contents**	**Produced by**	**How Many**	**Average Size of each object**
Maps				
MSL	Library of maps in a form for editing	GDDM-IMD	Not many: 1 per IMD user +2 or 3 common. Many maps per MSL	8*256-byte records per map
Generated mapgroups	Maps in usable form for execution	GDDM-IMD	Not many: 1+ per mapping program	4*400-byte records
Generated ADS	Data structures that correspond to maps	GDDM-IMD	Not many: 5 per mapping program	50*80-byte records
Image				
Projections (ADMPROJ)	Projection Information	GDDM programs	Not many: 0.5 per user	(2-5)*400-byte records
Data (ADMIMG)	Image Information	GDDM programs	Many: 10 per user	(150-350)*400-byte records for documents
Note 1. This applies to user-defined symbol sets only.				

Figure 12 (Part 2 of 2). GDDM object contents and size table

Virtual storage requirement of GDDM

The GDDM virtual storage requirement consists of two elements:

1. The GDDM code size
2. The dynamic storage requirement for each GDDM user.

The code requirement is dependent on the functions that your applications use and the devices on which they execute. Device type also influences the user dynamic storage requirement, although picture content is a more critical factor.

There are ways of reducing the code storage requirements by selectively loading parts of GDDM. Chapter 1, "Performance background" on page 1 gives you an overview of the techniques available. Chapter 4, "Repackaging for performance" on page 93 gives a detailed explanation of how to go about it with a more precise sizing of each of the GDDM functional areas.

Figure 13 gives you approximate minimum sizes for a single user, in an address space/partition, performing various functions under GDDM. Treat it with some caution (particularly the User Dynamic Storage requirement) as usage varies enormously in different installations. We do **not** guarantee the figures.

You can reduce your use of virtual storage by putting GDDM in a shared area like a discontiguous shared segment in VM/SP or a pageable link-pack area in OS. In this case, the virtual storage requirement of the GDDM code determines the size of the shareable area that GDDM will occupy.

In the MVS/XA environment, most of the GDDM code and some dynamic storage can be placed above the 16 megabyte line.

GDDM Base programming	1 700K + 60K per additional user
GDDM-PGF Programming	1 900K + 80K per additional user
ICU	2 200K + 90K per additional user
GDDM-IMD	1 200K + 60K per additional user
Image Symbol Editor	1 500K + 60K per additional user
Vector Symbol Editor	1 700K + 60K per additional user

Calculating the size you will need
To calculate the size of the address space, region, or virtual machine you will need, subtract any items held in shared storage such as the VM/CMS shared segment, and add storage for other system requirements.

Method used to arrive at these figures
These figures were calculated using the table in the next figure and adding 10%. You should understand that the User Storage is dependent on picture content and can vary considerably. Numbers quoted here are our best guess for "typical" GDDM usage.

There are various ways of reducing the code requirement. Read Chapter Chapter 1 for more information.

Figure 13. GDDM single user minimum virtual storage requirement (in bytes)

Figure 14 shows a more detailed version of virtual storage requirements. The figures in the table apply to all environments that support GDDM unless otherwise shown in the following notes.

If one copy of GDDM is shared among multiple environments in one processor, for example, between TSO and IMS/VS, 60K bytes should be added to the code storage requirement for the second and each subsequent environment supported.

Of necessity, all figures are approximate and are for guidance only.

GDDM Function	Code Size	Dynamic Storage per user 3279/3287	Dynamic Storage per user 3179-G, 4224 3270PC/G, /GX 5550	Dynamic Storage per user 3193
a. Base GDDM Alphanumerics	850K	20-30K	20-30K	20-30K
b. Base GDDM graphics	650K + a (See note 1)	Typically 40-90K. Likely maximum 250-350K	20-60K. Likely maximum 200-300K	Not applicable
c. Base GDDM image	70K + a	Typically 150-200K. Likely maximum 300K	Typically 150-200K. Likely maximum 300K	50-60K direct transmission, 150-200K indirect transmission.
d. Presentation Graphics Routines	220K + a + b	60-100K (See note 2)	40-60K (See note 2)	Not applicable
e. The Interactive Chart Utility	390K + a + b + d (See note 3)	100-150K (See note 2)	60-80K (See note 2)	Not applicable
f. The Vector Symbol Editor	210K + a + b	40-60K	30-50K	Not applicable
g. The Image Symbol Editor	130K + a + b	40-60K	40-60K	Not applicable
h. The Print Utility	10K + a + b	40-80K (See note 2)	40-80K (See note 2)	Not applicable
i. Run-time Mapping	55K + a	20-40K	20-40K	Not applicable
j. Interactive Map Definition	375K + a + h	40-50K	40-50K	Not applicable

Example. To calculate the size of Presentation Graphics Routines add: Presentation Graphics Routines 220K, + Base GDDM with Alphanumerics 850K (a), + GDDM Graphics 650K (b) = 1720K total. (Note that each item is only included once.)

Note 1. 650K is obtained using first level packaging stubs. Using second level packaging stubs to omit unwanted items like high resolution printing, can reduce this value. Letting GDDM load dynamically reduces the figure further. For example, for an output-only graphics application on a display terminal the size can be reduced to around 490K.

The packaging stubs are described in the appropriate *GDDM: Installation and System Management Guide* for your system.

Note 2. User storage size depends on picture type; in particular on how many graphics primitives (lines, arcs and so on) are used. Typical values are quoted. Maxima can be much higher see item (b). In particular enabling quality defaults in the ICU, enabling user control, and use of step-by-step charts in the ICU will increase storage requirements.

Note 3. Add 350K if ICU US English Help panels are packaged with GDDM.

Figure 14. GDDM virtual storage requirement (in bytes)

3270-PC/G and /GX segment storage requirement for GDDM

To fully exploit the capabilities of the 3270-PC/G and /GX work stations, GDDM must be able to operate in retained mode to these devices. This is the default case, if sufficient segment storage has been allocated to the GDDM host session in the work station.

Besides the increased function that retained mode operation offers, there are also potential performance advantages, particularly in the area of data-stream reduction.

The following table should be used as a starting point for 3270-PC/G and /GX segment storage allocation:

GDDM picture type	*Segment storage size (see Notes)*
Business charts	5–15K
General graphics	10K
Scientific and engineering	5–30K
Data-driven pictures	30–60K
Images	200–350K

Note 1.

The above figures do not include the costs of characters from any nondefault GDDM symbol sets that might be used in the picture. Allow 10K bytes for each GDDM-supplied symbol set. User-defined symbol sets may need more than this.

Note 2.

The range quoted for data-driven pictures (for example, complex cartographic or seismological applications) is thought to be reasonable. You should be aware, however, that there is no limit to the size of picture that GDDM can create for display on these work stations, excepting those imposed by the processing power and storage available in the mainframe host processor.

Note 3.

When using operator windows, the highest priority operator window has first choice of the symbol set storage. Any other operator windows get the remaining storage, in order of viewing priority.

Capacity planning – first-pass method

This section shows you a simple way of estimating how much your system resources will be used by GDDM and its associated applications.

You should already have decided how much resource you have available to devote to GDDM applications in terms of processor, link, and 3274 controller use.

The technique is intended to provide a "first pass" assessment of the impact of GDDM on these resources. If it shows that the proposed GDDM applications will take more than 50% of the available resources, you should investigate further using the more detailed approach outlined in "Capacity planning – detailed method" on page 76. Neither of the methods is intended to be an overall system performance estimating technique.

The aim of the process is to help you estimate the use, by GDDM and associated applications, of the following resources:

- The host processor
- The communications link:
  - The 3274 for local terminals
  - The 3274 and teleprocessing line for remote terminals.

Because of the wide variety of ways of using GDDM, it is emphasized that any answers obtained will only be approximate for two reasons:

- The approximate nature of the data and estimating techniques.
- The difficulty of defining the nature of screen output, particularly pictures.

Disclaimer

IBM does not guarantee that any of the results obtained by the methods described below will accurately match a real situation.

The data in the tables are for use in determining the approximate use of resources by transactions involving GDDM. They do not necessarily reflect the relative performance of the different controllers or processors for any other transaction type.

Applications and devices covered

The scope of this capacity planning exercise is limited to four major GDDM functions. These are:

Procedural alphanumeric output
Mapped alphanumeric input and output
Graphic output
Interactive Chart Utility.

The devices that can be associated with these functions, for this discussion, are:

For mapped and procedural alphanumerics: all members of the 3270 and 3270-PC families.
For graphics and the ICU: 3279 and 3287 (with PS support), 3179-G, and 3270-PC/G and /GX work stations.

Overview of the "first-pass" method

The main steps necessary to make the estimate are:

1. Analyze the interactions between terminal and computer in each GDDM application except the ICU, as follows:

   a. Divide the interactions into different types, for example, simple alphanumeric, complex graphic, and so on.

   b. For each class of interaction, estimate the number of interactions per hour.

2. Decide how many concurrently active ICU users there are likely to be.

3. Use the table provided to work out the resource usage of each of the interaction types in hypothetical processing units.

4. Calculate the actual resource usage by converting the figures arrived at into processing units for the real system components that interest you.

The method you use

The estimating technique revolves around identifying the GDDM-associated interactions between terminal user and the computer system.

An interaction is an input into the host computer by a terminal user to which the host computer responds. Pressing the ENTER key to send a transaction name, and then receiving a message back from the host, is an example of an interaction. Typing in the transaction name without pressing ENTER is not; no communication between terminal and host is involved in this. Pressing PF5 in the Interactive Chart Utility and the subsequent picture display is another example of an interaction.

Several interactions can be generated by a single application program or transaction.

The actual steps of the method are:

1. Decide which elements of your system you want to examine, processor, links, or 3274s. If you are interested in one particular link or 3274 control unit, you should identify GDDM applications that use that particular resource.

2. Examine the GDDM applications that you are interested in and calculate the number of interactions between terminal and computer that they contain. You can ignore interactions generated by Interactive Chart Utility usage for the moment.

3. Divide the interactions into four groups:

   a. Alphanumerics
   b. Local (uncompressed) 3279/3287 graphics
   c. Remote (compressed) 3279/3287 graphics
   d. 3270-PC/G and /GX graphics.

   For each group, use the relevant table, Figure 15 on page 71 for alphanumerics, Figure 16 on page 72 for 3279/3287 graphics or Figure 17 on page 73 for 3270-PC/G or /GX graphics, to divide the interactions into classes.

4. Calculate the total number of hourly interactions for each class and put the figure into column 1 in the estimating table against the relevant interaction type.

5. Calculate how many concurrently active ICU users will be on the system and which graphics devices they will be using. Again, for 3279s and 3287s, subdivide the users into two groups, local (uncompressed graphics data streams) and remote (compressed graphics data streams). Put the answers in column 1 of the relevant estimating table against interaction type 10.

GDDM estimating table (part 1) – alphanumerics							
Class of interaction		Processor		Link		3274	
	Rate	Multi-plier	Hourly value	Multi-plier	Hourly value	Multi-plier	Hourly value
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
1. Very simple procedural alpha less than 10 fields out/in	. . .	5.0 to 6.0	. . .	1.5 to 4.0	. . .	1.5 to 4.0	. . .
2. Very simple mapped alpha less than 10 fields out/in	. . .	6.0	. . .	1.5 to 4.0	. . .	1.5 to 4.0	. . .
3. Simple or average procedural alpha 10–50 fields out/in	. . . . . .	8.0 to 14.0	. . . . . .	8.0 to 10.0	. . . . . .	8.0 to 10.0	. . . . . .
4. Simple or average mapped alpha 10–50 fields out/in	. . .	6.0 to 7.0	. . .	4.0 to 10.0	. . .	4.0 to 10.0	. . .
5. Complex procedural alpha 50–200 fields out/in	. . .	14.0 to 30.0	. . .	10.0 to 15.0	. . .	10.0 to 15.0	. . .
6. Complex mapped alpha 50–200 fields out/in	. . .	7.0 to 8.0	. . .	10.0 to 15.0	. . .	10.0 to 15.0	. . .
Totals	xxx	xxx	. . .	xxx	. . .	xxx	. . .

Figure 15. GDDM estimating table (part 1) – alphanumerics

Instructions: Fill in the spaces marked " . . . " (Column 1 is the hourly interaction rate and comes from your observations. Columns 2, 4, and 6 contain ranges of values for processor, link, and 3274 associated with each of the interaction classes described. Pick a value from each range depending on where your interaction fits in relation to the interaction description. Columns 3, 5, and 7 are filled in from the values you picked from the preceding columns multiplied by the hourly rate from column 1.)

Finally calculate the totals (. . .) in the bottom row of the table.

GDDM estimating table (part 2) – Graphics and the ICU for 3279 and 3287							
Class of Interaction		Processor		Link		3274	
	Rate	Multi-plier	Hourly value	Multi-plier	Hourly value	Multi-plier	Hourly value
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
7. Simple graphics Simple base GDDM pictures, charts with < 20 points	Uncompressed						
	. . .	120	. . .	60	. . .	85	. . .
	Compressed						
	. . .	130	. . .	27	. . .	180	. . .
8. Medium graphics Medium base GDDM pictures, charts with 20–50 points.	Uncompressed						
	. . .	270	. . .	150	. . .	210	. . .
	Compressed						
	. . .	290	. . .	70	. . .	450	. . .
9. Complex graphics Complex base GDDM pictures, charts with > 50 points.	Uncompressed						
	. . .	480	. . .	250	. . .	350	. . .
	Compressed						
	. . .	510	. . .	110	. . .	750	. . .
10. Very complex graphics, for example, cartographics, engineering drawings that have 2000–20000 lines.	Uncompressed						
	. . .	500 to 5000	. . .	500	. . .	600	. . .
	Compressed						
	. . .	530 to 5300	. . .	350	. . .	1000	. . .
11. ICU – Enter number of ICU users, not the rate of interaction	Uncompressed						
	. . .	10000	. . .	5000	. . .	6000	. . .
	Compressed						
	. . .	10200	. . .	3500	. . .	10000	. . .
TOTALS	xxx	xxx	. . .	xxx	. . .	xxx	. . .

Figure 16. GDDM estimating table (part 2) - graphics and the ICU for 3279 and 3287

Instructions: Fill in the spaces marked " . . . " (Column 1 is the hourly interaction rate and should come from your observations. Differentiate between compressed and uncompressed graphics interactions, because each has different resource requirements. Columns 2, 4, and 6 contain values for processor, link, and 3274 associated with each of the interaction classes described. Columns 3, 5, and 7 are filled in from the values from the preceding columns multiplied by the hourly rate from column 1.)

Finally calculate the totals (. . .) in the bottom row of the table.

GDDM estimating table (part 3) for resource calculation – Graphics and the ICU for 3270-PC/G and /GX, and 3179 G							
Class of interaction		Processor		Link		3274	
	Rate	Multi–plier	Hourly value	Multi–plier	Hourly value	Multi–plier	Hourly value
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
7. Simple graphics Simple base GDDM pictures, charts with < 20 points	. . .	60	. . .	20	. . .	20	. . .
8. Medium graphics Medium base GDDM pictures, charts with 20–50 points.	. . .	110	. . .	50	. . .	50	
9. Complex graphics Complex base GDDM pictures, charts with > 50 points.	. . .	200	. . .	100	. . .	100	
10. Very complex graphics for example, cartographics, engineering drawings that have 2000–20000 lines.	. . .	200 to 2000	. . .	100 to 1000	. . .	100 to 1000	. . .
11. ICU – Enter number of ICU users, not the rate of interaction	. . .	6000	. . .	2500	. . .	2500	. . .
TOTALS	xxx	xxx	. . .	xxx	. . .	xxx	. . .

Figure 17. GDDM estimating table (part 3) for resource calculation

Instructions: Fill in the spaces marked " . . . " (Column 1 is the hourly interaction rate and should come from your observations. Columns 2, 4, and 6 contain values for processor, link, and 3274 associated with each of the interaction classes described. Columns 3, 5, and 7 are filled in from the values from the preceding columns multiplied by the hourly rate from column 1.)

Finally, calculate the totals (. . .) in the bottom row of the table.

Obtaining the results

When you have filled in the estimating tables above, you will have found the total processor, link, and 3274 costs for the GDDM applications in which you are interested. Add together the respective values for processor, link, and 3274 that you have obtained from using each part of the estimating table.

The next step is to convert these totals into resource usages that apply to your particular computer system.

How to get processor utilization values

GDDM processor costs are higher in some subsystems than others. The processor total you have obtained should be modified accordingly:

- Under CICS, GDDM has to give up control voluntarily. Add 10% to the processor total to allow for this.

- In TSO, output wait swaps will occur after screen output. This will add to processor costs. A rough estimate of the extra cost is:

  ```
  ( LINK total ) * 0.67
  ```

  Add this value to the processor total.

To obtain processor use divide the number you have by the relevant processor factor from the following table:

Processor type	Divisor		Processor type	Divisor
4321-1	780		3031-AP	7 300
4331	780		3032	9 900
4331-11	1 300		3083-S	10 700
4331-G2	1 500		3083-E	13 800
4341-9	1 900		3033-EX	14 700
4341-10	2 700		3033-N	15 100
4341	3 100		3033-U	17 900
4341-11	4 000		3033-B	19 900
4341-G2	5 000		3033-BX	21 200
4341-12	5 700		3033-J	25 600
4361-4	3 400		3033-JX	27 300
4361-5	4 300		3033-AP	31 800
4381-1	10 400		3033-MP	31 800
			3081-D	33 100
			3081-G	37 000
			3081-GX	39 900
			3081-K	45 900
			3081-KX	50 500
			3084	87 200
			3084-QX	94 200
			3090-2	101 000
			3090-4	170 000

Figure 18. Table of divisors for different processors

You should now have a number between 0 and 100. This is the percentage processor use caused by GDDM. If this number is greater than 50% of the spare processor capacity that you have available for GDDM, use the detailed capacity-planning technique in "Capacity planning – detailed method" on page 76 to analyze your applications in more depth.

How to get link utilization values

Take the final total in column 5 of the Estimating Table and divide it by (linkspeed*36) where linkspeed is the speed of your communications link in bytes per second.

Multiply the result by 100. You should now have a number between 0 and 100. This is the percentage communications link use caused by GDDM.

If this number is greater than 50% of the spare link use that you have available for GDDM then use the detailed capacity-planning technique in "Capacity planning – detailed method" on page 76 to analyze your applications in more depth.

How to get 3274 controller utilization values

Take the final total in column 7 of the Estimating Table and divide it by the relevant number from the following table:

3x74 type	Divisor
3274-1A	3600
3274-1D	4000
3274-31A	4300
3274-31D	5000
3274-41A	7200
3274-41D	7200
3174-1L	8400

Figure 19. Table of divisors for 3174 and 3274 controllers

You should now have a number between 0 and 100. This is the percentage 3274 controller use caused by GDDM. If the number is greater than 50% of the spare controller capacity that you have available for GDDM, use the detailed capacity-planning technique in "Capacity planning -- detailed method" on page 76 to analyze your applications in more depth.

Note: The figures in these tables are for use in estimating GDDM performance only. They have not been validated by measurement. IBM cannot vouch for their accuracy, or for the validity of the results. They are given in the belief that some help is better than none.

Capacity planning – detailed method

This section shows you how to estimate the resource utilizations of your GDDM-associated applications. It is a more detailed description of the method shown in Chapter 3, "Storage requirements and capacity planning" on page 61. Like the "first-pass" method, it is not intended to be an overall system performance estimating technique. The aim of the method, and the applications and devices covered, are the same as for the "first-pass method."

Overview of detailed method

The main steps necessary to make the estimate are:

1. Analyze the interactions between terminal and computer in each GDDM application except the ICU:

   a. Divide the interactions into different types, including the terminal type for the graphic ones. For example, your interaction types could be divided into three groups:

      - Simple alphanumerics
      - Simple 3279 graphics
      - Complex 3270-PC/G graphics.

   b. For each class of interaction, estimate the number of interactions per hour.

2. Decide on a typical interaction in each class and use the tables provided to work out its resource usage. Fill in the values in the master table.

3. Decide how much resource each user of the Interactive Chart Utility needs from the tables provided. Fill in the values in the master table.

4. From the values in the master table, calculate the actual resource usage by relating the figures arrived at to real processing units.

Step 1: Identify interactions, classify them, and estimate hourly rate

The estimating technique revolves around identifying the GDDM-associated interactions between terminal user and the computer system. Several interactions can be generated by a single application program or transaction.

1. Examine the GDDM applications that you are interested in and calculate the number of interactions between terminal and computer that they contain.
2. Estimate how many times each interaction will occur per hour.
3. Divide the interactions into the classes shown in the table.
4. From the above, calculate the total number of hourly interactions for each class and put the answer into Figure 20.
5. Interactive Chart Utility interactions can be ignored because the ICU is treated separately.

Table 1 – Interaction types and rates	
Interaction types	Hourly total
1. Very simple procedural alpha, less than 10 fields out and in	
2. Very simple mapped alpha, less than 10 fields out and in	
3. Simple procedural alpha, less than 30 fields out and in	
4. Simple mapped alpha, less than 50 fields out and in	
5. Complex procedural alpha, greater than 50 fields out and in	
6. Complex mapped alpha, greater than 50 fields out and in	
7. Simple graphics a. 3279/3287 b. 3270-PC/G and /GX line, bar, and 2-D pie charts created using Presentation Graphics Routines. Not much shading, not many lines.	
8. Medium/complex graphics a. 3279/3287 b. 3270-PC/G and /GX surface charts, histograms, 3-D pies created using PG Routines; most pictures created by base GDDM calls.	
9. Very complex graphics a. 3279/3287 b. 3270-PC/G and /GX pictures that contain more than 5000 lines, for example cartographics, complex shaded engineering drawings, anything that causes PS overflow on a 3279.	

Figure 20. Table 1. Interaction types and rates

Step 2: Calculate resource cost for "average" member of each group

The next stage is to calculate the resource cost for a typical interaction in each of the groups in Figure 20 using the tables that follow.

1. Decide on your typical interaction in each group. The one that is most commonly used is a sensible method of choosing this.

   It might be that there are several commonly used interactions. In this case it makes sense to identify subgroups within each group and estimate an hourly rate for each. Graphics interactions will almost certainly need this approach. Use Figure 20 on page 77 to note these subgroups.

2. Procedural Alphanumerics

   Use Figure 21 on page 81 to calculate the resource cost of each of your typical procedural alphanumeric interactions. When you have the processor, link, and 3274 controller values write them in the master table, Figure 31 on page 90.

   See the example in Figure 22 on page 82 to learn how to use the tables.

3. Mapped Alphanumerics

   Use Figure 23 on page 83 to calculate the resource cost of your typical mapped alphanumeric interactions. When you have the processor, link, and 3274 controller values write them in the master table, Figure 31 on page 90.

4. Graphics

   To calculate the costs of your graphics interactions you need to know something about the pictures being displayed. The easiest thing to measure or estimate is data stream length. There are two ways of doing this:

   a. Measure representative pictures using the facilities of the GDDM trace, or by using a subsystem trace.

   b. Estimate the data stream based on the following descriptions.

      For business charts, data streams are related approximately to the number of data points plotted and the chart type. The following list of chart types is arranged in data-stream order, least first.

      - Table chart
      - Line graph
      - Surface chart
      - Histogram
      - Bar chart
      - 2-D pie
      - Polar chart
      - 3-D pie
      - Tower chart.

The business chart figures that follow are based on bar charts. If your applications are producing line graphs, estimate up to 30% lower; for tower charts up to 30% higher.

1) Business charts: less than 20 data points plotted:

3279/3287 data stream	= 10 000 bytes uncompressed
	= 4 500 bytes compressed
3270-PC/G or /GX data stream	= 1 500 bytes

2) Business charts: 50 data points plotted:

3279/3287 data stream	= 20 000 bytes uncompressed
	= 9 000 bytes compressed
3270-PC/G or /GX data stream	= 3 000 bytes

3) Business charts: 200 data points plotted:

3279/3287 data stream	= 30 000 bytes uncompressed
	= 13 500 bytes compressed
3270-PC/G or /GX data stream	= 5 000 bytes

4) Simple base GDDM pictures, for example, block diagrams: less than 500 lines in the picture:

3279/3287 data stream	= 6 000 bytes uncompressed
	= 2 750 bytes compressed
3270-PC/G or /GX data stream	= 1 500 bytes

5) Average base GDDM pictures, for example, engineering drawings: 2 000 lines in the picture:

3279/3287 data stream	= 15 000 bytes uncompressed
	= 6 750 bytes compressed
3270-PC/G or /GX data stream	= 5 000 bytes

6) Complex base GDDM pictures, for example, cartographics: 5 000 lines, large amounts of area shading:

3279/3287 data stream	= 30 000 bytes uncompressed
	= 13 500 bytes compressed
3270-PC/G or /GX data stream	= 20 000 bytes

7) Very complex base GDDM pictures, for example, cartographics, seismology, civil engineering: 10 000 lines in the picture:

3279/3287 data stream	= 50 000 bytes uncompressed
	= 22 500 bytes compressed
3270-PC/G or /GX data stream	= 35 000 bytes

For complex pictures, the 3270-PC/G or /GX data stream is approximately three times the number of lines in the picture. The number of lines can be estimated by reference to any or all of:

- Length of the floating point Graphics Data File divided by 10
- Length of the fixed point Graphics Data File divided by 3
- Number of GSLINE calls
- Number of (x,y) points in the arrays used by GSPLNE, GSVECM.

Now you can use the Graphics Estimator Tables (Figure 24 on page 84 for 3279/3287 or Figure 25 on page 85 for 3270-PC/G and /GX) to calculate the costs of a single interaction in this group.

Table 2 — Procedural alphanumerics resource estimation							
		Processor		Link		3274	
	Number	Multi-plier		Multi-plier		Multi-plier	
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
a. Number of fields on screen	. . .	1.82	. . .	xxx	xxx	xxx	xxx
b. Number of modified fields	. . .	1.2	. . .	xxx	xxx	xxx	xxx
c. Average modified field length	. . .	xxx	xxx	xxx	xxx	xxx	xxx
d. Input data-stream length = (b * c)	. . .	xxx	xxx	1	. . .	1	. . .
e. Number of output fields	. . .	xxx	xxx	xxx	xxx	xxx	xxx
f. Average field length	. . .	xxx	xxx	xxx	xxx	xxx	xxx
g. Output data-stream length = (e * f)	. . .	xxx	xxx	1	. . .	1	. . .
h. Average number of interactions/transactions	. . .	xxx	xxx	xxx	xxx	xxx	xxx
i. GDDM initialization cost (1 / h)	. . .	100	. . .	250	. . .	300	. . .
j. Constant	xxx	xxx	6	xxx	xxx	xxx	xxx
GDDM SUBTOTAL	xxx	xxx		xxx		xxx	
k. The GDDM or subsystem transmission buffer size (default 1536)	. . .	xxx	xxx	xxx	xxx	xxx	xxx
l. Number of transmissions = ceiling (g / k) + 1	. . .	20	. . .	20	. . .	20	. . .
m. TSO swapping. If you use TSO, use the value obtained for Item l.	. . .	85	. . .	xxx	xxx	xxx	xxx
n. CICS overheads. If you use CICS, use the GDDM processor subtotal	. . .	0.1	. . .	xxx	xxx	xxx .	xxx
FINAL TOTALS	xxx	xxx		xxx		xxx	

Figure 21. Table 2. Procedural alphanumerics resource estimation

Instructions: Fill in the spaces marked " . . . " (Columns 3, 5, and 7 are calculated from column 1 by using the multiplier in the preceding column. For example, column 5 is calculated by multiplying the value in column 1 by the multiplier in column 4.) Finally, add up the values in columns 3, 5, and 7. An example of a completed table is shown on the next page.

Table 2 - Procedural alphanumerics resource estimation

		Processor		Link		3274	
	Number	Multiplier		Multiplier		Multiplier	
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
a. Number of fields on screen	10.	1.82	18.2	xxx	xxx	xxx	xxx
b. Number of modified fields	5.	1.2	6.	xxx	xxx	xxx	xxx
c. Average modified field length	20.	xxx	xxx	xxx	xxx	xxx	xxx
d. Input data-stream length = (b * c)	100	xxx	xxx	1	100	1	100
e. Number of output fields	10	xxx	xxx	xxx	xxx	xxx	xxx
f. Average field length	25	xxx	xxx	xxx	xxx	xxx	xxx
g. Output data-stream length = (e * f)	250	xxx	xxx	1	250	1	250
h. Average number of interactions/transactions	5	xxx	xxx	xxx	xxx	xxx	xxx
i. GDDM initialization cost (l/h)	1/5	100	20	250	30	300	40
j. Constant	xxx	xxx	6	xxx	xxx	xxx	xxx
GDDM SUBTOTAL	xxx	xxx	50·2	xxx	380	xxx	390
k. The GDDM or subsystem transmission buffer size (default 1536)	1536	xxx	xxx	xxx	xxx	xxx	xxx
l. Number of transmissions = ceiling (g/k) + 1	2	20	40	20	40	20	40
m. TSO swapping. If you use TSO, use the value obtained for Item 1.	2.	85	170	xxx	xxx	xxx	xxx
n. CICS overheads. If you use CICS, use the GDDM processor subtotal.	—	0.1	—	xxx	xxx	xxx	xxx
FINAL TOTALS	xxx	xxx	260·2	xxx	420	xxx	430

Figure 22. Example of filling in a table

Table 3 — Mapped alphanumerics resource estimation							
		Processor		Link		3274	
	Number	Multi-plier		Multi-plier		Multi-plier	
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
a. Number of constant fields	. . .	0.1	. . .	xxx	xxx	xxx	xxx
b. Number of variable fields	. . .	0.8	. . .	xxx	xxx	xxx	xxx
c. Number of modified fields	. . .	0.2	. . .	xxx	xxx	xxx	xxx
d. Average modified field length	. . .	xxx	xxx	xxx	xxx	xxx	xxx
e. Average output field length	. . .	xxx	xxx	xxx	xxx	xxx	xxx
f. Input data-stream length = (c * d)	. . .	xxx	xxx	1	. . .	1	. . .
g. Output data-stream length = ((a + b) * e)	. . .	xxx	xxx	1	. . .	1	. . .
h. Constant	xxx	xxx	8.9	xxx	xxx	xxx	xxx
i. Retrieve MAP from DASD (estimate probability between 0 and 1)	. . .	20	. . .	xxx	xxx	xxx	xxx
j. Average number of interactions/ transaction	. . .	xxx	xxx	xxx	xxx	xxx	xxx
k. GDDM initialization cost = (1 / j)	. . .	100	. . .	250	. . .	300	. . .
GDDM SUBTOTAL	xxx	xxx		xxx		xxx	
l. The GDDM or subsystem transmission buffer size (default 1536)	. . .	xxx	xxx	xxx	xxx	xxx	xxx
m. Number of transmissions = ceiling (g / l) + 1	. . .	20	. . .	20	. . .	20	. . .
n. TSO swapping. Use the value obtained for item (m) if you use TSO	. . .	85	. . .	xxx	xxx	xxx	xxx
o. CICS Overheads. Use the GDDM Processor subtotal if you use CICS	. . .	0.1	. . .	xxx	xxx	xxx	xxx
FINAL TOTALS	xxx	xxx		xxx		xxx	

Figure 23. Table 3. Mapped alphanumerics resource estimation

Instructions: Fill in the spaces marked " . . . " (Columns 3, 5, and 7 are calculated from column 1 by using the multiplier in the preceding column. For example, column 5 is calculated by multiplying the value in column 1 by the multiplier in column 4.) Finally, add up the values in columns 3, 5, and 7.

Table 4 — 3279/3287 graphics resource estimation							
		Processor		Link		3274	
	Number	Multi-plier		Multi-plier		Multi-plier	
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
For pictures that have uncompressed data streams:							
Category 7. Simple pictures							
Data-stream length	. . .	0.11	. . .	1	. . .	1.4	. . .
Constant (delete if none)	xxx	xxx	485	xxx	xxx	xxx	xxx
Category 8. Medium pictures							
Data-stream length	. . .	0.18	. . .	1	. . .	1.4	. . .
Constant (delete if none)	xxx	xxx	425	xxx	xxx	xxx	xxx
Category 9. Complex pictures							
Data-stream length	. . .	2.2	. . .	1	. . .	1.4	. . .
Constant (delete if none)	xxx	xxx	2700	xxx	xxx	xxx	xxx
For pictures that have compressed data streams:							
Category 7. Simple pictures							
Data-stream length	. . .	0.26	. . .	1	. . .	6.7	. . .
Constant (delete if none)	xxx	xxx	485	xxx	xxx	xxx	xxx
Category 8. Medium pictures							
Data-stream length	. . .	0.43	. . .	1	. . .	6.7	. . .
Constant (delete if none)	xxx	xxx	425	xxx	xxx	xxx	xxx
Category 9. Complex pictures							
Data-stream length	. . .	2.4	. . .	1	. . .	6.7	. . .
Constant (delete if none)	xxx	xxx	2700	xxx	xxx	xxx	xxx
a. Average number of interactions/transaction	. . .	xxx	xxx	xxx	xxx	xxx	xxx
b. GDDM initialization cost = (1/a)	. . .	100	. . .	250	. . .	300	. . .
GDDM SUBTOTAL	xxx	xxx		xxx		xxx	
c. The GDDM or subsystem output buffer size (default 1536)	. . .	xxx	xxx	xxx	xxx	xxx	xxx
d. Number of transmissions = ceiling ((column 5subtotal)/c) + 1	. . .	20	. . .	20	. . .	20	. . .
e. TSO swapping. Use the value obtained for Item (d) if you use TSO	. . .	85	. . .	xxx	xxx	xxx	xxx
f. CICS Overheads. Use the GDDM processor subtotal if you use CICS	. . .	0.1	. . .	xxx	xxx	xxx	xxx
FINAL TOTALS	xxx	xxx		xxx		xxx	

Figure 24. Table 4. 3279/3287 graphics resource estimation

Instructions: Fill in the spaces marked " . . . " for one category of picture. (Columns 3, 5, and 7 are calculated from column 1 using the multiplier in the preceding column. For example, column 5 is calculated by multiplying the value in column 1 by the multiplier in column 4.) Finally add up the values in columns 3, 5, and 7. You should only add in the "Constant" part for the category you have chosen.

Table 5 − 3270-PC/G and /GX graphics resource estimation							
		Processor		Link		3274	
	Number	Multi-plier		Multi-plier		Multi-plier	
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Category 7. Simple pictures Data-stream length	. . .	0.1	. . .	1	. . .	1	. . .
Category 8. Medium pictures Data-stream length	. . .	0.09	. . .	1	. . .	1	. . .
Category 9. Complex pictures Data-stream length	. . .	0.08	. . .	1	. . .	1	. . .
a. Average number of interactions/transaction	. . .	xxx	xxx	xxx	xxx	xxx	xxx
b. Number of nondefault fonts used by application	. . .	xxx	xxx	xxx	xxx	xxx	xxx
c. Average font cost = (b / a)	. . .	40	. . .	6000	. . .	6000	. . .
d. GDDM initialization cost = (1 / a)	. . .	100	. . .	500	. . .	500	. . .
GDDM SUBTOTAL	xxx	xxx		xxx		xxx	
e. The GDDM or subsystem output buffer size (default 1536)	. . .	xxx	xxx	xxx	xxx	xxx	xxx
f. Number of transmissions = ceiling ((GDDM column 5 subtotal) / e) + 1	. . .	20	. . .	20	. . .	20	. . .
g. TSO swapping. If you use TSO, use the value obtained for Item f.	. . .	85	. . .	xxx	xxx	xxx	xxx
h. CICS Overheads. Use the GDDM processor subtotal if you use CICS	. . .	0.1	. . .	xxx	xxx	xxx	xxx
FINAL TOTALS	xxx	xxx		xxx		xxx	

Figure 25. Table 5. 3270-PC/G and /GX graphics resource estimation

Instructions: Fill in the spaces marked " . . . " for one category of picture. (Columns 3, 5, and 7 are calculated from column 1 by using the multiplier in the preceding column. For example, column 5 is calculated by multiplying the value in column 1 by the multiplier in column 4.) Finally add up the values in columns 3, 5, and 7.

Step 3: Quantify resources used by the ICU

For ICU estimation categorize your users according to the complexity of what they display and the rate at which they display them.

Five items are involved:

- The type of terminal (3279 or 3270-PC/G or /GX work station) that is being used
- Frequency of display of chart
- Complexity of chart displayed
- Rate at which alphanumeric panels are displayed
- Complexity of the various panels displayed.

This is what you must do to make the estimation:

1. List your ICU users in Figure 26 on page 87.

2. Fill in the type of terminal that each person is using.

3. Fill in their frequency of chart display. 12 per hour is slow, 26 per hour is average, 40 per hour is fast. Decide on rates by observation, tracing, or by informed guesswork.

   What people are doing affects frequency of display. For example, entering lots of data will clearly slow down the display rate; using chart notes usually encourages people to display pictures more frequently. Familiarity with the ICU and keyboards generally will also affect display rates.

4. Fill in complexity of chart:

Simple	Less than 20 data points to be plotted
Average	20 to 50 data points to be plotted
Complex	More than 50 data points to be plotted.

5. Fill in rate at which alphanumeric panels are used:

2 per minute	Slow
3 per minute	Average
4 per minute	Fast.

   Decide on rates by observation, or by informed guesswork. (Use the same category as you chose for graphics if you are guessing.)

6. Fill in complexity of alphanumeric panel use:

Simple	Mainly retrieving pictures and displaying them and making minor modifications to chart layouts. Little use of the "data" panels, (2.1, 2.2, 2.6).
Average	Building charts from scratch; average use of the "data" panels; general usage of most of the alpha panels to set up the chart.
Complex	Building charts from scratch; modifying existing charts; much use of the "data" panels (2.1, 2.2, 2.6).

7. Now use the ICU alphanumerics estimating table and the relevant ICU graphics estimating table to get resource costs for each user.

   Put the numbers obtained from each into the ICU final estimating table.

Table 6 – ICU user categorization table					
		Alphanumeric		Graphics	
User	Terminal	Rate	Complexity	Rate	Complexity
A N Other	*3279*	*Slow*	*Average*	*Slow*	*Complex*

Figure 26. Table 6. ICU user categorization table

Instructions: Fill in the values of the matrix for each user. Sl = Slow, A = Average, F = Fast, Si = Simple, A = Average, C = Complex. The top line is filled in as an example.

Table 7 – ICU alphanumerics estimating table				
Menu Complexity – >		Simple	Average	Complex
Slow Display Rate	Host	14400	28800	43200
	Link	108000	162000	216000
	3274	108000	162000	216000
Average Display Rate	Host	28800	43200	57600
	Link	108000	252000	324000
	3274	180000	252000	324000
Fast Display Rate	Host	43200	57600	72000
	Link	252000	360000	468000
	3274	252000	360000	468000

Figure 27. Table 7. ICU alphanumerics estimating table

Instructions: Find the component values in the matrix for each user and enter them in the ICU final estimating table.

Table 8 – ICU graphics (3279) estimating table							
Menu Complexity – >		Simple		Average		Complex	
Data Stream – >		Uncmp	Compr	Uncmp	Compr	Uncmp	Compr
Slow Display Rate	Host	18000	20000	25000	27500	32000	35500
	Link	79000	35500	122000	55000	173000	78000
	3274	113000	237000	173000	367000	246000	520000
Average Display Rate	Host	43000	47500	57500	63000	72000	79000
	Link	173000	78000	270000	121500	371000	167000
	3274	246000	520000	383000	810000	526500	1113600
Fast Display Rate	Host	72000	79000	86500	95000	108000	119000
	Link	263000	118000	410000	184500	569000	256000
	3274	373000	789000	587000	1231000	813000	1708000

Figure 28. Table 8. ICU graphics (3279) estimating table

Instructions: Establish whether the user is displaying pictures that have compressed or uncompressed data streams. Channel-attached users should be using uncompressed data streams and link-attached users compressed ones. Now find the component values in the matrix for that user and enter them in the ICU final estimating table.

Table 9 – ICU graphics (3270-PC/G and /GX) estimating table				
Menu Complexity – >		Simple	Average	Complex
Slow Display Rate	Host	7200	10000	12800
	Link	15800	24400	34600
	3274	8000	12000	17000
Average Display Rate	Host	17200	23000	28800
	Link	35600	54000	74200
	3274	18000	27000	37000
Fast Display Rate	Host	28800	34600	43200
	Link	52600	82000	113800
	3274	26000	41000	67000

Figure 29. Table 9. ICU graphics (3270-PC/G and /GX) estimating table

Instructions: Find the component values in the matrix for the user and enter them in the ICU final estimating table.

Table 10 – ICU final estimating table			
	Processor (1)	Link (2)	3274 (3)
User 1. ICU Graphics ICU Alphanumerics			
User 2. ICU Graphics ICU Alphanumerics			
User 3. ICU Graphics ICU Alphanumerics			
User 4. ICU Graphics ICU Alphanumerics			
User 5. ICU Graphics ICU Alphanumerics			
User 6. ICU Graphics ICU Alphanumerics			
User 7. ICU Graphics ICU Alphanumerics			
User 8. ICU Graphics ICU Alphanumerics			
User 9. ICU Graphics ICU Alphanumerics			
ICU Subtotal			
TSO swapping overhead = 0.05*ICU Link subtotal (if applicable)		xxx	xxx
CICS overheads = 0.1*ICU Processor subtotal (if applicable)		xxx	xxx
FINAL TOTALS			

Figure 30. ICU final estimating table

Step 4: Find total use made of resources

By now you should have worked through the estimating table for each of the interaction groups that you entered into the master table. You now have total processor, link, and 3274 costs for a single interaction from each of the groups that you identified. Transfer these totals to the master table, Figure 31, if you have not done so already.

Now multiply columns 2, 4, and 6 by the hourly rates in Figure 20 on page 77. Enter the answers in columns 3, 5, and 7. When you have done this for all the interaction groups, add up the values that you have in each of columns 3, 5, and 7. These are the total processor, link, and 3274 costs of the GDDM applications that you are interested in.

The next step is to convert these totals into resource utilizations that apply to your particular computer system.

Master table for resource calculation							
		Processor		Link		3274	
Class of interaction (See Table 1)	Hourly rate (mult)	Value – from table	Hourly value	Value from table	Hourly value	Value from table	Hourly value
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
1	...	...	...	...	...	...	...
2	...	...	...	...	...	...	...
3	...	...	...	...	...	...	...
4	...	...	...	...	...	...	...
5	...	...	...	...	...	...	...
6	...	...	...	...	...	...	...
7	...	...	...	...	...	...	...
8	...	...	...	...	...	...	...
9	...	...	...	...	...	...	...
ICU	xxx	xxx	...	xxx	...	xxx	...
TOTALS	xxx	xxx	...	xxx	...	xxx	...

Figure 31. Master table for resource calculation

Instructions: Fill in the spaces marked " . . . " (Column 1 comes from table 1 and is used as a multiplier. Columns 2, 4, and 6 should be filled in with the results from the other tables. Columns 3, 5, and 7 are filled in from the values in the preceding columns multiplied by the hourly rate from column 1.)

Finally calculate the totals in the bottom row of the table.

Disclaimer

The data in the following tables are for use in determining the approximate use of resources by transactions involving GDDM. They do not necessarily reflect the relative performance of the different controllers or processors for any other transaction type.

Processor utilization

Take the final total in column 3 of the master table and divide it by the relevant processor factor from the following table.

Processor type	Divisor		Processor type	Divisor
4321-1	780		3031-AP	7 300
4331	780		3032	9 900
4331-11	1 300		3083-S	10 700
4331-G2	1 500		3083-E	13 800
4341-9	1 900		3033-EX	14 700
4341-10	2 700		3033-N	15 100
4341	3 100		3033-U	17 900
4341-11	4 000		3033-B	19 900
4341-G2	5 000		3033-BX	21 200
4341-12	5 700		3033-J	25 600
4361-4	3 400		3033-JX	27 300
4361-5	4 300		3033-AP	31 800
4381-1	10 400		3033-MP	31 800
			3081-D	33 100
			3081-G	37 000
			3081-GX	39 900
			3081-K	45 900
			3081-KX	50 500
			3084	87 200
			3084-QX	94 200
			3090-2	101 000
			3090-4	170 000

Figure 32. Table of divisors for different processors

You should now have a number between 0 and 100. This is the percentage processor utilization caused by GDDM.

Link utilization

If all GDDM users are attached over the same communications link then take the final total in column 5 of the master table and divide it by (linkspeed*36) where linkspeed is the speed of your communications link in bytes per second.

You should now have a number between 0 and 100. This is the percentage communications-link utilization caused by GDDM.

If the GDDM users are attached to a variety of communications links and you are interested in individual link utilizations, create a separate master table for each link, identifying the GDDM interactions that occur on it and their hourly rates.

3274 Controller utilization

If all GDDM users are attached to the same 3274 controller, take the final total in column 7 of the master table and divide it by 1000. Now divide it by the relevant number from the following table:

3x74 type	Divisor
3274-1A	360
3274-1D	400
3274-31A	430
3274-31D	500
3274-41A	720
3274-41D	720
3174-1L	840

Figure 33. Table of divisors for 3174 and 3274 controllers

You should now have a number between 0 and 100. This is the percentage 3274 controller utilization caused by GDDM.

If the GDDM users are attached to several controllers and you are interested in individual 3274 utilizations, create a separate master table for each, identifying the GDDM interactions that occur on it and their hourly rates.

Note: The figures in these tables are for use in estimating GDDM performance only. They have not been validated by measurement. IBM cannot vouch for their accuracy, or for the validity of the results. They are given in the belief that some help is better than none.

Chapter 4. Repackaging for performance

Use this chapter to:

- Repackage the GDDM executable code to reduce dynamic loading. (The advantages and disadvantages of doing this are discussed in "Finding the optimum loading and packaging combinations" on page 35.)
- Repackage a GDDM utility or GDDM application program so that dynamic loading is eliminated or reduced. (Again, as discussed in "Finding the optimum loading and packaging combinations" on page 35.)
- Run multiple versions of GDDM with different defaults modules, or override shared defaults modules such as those in a VM saved segment.

Background to repackaging

GDDM executable code modules are, by default, loaded dynamically as needed. They can be repackaged so that some or all of them are to be loaded during program initialization. They can also be repackaged with a GDDM utility or application program so that everything is loaded together, eliminating dynamic loading entirely.

To understand this, and the similar techniques used to run with different defaults modules, you must understand how dynamic loading takes place.

If no packaging is done, the following series of loads takes place when a GDDM application program or GDDM utility is executed.

1. The GDDM utility or application program is loaded.
2. The GDDM initially-loaded modules are loaded. These are:

    a. The application interface controller
    b. The external defaults module
    c. The subsystem initializer.

3. Subsequently, other GDDM modules are loaded as needed by the GDDM subsystem initializer, unless they are on a DCSS.

This process is shown in Figure 34 on page 94.

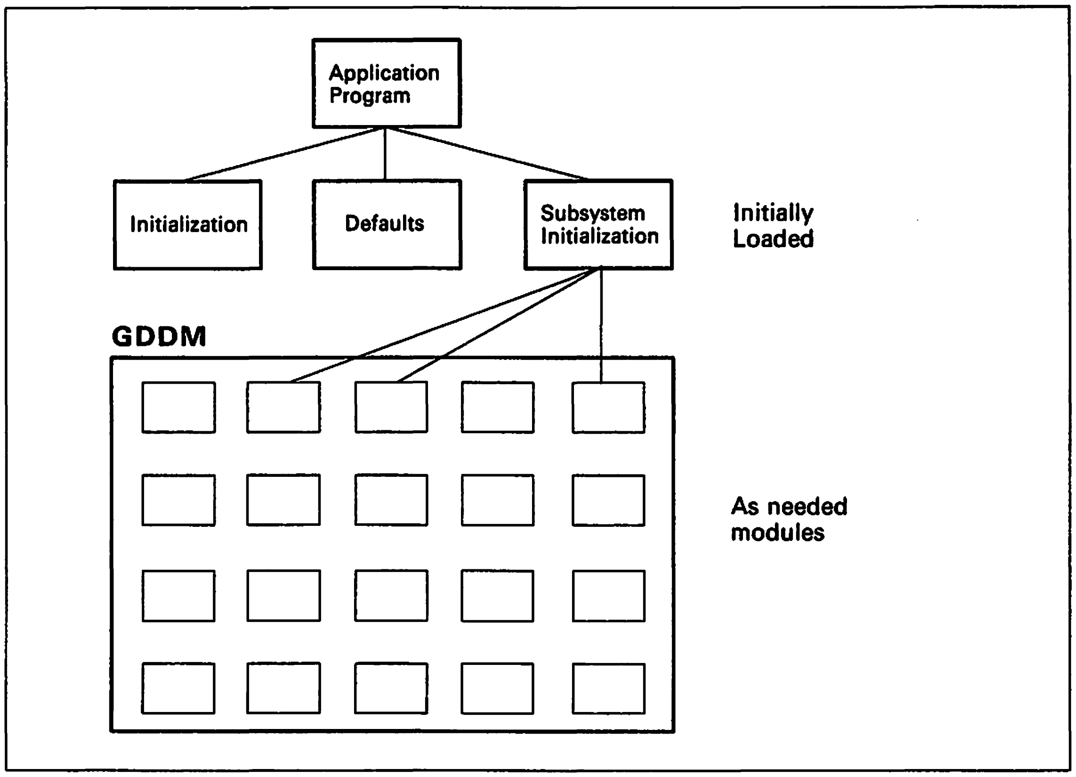


Figure 34. The default order of loading during GDDM utilities and programs

You can reduce the number of loads by packaging the items together. This can be done in several ways, the most important of which are:

1. Repackaging the GDDM executable code so that "as needed" modules are loaded with the subsystem initializer

2. Repackaging a utility or application program with all or part of the GDDM executable code

3. Repackaging a utility or application program with a version of the GDDM external defaults module.

These options are shown in Figure 35 on page 95. Note that any module that is not repackaged will be loaded dynamically as needed. GDDM will still work regardless of what you put in or leave out of the repackaged modules.

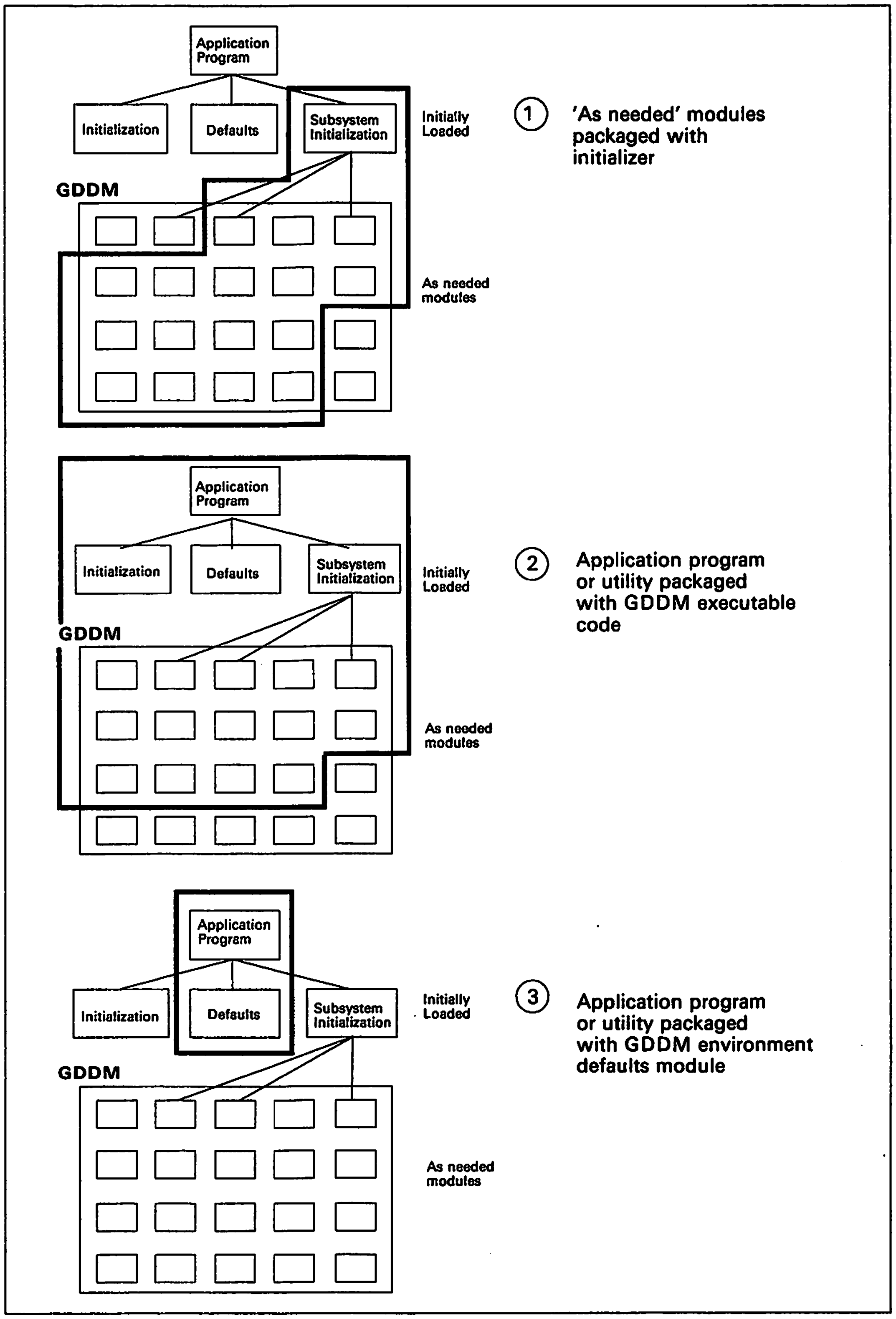

Application Program
Initialization
Defaults
Subsystem Initialization
GDDM
Initially Loaded
As needed modules
① 'As needed' modules packaged with initializer
② Application program or utility packaged with GDDM executable code
③ Application program or utility packaged with GDDM environment defaults module

Figure 35. Possible loading combinations

How to repackage

In this chapter, the term link-editing refers to the use of the LOAD and INCLUDE commands followed by the GENMOD command to create program modules; nonrelocatable files whose external references have been resolved. These files have a file type of MODULE.

You repackage GDDM using the linkage editor (or equivalent loading program). However, the linking of the GDDM modules that are "loaded as needed" has to be done in a special manner.

GDDM supplies modules known as **packaging stubs** that contain references (V-cons) to groups of associated modules. These packaging stubs must be linked with the subsystem initialization module to "drag in" the associated modules. This is shown in Figure 36 on page 97.

Provisos about packaging

Before you consider packaging you should be aware of the following consequences.

Re-link-editing and possible future releases

GDDM releases from Version 1 Release 3 have been designed with the intention that application programs **will not** have to be re-link-edited as a matter of course for any subsequent releases of GDDM.

If packaging options that involve link-editing the application are used, the application will have to be re-link-edited for any subsequent release of GDDM.

Retaining original libraries for service

IBM service procedures (PTF Tapes) assume that the GDDM libraries have not been repackaged. This may cause problems if one of the modules that has been repackaged needs to be changed by service.

Repackaged GDDM routines should therefore be placed in libraries other than the original installation libraries.

The original libraries should be retained and service applied to these libraries. Packaged application programs or repackaged GDDM routines should be regenerated to include the serviced modules.

Special considerations for repackaging modules on MVS/XA

Some of the modules and utilities in GDDM have the attribute RMODE(24). If you include any of these in a repackaged load module, the entire load module will be assigned the attribute RMODE(24), and as a result, you will not be able to place it in a link pack area with a virtual address above 16M bytes.

If you wish to repackage GDDM on MVS/XA, you should note the "Special requirements for MVS/XA" on page 107.

Repackaging the GDDM executable code on its own

GDDM executable code is repackaged using the subsystem initialization module and packaging stubs. The subsystem initialization module is always loaded at the start of a program. Figure 37 shows that it is called ADME000x, where "x" varies according to the subsystem you are using.

To repackage the executable code, link-edit the appropriate ADME000x module with one or more packaging stubs. The packaging stubs "drag in" other routines to form a load module that contains all the modules referenced in the packaging stubs. The composite module is given the same name as the ADME000x module. Note that this can only be done with the ADME000x modules, and that these modules can tell if they have been linked with executable code only if this has been done through packaging stubs.

The effect of repackaging the GDDM executable code is shown in Figure 36.

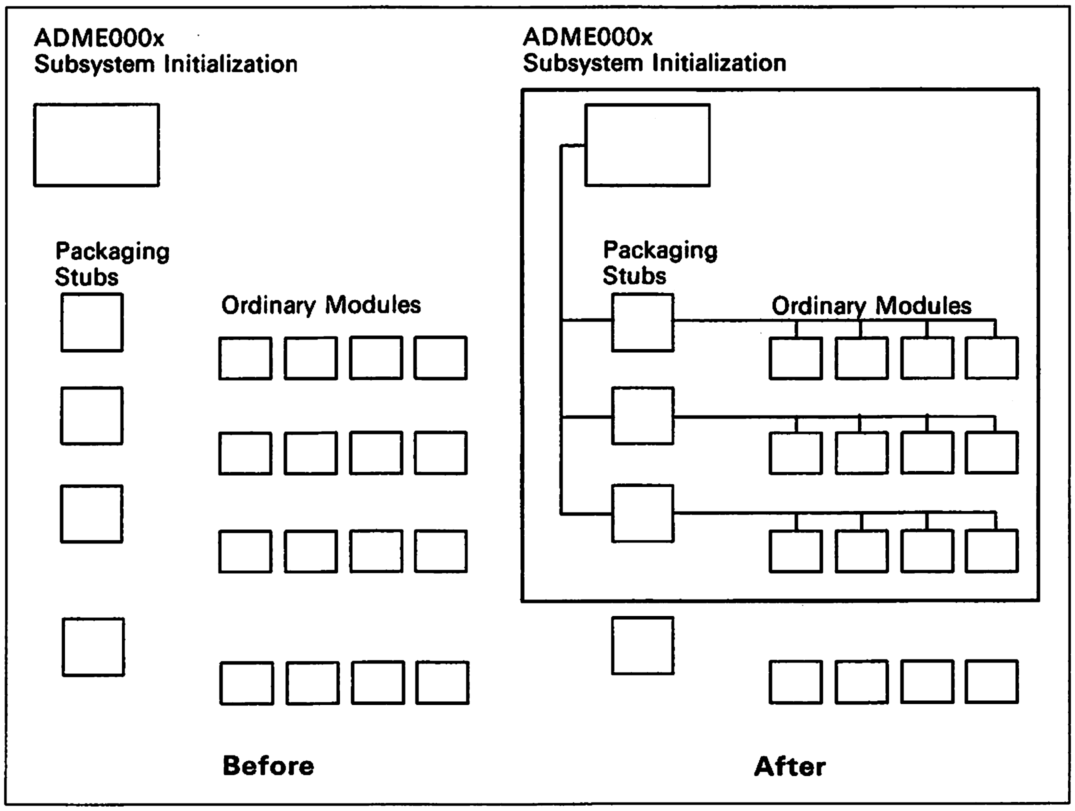


Figure 36. How packaging stubs are used to repackage the executable code

Figure 37 shows the name of the subsystem initialization module for each subsystem.

Names of initially – loaded modules			
Subsystem	Application interface initialization module	External defaults modules	Subsystem initialization modules
CICS	ADMACIN	ADMADFC	ADME000C
IMS/VS	ADMACIN	ADMADFI	ADME000I
TSO	ADMACIN	ADMADFT	ADME000O
TSO Print Utility (VTAM)	ADMACIN	ADMADFT	ADME000O
VM/CMS	ADMACIN	ADMADFV	ADME000V

Figure 37. Names of initially – loaded modules

Levels of packaging stubs

Two levels of packaging stub are provided to give you more control of the modules you repackage. The full screen manager and the ICU can both be subdivided by use of second level packaging stubs. (The full screen manager carries out most of the graphic and alphanumeric processing within GDDM.)

The contents of packaging stubs

Figure 38 shows the contents of the first level packaging stubs. Figure 39 and Figure 40 show the contents of the second level packaging stubs for the full screen manager and the ICU respectively. Note that the use of a first level stub automatically includes all the associated second level stubs.

GDDM first-level packaging stubs		
Name	Contents	Size (in bytes)
	GDDM Base	
	Subsystem-adapter routines	
ADMUX000	Common (used on all subsystems)	62K
ADMUX00x	Subsystem-dependent	
ADMUX00C	CICS	72K
ADMUX00I	IMS/VS	77K
ADMUX00O	OS/VTAM (see note)	90K
ADMUX00T	OS/TSO	89K
ADMUX00V	VM/CMS	84K
	Application-adapter routines	
ADMUXA00	– Common	202K
ADMUXA0x	Messages	
ADMUXA0A	US-English	40K
ADMUXA0B	Brazilian	40K
ADMUXA0D	Danish	40K
ADMUXA0F	French	40K
ADMUXA0G	German	35K
ADMUXA0H	Korean (Hangeul)	40K
ADMUXA0I	Italian	40K
ADMUXA0K	Japanese (Kanji)	38K
ADMUXA0N	Norwegian	34K
ADMUXA0S	Spanish	38K
ADMUXA0V	Swedish	40K
ADMUXD00	Full-Screen Manager routines See later figure second-level split	1165K
ADMUXI00	Image Symbol Editor subroutines	130K
ADMUXO00	Print Utility subroutines	8K
ADMUX300	Image subroutines	261K

Figure 38 (Part 1 of 3). GDDM first-level packaging stubs

GDDM first-level packaging stubs		
	GDDM-PGF	
	Presentation Graphics Routines	
ADMUXB00	Common (processing routines)	196K
ADMUXB0x	Date/day routines	
ADMUXB0A	US-English	0.9K
ADMUXB0B	Brazilian	0.9K
ADMUXB0D	Danish	0.9K
ADMUXB0F	French	0.9K
ADMUXB0G	German	0.9K
ADMUXB0H	Korean (Hangeul)	0.9K
ADMUXB0I	Italian	0.9K
ADMUXB0K	Japanese (Kanji)	0.9K
ADMUXB0N	Norwegian	0.9K
ADMUXB0S	Spanish	0.9K
ADMUXB0V	Swedish	0.9K
	GDDM-PGF and ICU	
ADMUXPAx	Messages	
ADMUXPAA	US-English	19K
ADMUXPAB	Brazilian	19K
ADMUXPAD	Danish	19K
ADMUXPAF	French	19K
ADMUXPAG	German	19K
ADMUXPAH	Korean (Hangeul)	19K
ADMUXPAI	Italian	19K
ADMUXPAK	Japanese (Kanji)	19K
ADMUXPAN	Norwegian	19K
ADMUXPAS	Spanish	19K
ADMUXPAV	Swedish	19K
	ICU Subroutines	
	See later figure second-level split	
ADMUXP00	Common (processing routines)	324K
ADMUXP0x	Menus and HELP	
ADMUXP0A	US-English	405K
ADMUXP0B	Brazilian	405K
ADMUXP0D	Danish	405K
ADMUXP0F	French	406K
ADMUXP0G	German	405K
ADMUXP0H	Korean (Hangeul)	405K
ADMUXP0I	Italian	405K
ADMUXP0K	Japanese (Kanji)	405K
ADMUXP0N	Norwegian	405K
ADMUXP0S	Spanish	400K
ADMUXP0V	Swedish	405K
ADMUXV00	Vector Symbol Editor subroutines	206K

Figure 38 (Part 2 of 3). GDDM first-level packaging stubs

GDDM first-level packaging stubs		
	GDDM-IMD Interactive mapping routines	
ADMUX100	GDDM-IMD subroutines	343K
ADMUX10A	GDDM-IMD messages	19K
Note The OS/VTAM Subsystem-Adapter routines are used by the TSO Print Utility on VTAM. Code sizes quoted are approximate and for guidance only.		

Figure 38 (Part 3 of 3). GDDM first-level packaging stubs

Full Screen Manager second-level packaging stubs		
Name	Contents	Size (in bytes)
Alphanumeric stubs For alphanumeric-only support use all the following		
ADMUXDA0	Alphanumerics	35K
ADMUXDB0	Partitions	6K
ADMUXDC0	Common device processor	34K
ADMUXDE0	Partition sets	7K
ADMUXDP0	Direct printer support	39K
ADMUXDS0	Supervisor	100K
ADMUXDW0	Data stream generator	230K
	Total	451K
Graphic stubs For basic graphics support use all the alpha stubs plus		
ADMUXDG0	Graphics device processor	442K
and for 3279 add		
ADMUXDH0	Graphics generator	41K
Mapping stubs For runtime mapping support use all the alpha stubs plus		
ADMUXDM0	Mapping processor	56K
Image stubs For runtime image support use all the alpha stubs plus		
ADMUXD30	Image processor	66K
Miscellaneous stubs Miscellaneous individual stubs that can be added as needed		
ADMUXDI0	Queued printer support (input)	13K
ADMUXDJ0	Composed-page printers	33K
ADMUXDK0	IPDS printers	21K
ADMUXDL0	System printers	5K
ADMUXDO0	Queued printer support (output)	14K
ADMUXDT0	Plotters	73K
	Total	160K
Code sizes quoted are approximate and for guidance only.		

Figure 39. Full Screen Manager second-level packaging stubs

ICU second-level packaging stubs		
Name	**Contents**	**Size (in Bytes)**
For menu panels		
ADMUXPMx	Menus, messages: choose from list	
ADMUXPMA	US-English	62K
ADMUXPMB	Brazilian	62K
ADMUXPMD	Danish	62K
ADMUXPMF	French	66K
ADMUXPMG	German	62K
ADMUXPMH	Korean (Hangeul)	62K
ADMUXPMI	Italian	62K
ADMUXPMK	Japanese (Kanji)	62K
ADMUXPMN	Norwegian	62K
ADMUXPMS	Spanish	64K
ADMUXPMV	Swedish	62K
For HELP panels		
ADMUXPHx	Help panels: choose from list	
ADMUXPHA	US-English	340K
ADMUXPHB	Brazilian	340K
ADMUXPHD	Danish	340K
ADMUXPHF	French	341K
ADMUXPHG	German	340K
ADMUXPHH	Korean (Hangeul)	340K
ADMUXPHI	Italian	340K
ADMUXPHK	Japanese (Kanji)	340K
ADMUXPHN	Norwegian	340K
ADMUXPHS	Spanish	335K
ADMUXPHV	Swedish	340K
Code sizes quoted are approximate and for guidance only.		

Figure 40. ICU second-level packaging stubs

Which stubs to use

You should include only those packaging stubs that are of interest to you. This means that you will exclude any subsystem adapter routine that you are not interested in (for instance, exclude VM, CICS, and so on, if you use only TSO), and any message routines that are in languages that you do not use (for instance, exclude French, English, and so on if you use only German).

When you have excluded these obvious stubs, you should consider the processing code.

The main processing for all graphics in both GDDM and PG Routines is in fact done by the Full Screen Manager, and major savings can be made by including this in any repackaging.

For the Full Screen Manager and the ICU, second levels of packaging stubs are supplied, to enable their functions to be subdivided.

You may consider excluding the utility subroutines from the package and repackaging them separately. If you are doing this, the Image Symbol and Vector Symbol Editor stubs are candidates for exclusion, as is the print utility stub. Note that for GDDM-IMD, the frame definitions are held as GDDM objects, and cannot therefore be packaged with GDDM-IMD.

Before you exclude the ICU stub, you should remember that the ICU can be called by an application program. Also with the ICU you should remember that the ICU uses the PG Routines, which in turn use the Full Screen Manager.

Rather than using ADMUXD00, which gives the whole of the Full Screen Manager, individual functions can be packaged. Figure 39 on page 102 shows the stubs required to do this. You might include only the first set of these stubs if, for example, you used GDDM only for its alphanumeric support.

Rather than using ADMUXP0x, which gives all the ICU frames, you can separate the HELP panels from the menu panels. Figure 40 on page 103 shows the stubs required to do this. You could use these stubs, for example, to repackage only the working code, and have the HELP panels loaded as needed.

Repackaging executable code with a utility or application program

When you repackage GDDM executable code with an application program or the invoking routine for a GDDM utility, you use the linkage editor (or equivalent) to link the program and the subsystem-initialization module. This, in turn, is linked with the executable code. In practice this means packaging:

- The application program or the invoking routine for a GDDM utility
- The subsystem-initialization module
- Any packaging stubs that are relevant.

Special requirements for utilities

When repackaging the invoking routine for a GDDM utility, the internal link-edit name of the utility invoking routine must be used. These names are shown in Figure 41 on page 105. If utilities are not listed in this figure, they cannot be repackaged.

Internal Link-edit names of invoking routines for GDDM utilities					
Subsystem	Image Symbol Editor	Vector. Symbol Editor	Chart Utility	Print Utility	GDDM-IMD
CICS	ADMISSEC	ADMVSSEC	ADMPSTBC	ADMOPUC	ADM1IMDC
IMS/VS	ADMKSCHI	ADMKSCHI	ADMKSCHI	ADMOPUI	–
OS/VTAM	–	–	–	ADMOPUT & ADMOPST	–
OS/TSO	ADMISSET	ADMVSSET	ADMPSTBT	–	ADM1IMDT
VM/CMS	ADMISSEV	ADMVSSEV	ADMPSTBV	ADMOPUV	ADM1IMDV

Notes:
Any link-editing of the CICS utility invoking routines must also include DFHEAI at offset 0 and DFHEAI0.
(They are the CICS EXEC Interface stubs.)

Any link-editing of the IMS/VS utility invoking routines must also include ASMTDLI, the IMS/VS Application Interface stub.

On MVS/XA, the above routines have these RMODE attributes:
For IMS/VS, OS/VTAM, OS/TSO: RMODE(24)
For CICS/VS: RMODE(ANY)

Figure 41. Internal link-edit names of GDDM utilities

Special requirements for application programs

As described in the *GDDM Base Programming Reference* manual, applications have to be linked with a GDDM interface module (for example, ADMASLC on CICS). This can be done either automatically through using a special form of FSINIT within the program, or explicitly through linkage editor statements.

Exactly the same rules apply when including packaged GDDM executable code.

Eliminating dynamic loading completely

If you want to eliminate dynamic loading completely, you must include in the repackaged module the initially-loaded modules and all the modules that would otherwise be loaded as needed.

All the initially-loaded modules can be included by using packaging stubs. The common adapter stub (ADMUX000) will automatically include the application interface controller. The subsystem adapter stub (ADMUX00x) will automatically include the corresponding external defaults module and subsystem-initialization module.

There is no need to eliminate dynamic loading completely. Any modules that are needed, but are not included in the package, will be loaded dynamically when required.

Repackaging an application or utility with a special defaults module

It is possible to override a shared defaults module by packaging an application program or GDDM utility invoking routine with a special external defaults module. This is done by linking the application or utility invoking routine with the external defaults module. Normal linkage editor (or equivalent) methods are used to link the application or utility invoking routine and the required version of the defaults module.

The same method can be used if you want to use two sets of defaults in an installation; for example, if you wanted to use the ICU in both French and English. Assuming English was specified in the normal defaults module, you would link a copy of the ICU invoking routine with a defaults module that specified French, and make the resulting load module available with a different name. French users would use this version.

The name of the defaults module varies according to the subsystem as shown in Figure 37 on page 98. You will have to assemble a suitable version of the defaults module before doing the packaging.

If you are repackaging with an application program you will have to follow the normal linking methods as described in "Special requirements for application programs" on page 105.

If you are repackaging with a utility invoking routine, you will have to use the internal linkage-editor name for the utility invoking routine as shown in Figure 41 on page 105.

This use of a defaults module is in addition to other methods of specifying user defaults, which are described in the *GDDM Base Programming Reference* manual.

Repackaging for multiple subsystems

If you use GDDM in more than one OS-based subsystem you can repackage the executable code and place it in shareable storage available to all the subsystems.

The packaging stubs required for all of the subsystems that you intend to use should be included in such a composite routine. A linkage-editor NAME and ENTRY statement should be specified for one of the subsystem-initialization routines, and linkage-editor ALIAS statements should be specified for the others.

Application programs and utility invoking routines can be included in the composite module, but only for one subsystem. This limitation arises from the link-editing mechanisms used by GDDM.

Special requirements for various subsystems

Special requirements for CICS

Except on MVS/XA, CICS has a limit of 512K bytes on the size of any one load module. Therefore, any repackaging of GDDM and PGF modules for use in the non-MVS/XA CICS environment should be such that this limit is not exceeded.

Composite modules including utility invoking routines or the subsystem initialization module for use under CICS must include DFHEAI and DFHEAI0. DFHEAI0 must be at offset 0 in the composite load module.

Special requirements for IMS/VS

Under IMS/VS the utility invoking routines must be linked with the ASMTDLI, the IMS/VS application interface stub.

Special requirements for VM/CMS

Under VM/CMS the packaging facilities are used in the construction and operation of a discontiguous saved segment (DCSS). This is more fully described in the *GDDM Installation and System Management for VM*.

If the packaging stub ADMUX00V is loaded with an application program, it will suppress use of the saved segment.

Special requirements for MVS/XA

On subsystems operating under MVS/XA (that is, CICS/VS, IMS/VS, and OS/TSO), some of the modules and utility invoking routines in GDDM have the attribute RMODE(24). (Usually, these are routines containing RMODE-sensitive application or subsystem interfaces.) If you include any of these in a repackaged load module, the entire load module will be assigned the attribute RMODE(24), and, as a result, you will not be able to place it in a link pack area above a virtual address of 16M bytes.

In particular, the subsystem adapter packaging stubs (ADMUX00x) for these subsystems all include at least one module with RMODE(24). This means that if you follow the normal procedures for repackaging described above, you will not be able to locate the resultant load module above 16M bytes.

If you wish to repackage GDDM code on MVS/XA, such that most of the code can be retained above 16M bytes, you should generate two composite routines, as follows:

- Package only the subsystem adapter packaging stub (ADMUX00x) with the corresponding subsystem-initialization module. The resultant composite routine will have RMODE(24), and can be located in shareable storage below 16M bytes if desired.

- Package any other required packaging stubs with the GDDM Application Interface Initialization module (ADMACIN). The common adapter packaging stub (ADMUX000) will automatically include this routine. The resultant composite routine will have RMODE(ANY), and can be located in shareable storage above 16M bytes if desired.

- Application code or the invoking routine for a GDDM utility cannot be packaged with the GDDM Application Interface Initialization module (ADMACIN). Application code or the invoking routine for a GDDM utility can be packaged with the subsystem-initialization module, but any resultant composite routine will acquire RMODE(24).

The "Examples of repackaging" later in this chapter include an example for MVS/XA systems.

Instructions for repackaging

The list below is a suggested order for tackling repackaging:

1. Decide the type of repackaging you are trying to do.

   This is discussed earlier in this chapter and in more broader terms in Chapter 1, "Performance background" on page 1 and Chapter 2, "Tuning and customizing by subsystem" on page 41.

2. If you are using CICS, IMS/VS, VM/CMS, or MVS/XA see "Special requirements for various subsystems" on page 107. If you are repackaging the defaults module, go to item 5.

3. Look at "The contents of packaging stubs" on page 99 and decide which packaging stubs to include.

4. Find the name of your subsystem initialization module in Figure 37 on page 98.

5. If you are changing the defaults module:

   a. Find the name from Figure 37 on page 98.

   b. Change the defaults module as required. Look up "GDDM Defaults and Nicknames" in the *Installation and System Management* manual appropriate to your subsystem for details.

   c. Reassemble the defaults module.

6. If you are linking with a GDDM utility invoking routine, find its internal link-edit name from Figure 41 on page 105.

7. Use the linkage editor or equivalent to create a module containing the required elements and having the correct name.

The examples that follow can be used as models. Further information on link-editing with GDDM can be found in the *GDDM Base Programming Reference* manual.

Examples of repackaging

Repackaging executable code for IMS/VS

The following example shows the statements required to repackage GDDM executable code. The newly packaged load module contains all the GDDM and PGF routines required for execution of PGF Presentation Graphics Routines (excluding the ICU) using American-English.

The packaging shown in this example does not eliminate the dynamic loading of initially-loaded routines, **ADMADFI** and **ADMACIN**.

```
//LINK      EXEC PGM=IEWL,PARM='RENT,REFR,REUS,SIZE=(256K,64K)',
//               REGION=512K
//SYSUT1    DD   UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSLIB    DD   DSN=gddm-executable-load-library,DISP=SHR
//SYSLMOD   DD   DSN=new-packaged-gddm-load-library,DISP=OLD
//SYSPRINT  DD   SYSOUT=A
//SYSLIN    DD   *
 INCLUDE SYSLIB(ADME000I)      IMS/VS subsystem initializer
 INCLUDE SYSLIB(ADMUX000)      Common subsystem adapter
 INCLUDE SYSLIB(ADMUX00I)      IMS/VS subsystem adapter
 INCLUDE SYSLIB(ADMUXA00)      Common Application adapter routines
 INCLUDE SYSLIB(ADMUXA0A)      US-English messages
 INCLUDE SYSLIB(ADMUXD00)      Full screen manager
 INCLUDE SYSLIB(ADMUXB00)      PGF common processing routines
 INCLUDE SYSLIB(ADMUXB0A)      PGF US-English date text routine
 INCLUDE SYSLIB(ADMUXPAA)      PGF US-English messages
 ORDER ADME000I
 ENTRY ADME000I
 NAME  ADME000I(R)
/*
```

Execution of the example will generate a packaged version of the routine ADME000I, and will store this version in a new GDDM load module library.

Repackaging ICU with executable code on TSO

The following example shows the statements required to generate a version of the ICU to be used on TSO. It contains all the required routines for execution using US-English. Such repackaging would speed execution of the ICU because no dynamic loading would be required during its execution.

The ICU is a special GDDM application program using the GDDM System Programmer Interface (SPI). Similar statements are required to package a user application program.

```
//LINK      EXEC PGM=IEWL,PARM='RENT,REFR,REUS,SIZE=(256K,64K)',
//               REGION=512K
//SYSUT1     DD  UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSLIB     DD  DSN=gddm-executable-load-library,DISP=SHR
//SYSLMOD    DD  DSN=newly-packaged-gddm-load-library,DISP=OLD
//SYSPRINT   DD  SYSOUT=A
//SYSLIN     DD  *
 INCLUDE SYSLIB(ADMPSTBT)       ICU invoking routine link-edit name
 INCLUDE SYSLIB(ADMASPT)        GDDM/TSO interface stub
 INCLUDE SYSLIB(ADMUX000)       Common subsystem adapter
 INCLUDE SYSLIB(ADMUX00T)       TSO subsystem adapter
 INCLUDE SYSLIB(ADMUXA00)       Common Application adapter routines
 INCLUDE SYSLIB(ADMUXA0A)       US-English messages
 INCLUDE SYSLIB(ADMUXD00)       Full screen manager
 INCLUDE SYSLIB(ADMUXB00)       PGR common processing routines
 INCLUDE SYSLIB(ADMUXB0A)       PGR US-English date-text routine
 INCLUDE SYSLIB(ADMUXPAA)       PGR US-English messages
 INCLUDE SYSLIB(ADMUXP00)       ICU common processing routines
 INCLUDE SYSLIB(ADMUXP0A)       ICU US-English messages
 ORDER ADMPSTBT
 ENTRY ADMPSTBT
 ALIAS ADMCHART                 Alias name of the ICU
 NAME  ADMPSTBT(R)
/*
```

Execution of the example will generate a packaged version of the routine ADMPSTBT, and will store this version in a new GDDM load module library.

ADMCHART is an alias name for ADMPSTBT, through which the utility may be invoked.

Repackaging GDDM/VSE with the initially loaded modules

The following example shows the statements required to generate a version of a VSE PL/I application program using the reentrant interface to GDDM. The program is packaged with the initially loaded modules.

See *GDDM Base: Programming Reference* for a description of the interface modules ADMASRB and ADMASLC.

```
// OPTION CATAL
   PHASE phase-name, *
   INCLUDE DFHPL1I                      CICS PL/I EXEC interface stub
   INCLUDE PL/I-relocatable-module      The application program
   INCLUDE ADMASRB                      GDDM entry points (reentrant)
   INCLUDE ADMASLC                      GDDM CICS interface
   INCLUDE ADMUX000                     Common subsystem adapter
   INCLUDE ADMUX00C                     CICS subsystem adapter
// EXEC LNKEDT
/&
```

Repackaging executable code for subsystems on MVS/XA

The following example shows the statements required to repackage all GDDM executable code (including GDDM-PGF and GDDM-IMD subroutines) for CICS, IMS/VS, and TSO on MVS/XA. The newly-packaged load modules contain all the required GDDM/MVS, GDDM-PGF, and GDDM-IMD routines for execution using US-English, but do NOT include the GDDM utility invoking routines listed in Figure 41 on page 105. The packaging shown in this example does not eliminate the dynamic loading of initially-loaded routine, ADMADFx.

```
//LINK      EXEC PGM=IEWL,PARM='RENT,REFR,REUS,SIZE=(256K,64K)',
//          REGION=512K
//SYSUT1    DD   UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSLIB    DD   DSN=gddm-executable-load-library,DISP=SHR
//SYSLMOD   DD   DSN=new-packaged-gddm-load-library,DISP=OLD
//SYSPRINT  DD   SYSOUT=A
//SYSLIN    DD   *

  INCLUDE SYSLIB(ADME000C)     CICS/VS subsystem initializer
  INCLUDE SYSLIB(ADMUX00C)     CICS/VS subsystem adapter
  ORDER DFHEAI
  ORDER ADME000C
  ENTRY ADME000C
  NAME  ADME000C(R)

  INCLUDE SYSLIB(ADME000I)     IMS/VS subsystem initializer
  INCLUDE SYSLIB(ADMUX00I)     IMS/VS subsystem adapter
  ORDER ADME000I
  ENTRY ADME000I
  NAME  ADME000I(R)

  INCLUDE SYSLIB(ADME000O)     TSO subsystem initializer
  INCLUDE SYSLIB(ADMUX00O)     VTAM subsystem adapter
  INCLUDE SYSLIB(ADMUX00T)     TSO subsystem adapter
  ORDER ADME000O
  ENTRY ADME000O
  NAME  ADME000O(R)

  INCLUDE SYSLIB(ADMUX000)     Common subsystem adapter
  INCLUDE SYSLIB(ADMUXA00)     Common Application adapter routines
  INCLUDE SYSLIB(ADMUXA0A)     US-English messages
  INCLUDE SYSLIB(ADMUXD00)     Full screen manager
  INCLUDE SYSLIB(ADMUXB00)     PGF common processing routines
  INCLUDE SYSLIB(ADMUXB0A)     PGF US-English date text routine
  INCLUDE SYSLIB(ADMUXPAA)     PGF US-English messages
  INCLUDE SYSLIB(ADMUXP00)     ICU common processing subroutines
  INCLUDE SYSLIB(ADMUXP0A)     ICU US-English panels
  INCLUDE SYSLIB(ADMUXI00)     Image Symbol Editor subroutines
  INCLUDE SYSLIB(ADMUXO00)     Print Utility subroutines
  INCLUDE SYSLIB(ADMUXV00)     Vector Symbol Editor subroutines
  INCLUDE SYSLIB(ADMUX100)     GDDM-IMD subroutines
  INCLUDE SYSLIB(ADMUX10A)     GDDM-IMD messages
  ORDER ADMACIN
  NAME  ADMACIN(R)
/*
```

Execution of the example will generate packaged versions of the routines ADME000C (for CICS), ADME000I (for IMS/VS), ADME000O (for TSO and TSO Print Utility), and ADMACIN (common to all subsystems), and will store these versions in a new GDDM load module library.

Repackaging application program with executable code on VM/CMS

The following example shows the VM/CMS commands entered to create a composite load module for an application program and to execute the composite module. All GDDM routines other than those required for the production of language-dependent error messages are included in the composite module. The language-dependent routines would be dynamically loaded should the application program require the generation of an error message.

The commands shown suppress the use of any copy of GDDM and PGF installed into a discontiguous saved segment (DCSS):

```
GLOBAL TXTLIB ADMPLIB ADMRLIB ADMGLIB PLILIB
LOAD appln-name ADMUX000 ADMUX00V ADMUXA00 ADMUXD00 (START
```

Loaded items are: the application, common subsystem adapter routine (ADMUX000), VM subsystem adapter routine (ADMUX00V), common application adapter routine (ADMUXA00), and the full screen manager (ADMUXD00).

Overriding saved segment defaults module on VM/CMS

A shared segment is frequently used in VM/CMS to reduce the dynamic loading that would otherwise be done for each separate virtual machine. Sometimes it is necessary to override the use of the defaults module for a particular program. A typical example is to use the trace facilities on the ICU, or to have the ICU panels appear in a language other than the default provided for your installation.

The following example shows the VM/CMS commands entered to invoke a GDDM application program on VM/CMS, using a GDDM Environment Defaults Module other than that available in a discontiguous saved segment (DCSS).

The commands shown would permit the use of any other GDDM and PGF routines that were present in a discontiguous saved segment (DCSS).

- Create a suitably modified defaults module and generate it, as described in the *GDDM Installation and System Management* manual for your system.
- Make sure it is present either as a file of type TEXT, or in a TXTLIB of name DFTLIB.
- Issue the commands:

  ```
  GLOBAL TXTLIB DFTLIB ADMRLIB PLILIB
  LOAD appl-name ADMADFV (START
  ```

Loaded items are: the application, and the VM environment defaults module (ADMADFV).

Chapter 5. Application programming for performance

This chapter contains information for application programmers on how to write their GDDM programs to provide improved performance. The topics it covers are:

1. The ADMUFO CSECT, for checking the validity of parameters in GDDM calls
2. Hints for getting the most from your terminal when handling:
   a. Graphics data
   b. Alphanumeric data, through procedural calls and mapping
   c. Image data
   d. Windows
   e. Partitions.
3. How you can help end users
4. Some information specific to programming under CICS.

ADMUFO – the user fast option

Whenever a program makes a GDDM call, all the parameters attached to the call are checked for validity. The cost of doing this can be quite substantial. GDDM allows you to omit this parameter checking and so save processor resource. The way to do this is to define an external control section (CSECT) named ADMUFO to be link-edited with the application program and the GDDM interface module. The contents of the CSECT do not have to be defined.

The ADMUFO CSECT can be defined using standard assembler language facilities, thus:

```
ADMUFO CSECT
       END
```

Alternatively, high-level language constructs can be used, where such are available. In PL/I, the CSECT could be generated by a declaration of the form:

```
DECLARE ADMUFO STATIC EXTERNAL;
```

ADMUFO will give most benefit where the cost of parameter checking is a significant proportion of the total cost of the calls made. Programs that use GDDM procedural alphanumeric calls are likely to show up to 15% reduction in the processor cost attributable to GDDM. The saving for programs that display graphics on the 3279 is more likely to be up to 3%. The saving on 3270-PC/G and 3270-PC/GX work stations is likely to be up to 20%.

The important point to make about ADMUFO is that it should not be included in a program until that program has been found to be free of error. ADMUFO will mask any error caused through having invalid parameters on a call. You could spend much time identifying problems that would otherwise have been diagnosed quickly for you by GDDM's checking of the parameters on each call.

On MVS/XA, the fast bypass will not be invoked if the application addressing mode requires a mode change (that is, if the application call is in 24-bit mode). In this instance, it is necessary to generate a parameter-list copy, with the top bytes of each address word cleared.

This generally means that the User Fast Option functions on MVS/XA only for 31-bit mode applications. An application program executing in AMODE(24) will execute, but with the fast path disabled.

(ADMUFO is described in the *GDDM Base Programming Reference* manual.)

Getting the most from your terminal

As we saw in Chapter 1, "Performance background," GDDM supports a wide range of device classes, some of them possessing varying degrees of processing power. For the more intelligent devices, this section tells you things you can do in your application to take advantage of that power, and so use less host resource. For the less-intelligent devices, this section tells you ways that you can cut your host-performance costs.

Graphics

As a general observation, it is possible to make significant savings in graphics performance, on a device such as the 3279, only by reducing the quality of the picture to be displayed.

1. Reducing the size of the displayed picture will reduce data stream and processor requirements.

2. Pictures that use only a single color have significantly shorter data streams and use slightly less processor resource.

3. Use of mode-1 text rather than mode-2 or -3 also reduces data stream and processor requirements.

Other points to note are:

- The cost of initialization and termination of a GDDM program is significant. One program that displays several pictures is more economic than several programs that each display a single picture.
- There are small savings to be made by using GSVECM or GSPLINE calls to draw multiple lines rather than a series of GSLINE calls.
- There are two methods of saving and displaying pictures:

  Graphics Data Format (GDF) – the picture is held as a series of line, arc, and area definitions that are device-independent. Processing has to be done to convert the picture to a device-dependent form before it can be redisplayed. Little processing work has to be done to redisplay it on 3270-PC/G and /GX work stations; a substantial amount of processing work is required for a 3279.

  FSSAVE/FSSHOW format – the picture is held in **a device-dependent format**.
- On the 3179-G, using non-default vector symbol sets will increase host-processor and data-stream requirements.
- Panning and zooming at a device like a 3279 is performed in the host by GDDM. If the user is doing a lot of panning and zooming, the interaction between the host and the terminal will be intensive, increasing both processor time and the frequency of data-stream transmission.
- On the 5080 graphics system, images use up a lot of storage because each pixel is represented by a byte.

  All mode-2 symbol sets apart from the default are expanded into images and therefore occupy more storage.

  All mode-3 symbol sets (including the default) are expanded into vectors, increasing processing time.
- For 4250, 3800, and 3820 printer picture production, you can trade host cycles against dynamic storage use by setting the **swathe count** in the HRISWATH processing option, and by directing expanded GDF to a spill file in the HRISPILL processing option. See also "Special considerations for composed-page printing" on page 29, and Chapter 2, "Tuning and customizing by subsystem" on page 41.

Processing graphics segments on IBM 3270-PC/G and /GX Work Stations

3270-PC/G and 3270-PC/GX work stations can be used in two modes: **retained** mode and **non-retained** mode. Both modes can provide performance advantages for interactive graphics applications over those provided in previously supported displays.

By default, both 3270-PC/G and /GX work stations are opened as family-1 devices in **retained** mode. This means that retained primitives are stored in the work stations' segment buffers and can be used to refresh the picture after it has been modified. This is an efficient way of using the work station, provided the application program makes good use of graphics segments to break the picture into parts that can be handled easily. Also, retained mode retransmits shorter data streams for graphics areas.

However, storage in these work stations is restricted, and some pictures might be too complicated to store as a whole in the work stations' segment buffers. In such a case, the application program should use the work station in **non-retained** mode. (GDDM switches into non-retained mode, but with a significant cost in host processing.)

An application program can explicitly set non-retained mode if the picture is expected to be too large to fit into segment storage by specifying a DSOPEN call with the value of fullword 2 set to 1 in processing option group 17, SEGSTORE. See the description of DSOPEN in the *GDDM Base Programming Reference* manual for full details.

In non-retained mode, GDDM transmits the picture to the work station as vectors; the work station discards them after they have been processed. GDDM has to retransmit the entire picture (including the graphics area and its boundary) to the work station every time the application program updates it.

For interactive graphics, non-retained mode is, therefore, not as efficient as retained mode, but the performance should be satisfactory for simple output graphics, compared with a 3279 display.

Loading symbol sets: Unlike on the 3279, the GSLSS call loads the symbol set into the work station. This happens **even if the symbol set has already been loaded**. Code containing such redundant GSLSS calls can significantly increase the data-stream size for a 3270-PC/GX, and would actually perform better on a device like a 3279, that cannot have symbol sets downloaded to it.

If you are writing an application that uses several symbol sets, you should therefore ensure either that your program does not contain redundant GSLSS calls, or that it keeps track of which symbol sets are loaded. You can do this by keeping a record within your program of symbol set names as they are loaded, or by using the call GSQSS before the GSLSS, to check with GDDM what has been loaded so far.

Panning and zooming at a work station: The work stations can locally perform the panning and zooming functions of user control. You can take advantage of this using the LCLMODE processing option. It causes the initial data stream that is sent to the device to be increased, but savings occur if the terminal user actually uses panning and zooming frequently.

Redrawing pictures on a 3179-G or 3270-PC/G or /GX: Use of picture segments gives significant performance advantages where the application is making changes to a single picture rather than displaying different pictures. Segments do cause an overhead in the data stream and work station, so don't overdo things.

When you make changes to a picture displayed on a 3179-G, or 3270-PC/G or /GX, the work station may draw just the update or it may need to redraw the entire picture.

You can control whether redraw occurs, by specifying that picture updates are optimized so that only changed segments are updated. The operator can select draft-draw mode, using user control. You specify it by the value that you set either in fullword 2 of processing option group 26 (FASTUPD) of the DSOPEN call, or in the single parameter to the FSUPDM call. The value has the following effect for both calls:

0 Total redraw will occur if necessary (the default). Additions to the picture will generally cause a partial redraw. Deleting segments (because deleting one may make other segments visible) or changing segment attributes (for example, visibility or priority) or transforms requires a total picture redraw. The work station needs to redraw the entire picture to make them appear correctly. It may therefore be best to batch operations that cause a total picture redraw.

For retained mode, total redraw does not mean that the entire picture is retransmitted by GDDM. Only the changes are sent, and the work station incorporates these into the picture definition that it holds in segment storage.

1 Picture to be optimized by a method chosen by GDDM as being the most suitable for the device.

Alphanumerics

There are two different methods of producing alphanumeric displays using GDDM: procedural calls or mapping.

Procedural

This method is based on a set of GDDM calls that:

- Define alphanumeric fields, their length, position, and attributes
- Initialize them with data
- Display them
- Read in any user-modified fields.

Using the procedural technique, the alphanumeric display has to be recreated every time it is needed. A way to avoid this will be discussed later.

The performançe implications of dynamically creating displays are important. Much amount of work has to be done to define the fields, initialize them, and do such things as checking that each field has a unique identifier or does not overlap with any other field. Mapping avoids these costs during the display process.

Mapping

This method of displaying alphanumerics revolves around the one-time definition of an alphanumeric screen or map. This can be displayed later by other programs. The map definition is done interactively using GDDM-IMD, the GDDM-Interactive Map Definition program product. The advantage of this method is that once the map has been defined, the user program has only to retrieve and display it; there is no error checking to be done at run time.

There are advantages and disadvantages for both procedural and mapped alphanumerics. Procedural alphanumerics have the advantage of being completely flexible; displays that are created on a one-time basis are ideal candidates for this technique. Mapping is ideal for programs that display predefined menus. Mapped alphanumerics generally have significant performance advantages over procedural calls, particularly when 20 or more fields are displayed on the screen. Savings of 60 to 80% of the processor cost are likely if mapped displays are used in preference to procedural alphanumeric calls.

The mapping performance advantage does, however, depend on the system overhead to retrieve the map from DASD. This can be a one-time cost for the application, the map being kept in storage for later reuse. This implies an increase in the dynamic storage requirement for the user, but the associated paging almost certainly costs less than retrieving the map from DASD a second time. GDDM automatically maintains recently-used mapgroups in storage in case the application uses them again. The amount of storage that GDDM will use to hold such recently-used mapgroups (and hence the number of mapgroups retained in storage) is controlled by the ADMMDFT MAPGSTG external default. Specification of this option is covered fully under "GDDM Defaults, Exits, and Nicknames" in the *Base Programming Reference* manual.

Procedural alphanumerics performance hints

There are two basic rules to follow for minimizing processor usage when using GDDM procedural alphanumerics:

1. Perform operations on the alphanumeric fields in screen order.

2. The field identifier should be related to the screen position of the field.

   In other words, if you want to define fields, for instance, do it sequentially. Start at the top row, define the fields from left to right, and then do the same for each successive row until you reach the right-hand end of the bottom row. The field identifier should increase as you work down the screen. Experiments have shown that using this technique can reduce processor costs by up to 40% compared with assigning unrelated field identifiers and accessing fields in random order.

Other miscellaneous points to bear in mind when programming with procedural calls:

- ASDFMT is slightly cheaper than ASDFLD when defining fields, although this gain is probably offset by use of ADMUFO.

- Using unnecessary parameters on ASDFMT and ASRFMT calls requires additional processor resource.

- Deep fields are slightly cheaper than having a field on every line of the display. If the data on each line is of varying length, the data stream of the deep field will be longer as padding is inserted on each line.

- Avoid defining long fields in which only the first few bytes contain data. Although the rest of the field will get padded with nulls or blanks, there is a processing cost involved in compressing the padding bytes when the data stream is transmitted to the terminal.

- Use of attributes such as highlighting, color, or reverse video, on a character rather than a field basis, involves extra work in the processor. These should be used sparingly from the human factors point of view also, if they are to have an effect.

Earlier, it was mentioned that it is possible to avoid recreating an alphanumeric screen before redisplaying it within an application. This can be achieved by defining multiple pages using FSPCRT; alphanumeric panels that are going to be displayed several times within the application should each be defined on a separate page. To redisplay a particular panel, reselect that page using FSPSEL and issue the ASREAD to do the display. You must use a unique page for every alphanumeric panel that you want to keep in this way. The following example shows how to do it.

```
         .
         .
CALL FSPCRT(1000,0,0,2);     /* Define a page with an id of 1000 */
CALL FSPSEL(1000);           /* Make it the current page         */
CALL ASDFLD(1,2,5,1,16,2);   /* Define an alphanumeric field     */
                             /*            on the current page */
CALL ASCPUT(1,16,'THIS IS SCREEN 1');  /* Initialize it          */
CALL ASREAD(ATT,ATTM,COUNT);/* Display the current page          */
         .
CALL FSPCRT(1001,0,0,2);     /* Define a page with an id of 1001 */
CALL FSPSEL(1001);           /* Make it the current page         */
CALL ASDFLD(1,2,5,1,16,2);   /* Define an alphanumeric field
                                           on the current page */
CALL ASCPUT(1,16,'THIS IS SCREEN 2');  /* Initialize it          */
CALL ASREAD(ATT,ATTM,COUNT);/* Display the current page          */
         .
CALL FSPSEL(1000)            /* Reselect page 1000               */
CALL ASREAD(ATT,ATTM,COUNT);/* Display it                        */
```

Keeping the pages will increase the amount of dynamic storage that the application uses, but the technique is unlikely to add much to application overheads. Processor costs for the second and subsequent displays of a panel are likely to be 30 to 40% of the original creation and display cost.

Image

Image processing involves the capture, storage, transmission, manipulation, and display of very large amounts of data. Anything that you can do to reduce the sheer volume of data will improve the performance of image applications. This section explains how to do this, and also tells you how to take advantage of the processing power available on image devices.

Image processing on image devices

You can reduce the amount of image data to be processed by excluding any information that is not required. For example, if your application involves the processing of several standard forms that have a certain amount of common information, there is little point in capturing, transmitting, or storing that common information. It makes more sense to keep just the parts of the form that differ. You can do this by defining sub-images to be extracted, using:

- IMREX or IMREXR calls in a transfer operation
- IMATRM call to trim an image, without transferring it

You can also use the IMRSCL call to reduce the size of an image, if appropriate.

If an image is only intended for display at a terminal, or on a low-resolution printer, there is no point in creating, keeping or sending it at a high resolution. The resolution of an image is specified when it is created, using call IMACRT, or changed using IMARES.

The application has control over compression only on IMAPTx calls. Compressing image data shortens image data streams and results in faster transmission between the host and the device. This could be a benefit for remotely-connected devices. However, you should also consider the time taken for the host and the device to compress or decompress the data. The MMR compression algorithm is not suited to images containing photographs or half-tone pictures, where the pixels alternate frequently between the on and off states. In many cases, MMR compression can actually expand the data.

The 3193 can perform transfer operations in the device itself. That is, it can apply a projection to image data. This is called **host offload**, and has the benefit of improved performance for the end-user, and uses less system storage.

The factors affecting host offload are:

- The capabilities of the device
- The level of quality specified as acceptable in your program
- Whether the data is to be used after transmission.

If you are sending image data from your program to a 3193, using IMAPTx calls, any associated projection may be performed in the device, instead of by GDDM in the host, as long as the 3193 can cope with the projection, and depending on the level of quality that you specify as acceptable, using the ISCTL call. ISCTL is fully described in the *GDDM Application Programming Guide.*

Not all transform elements are within the 3193's capabilities. For example, the 3193 is capable of scaling by factors of 0.25, 0.33, 0.5, 0.66, 0.75, 1.0, 1.33, 1.5, 2.0, 3.0, and 4.0. If an application requires an image to be scaled by 2.4 on output, then GDDM has to do it in the host, unless you specify that a certain level of approximation is acceptable. You can do this with the quality parameter in the call ISCTL. If you specify a low value in this parameter, for example, GDDM will allow the device to approximate 2.4 to the device scaling of 2.0. (a value of 2.5 would be approximated to 3.0). If you specify a high value, you are saying that the device's approximation would be unacceptable to you, and the scaling of 2.4 would be performed in the host.

You can use the ISFLD call to control whether an image is to be write-only or read-write. If you do not require the image data to be read subsequently, you should specify "write-only." GDDM can then offload processing to devices, such as the 3193, which support image processing in write-only mode. If you do require the image data to be read subsequently, you should specify read-write. Host offload will then not occur, and GDDM will emulate the processing in the host.

Host offload is a prerequisite for direct transmission. Direct transmission only occurs with IMAPTx transfer operations that have the 3193 as their target.

Projections associated with transfers from image − 1 to image 0 can be carried out in the device. This saves processor utilization and usually means that GDDM doesn't need to keep a copy of the transformed image, so saving virtual storage as well.

Image processing on graphics devices

Image processing is emulated by GDDM on graphics devices such as, for example, the 3179-G. So an image application will use more host and communications resource when running on a graphics device than it will on an image device.

The 4224 printer supports image data without emulation.

Partitions

When you use overlapping partitions in an application, GDDM monitors changes to the partitions and minimizes data stream generation. This improves the front-of-screen appearance, by avoiding unnecessary redrawing of the data, but increases processor usage. As an example, a map in an overlapped partition may use significantly more processor time than a map in a non-overlapped partition.

Using GDDM's partitions to organize your data can improve the performance and the end-user interface of your application. For example, on the 3179-G, when a redraw is required in a partition because a segment has been deleted, only that screen area is refreshed.

How you can help the end user

For a device such as the 3279, the number of PS cell definitions that must be transmitted to the terminal to create a picture depends on the picture's complexity. For example, a multicolored symbol requires three times as many bits for its definition as does the same symbol in monochrome. A complex chart with many lines, shaded areas, colors, and vector text needs much more PS information than a simple one using two or three colors and hardware text characters. Because the more complex picture requires more dynamic storage and takes longer to appear on the screen, the application programmer will generally need to achieve some balance between picture requirements and the operating environment.

Depending on system use, picture complexity, and other factors, several seconds may be required to complete a graphics display on the terminal. In designing interactive application programs that generate pictorial displays, the programmer should attempt to provide some response to the user as soon as possible after the last user action. For example, if the application program must search a large data base or perform extensive calculations before the picture can be constructed, a message might be displayed, indicating that work is in progress. As another example, if data is readily available and the user expects to see the picture, the program should force some information (such as chart axes or a title) onto the screen as soon as possible. The information presented should be something the user would like to see early in the picture generation. However, it should be noted that excessive forcing of partial graphics information can increase the total time needed to send the picture to the terminal, by increasing the number of transmissions.

Performance and the Interactive Chart Utility

Certain calls in the PGF application programming interface let you use parts or all of the ICU as the end-user interface for your own application. You should be aware that there are various techniques for using the ICU efficiently. You could pass these techniques on to users of your application, in the user documentation of your application (for example, help panels):

- PF12 takes you directly to the Home panel. Use it in preference to PF3, which might get you there indirectly.
- It is not necessary to press ENTER to get GDDM to take data on panels like Data Entry. This automatically occurs when you leave the panel.
- When drawing line graphs or surface charts, do not use very high values for the curve-smoothing option until the picture is ready for final display.
- You can go to most panels (for example, 4.4.1) directly from the Home panel, without displaying any of the intermediate routing panels.
- In preview mode, accessible from panel M (Menu Control), you can display a small version of the chart on the panels. The chart is updated if any option is changed. Use of preview mode may reduce resource requirements by reducing the number of full screen picture displays that the user requests. However, if you use preview mode and don't reduce the number of times you request full screen picture displays, you will increase rather than decrease the system resource requirements.
- Extensive use of the ready-made chart function of the **chart-by-example** panels increases system-resource requirements. This is because the ICU first displays examples of the different chart types, then when you have chosen a chart type, displays several examples of variations of that chart type, from which you choose the one best suited to your needs. The displaying of choices increases resource usage. In fact, displaying seven small chart examples on one page uses similar amounts of host-processor and data-stream resource to displaying seven separate pages with a chart on each page. However, the user can use the PF6 key to suppress the display of the small charts.

Application programming under CICS

Under the CICS subsystem, you can specify that the terminal on which a GDDM transaction is running be handled in **pseudoconversational mode**. In this mode, you "divide" the transaction into several GDDM instances. Once output has been sent to the terminal, GDDM frees all its host resources. GDDM is only reinitialized when CICS reinvokes the transaction to handle the next input from the terminal, and the transaction picks up where it left off, using the same mode. The benefit of using this mode is that host resources are not tied up while the end user considers his or her next action.

Pseudoconversational mode is available for the following:

- CICS under MVS or VSE

- Default family-1 display device
- Mapped alphanumeric data
- Output-only graphics data that is not to be updated.

The mode is invoked by a processing option to DSOPEN after the first FSINIT in a GDDM transaction. Each subsequent instance of GDDM in the transaction continues the mode by further FSINITs and DSOPENs. It is the responsibility of the transaction to maintain continuity between the initial instance of GDDM and subsequent instances within the transaction. This involves several calls in each instance – for example, to maintain the output to the screen, and to save, at the end of each instance, information that is required by subsequent instances. The need to issue these calls inevitably increases the processor usage, typically to twice that used by "ordinary" conversational mode. You must therefore make a trade-off between system resource savings through releasing GDDM resources between each instance, and the increase in processor usage because of the greater number of GDDM calls in each instance.

The GDDM functions that can be used in pseudoconversational mode are restricted. See the *Base Programming Reference* manual for full details of use of the mode, and the calls that are restricted.

GDDM glossary

This glossary defines various terms used in the documentation of GDDM.

This glossary includes terms and definitions from the *IBM Vocabulary for Data Processing, Telecommunications, and Office Products*, GC20-1699.

A

AAB. Application anchor block.

absolute data. In GDDM-PGF, the actual y values to be plotted. Contrast with **relative data.**

active partition. The partition containing the cursor. Contrast with **current partition.**

adjunct. In mapped alphanumerics, one of a set of optional subfields in an application data structure that specifies some attribute of a data field, for example, that it is highlighted. An adjunct enables the attribute to be varied at run time.

ADS. Application data structure.

AIC. Application interface component.

AID. Attention identifier.

alphanumeric character attributes. In GDDM, comprise the highlighting, color, and symbol set to be used.

alphanumeric cursor. A physical indicator on a display. It may be moved from one hardware cell to another.

alphanumeric field. A field (area of a screen or printer page) that can contain alphabetic, numeric, or special characters. In GDDM, contrast with **graphics field.**

alphanumeric field attributes. In GDDM, comprise intensity, highlighting, color, symbol set to be used, field type, field end output conversion, input conversion, translate table assignment, transparency, field outlining, and mixed-string fields.

alphanumeric label. In GDDM-PGF or the ICU, a user-specified alphanumeric string used to annotate an x-axis or y-axis scale mark. Contrast with **numeric label.**

alternate device. In GDDM, a device to which copies are sent of the primary device's output. Usually the alternate device is a printer or plotter. See also **primary device.**

annotation. An added descriptive comment or explanatory note.

aperture. See **pick aperture.**

API. Application program interface.

APL. One of the programming languages supported by GDDM.

application data structure (ADS). A structure created by IMD that contains an entry for each variable field within a map. The data to be displayed in a mapped field is placed into the application data structure by the user's program.

application image. In GDDM, an image contained in GDDM main storage, and independent of any device or GDDM page. Contrast with **device image.**

application program interface (API). The formally defined programming-language interface between an IBM system control program or licensed program and its user.

area. In GDDM, a graphics area is a shaded shape, such as a solid rectangle. It is created by opening the area, defining its outline, and closing the area.

aspect ratio. The width-to-height ratio of an area, symbol, or shape.

attention identifier. A number indicating which button the operator pressed to satisfy a read operation. For example, 0 (returned from GDDM to the application program) means that the operator pressed the ENTER key.

attribute byte. The screen position that precedes an alphanumeric field on a 3270-family device and holds the attribute information. See also **trailing attribute byte.**

attribute table. In GDDM-PGF, a set of values for one particular attribute (for example, shading pattern), that are used in sequence to display the components of a business chart.

attributes. Characteristics or properties that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also **alphanumeric character attributes, alphanumeric field attributes,** and **graphics attributes.**

autoranging. In GDDM-PGF, the process in which the axis ranges are determined by the extremes of the data values passed by the application. Synonymous with **autoscaling.**

autoscaling. Synonym for **autoranging.**

axis. In a chart, a line that is drawn to indicate units of measurement against which items in the chart can be viewed. GDDM-PGF charts have an x, y, and (in the case of tower charts) z axis.

axis label. In GDDM-PGF, text appearing at or between axis major scale marks on a business chart. Such labels may be numeric or alphanumeric. Contrast with **data label.**

axis title. In GDDM-PGF, a text string describing what an axis represents.

B

background color. Black on a display, white on a printer. The initial color of the display medium. Contrast with **neutral color.**

bar chart. A chart consisting of several bars of equal width. The value of the dependent variable is indicated by the height of each bar. Synonymous with **column chart.**

BASIC. One of the programming languages supported by GDDM.

BDAM. Basic Direct Access Method.

bi-level image. An image in which each pixel is either black or white (value 0 or 1). Contrast with **gray-scale image** and **halftone image.**

blank character. An empty character represented by X'40' in the EBCDIC code. In GDDM-PGF, such a character occupies one position in a label or a key and may be used for positioning purposes. Contrast with **null character.**

BMS. Basic Mapping Support (CICS/VS).

BPAM. Basic Partitioned Access Method.

business graphics. The methods and techniques for presenting commercial and administrative information in chart form. For example, the creation and display of a sales bar chart. Contrast with **general graphics.**

C

CCW. Channel command word.

CDPF. Composed Document Print Facility.

cell. See **character cell.**

channel-attached. Pertaining to devices that are attached directly to a computer by means of data (I/O) channels. Synonymous with **local.** Contrast with **link-attached.**

character. A letter, digit, or other symbol.

character attributes. See **alphanumeric character attributes.** See also **graphics text attributes.**

character box. In GDDM, the rectangle or (for sheared characters) the parallelogram boundaries that govern the size, orientation, spacing, and italicizing of individual symbols or characters to be shown on a display screen or printer page.

The box width, height, and if required, shear, are specified in world coordinates and may be program-controlled. See also **character mode.** Contrast with **character cell.**

character cell. The physical, rectangular space in which any single character or symbol is displayed on a screen or printer device. The size and position of a character cell are fixed. Size is usually specified in pixels on a given device, for example, 9 by 12 on an IBM 3279 Model 3 display. Position is addressed by row and column coordinates. Synonymous with **hardware cell** and **symbol cell.** Contrast with **character box.**

character code. The means of addressing a symbol in a symbol set, sometimes called **code point.**

The particular form and range of codes depends on the GDDM context, for example:

- For the Image Symbol Editor, a hexadecimal constant in the range X'41'–X'FE', or its EBCDIC character equivalent.
- For the Vector Symbol Editor, a hexadecimal constant in the range X'00'–X'FF' or its EBCDIC character equivalent.
- For the GDDM API, a decimal constant in the range 0 through 239, or subsets of this range (for example, a marker symbol code range of 1 through 8).

character grid. A notional grid that covers the chart area. The size of the grid determines the basic size of the characters in all text constructed by PG routines. It is the fundamental measurement in chart layout, governing the spacing of mode-2 characters and the size of mode-3 characters. It also governs the size of the chart margins and thus the plotting area.

character matrix. Synonym for **dot matrix.**

character mode. In GDDM, the type of characters to be used. There are three modes:

- Mode-1 characters are loadable into PS and are of device-dependent fixed size, spacing, and orientation, as are hardware characters.
- Mode-2 characters are image (ISS) characters. Size and orientation are fixed. Spacing is variable by program.
- Mode-3 characters are vector (VSS) characters. Box size, position, spacing, orientation, and shear of individual characters are variable by program.

chart. In GDDM, usually means business chart, for example, a **bar chart.**

chart annotation. Annotative text added to a business chart. In GDDM-PGF, referred to as **chart notes.**

chart area. In GDDM-PGF, the part of the picture space in which a business chart is to be drawn.

chart attributes. In GDDM-PGF, define how each part of the chart will appear - for example, the font to be used for the chart heading.

chart data. An ICU chart is saved in two distinct parts, the data and the format. The chart data consists of the x and y values, the data labels, the data group names, and the chart heading.

chart data attributes. In GDDM-PGF, define the appearance of the data representation. For example, the color of the lines on a line graph or the shading patterns used for the sectors of a pie chart.

chart format. An ICU chart is saved in two distinct parts, the data and the format. The chart format consists of the chart type, the chart attributes, the axis characteristics, the chart layout, and the chart notes.

chart notes. In GDDM-PGF, additional text to annotate a business chart. May be used in isolation to create alphanumeric presentation material (using the ICU interactive notes facility).

chart type. In GDDM-PGF, specifies whether the business chart should be a line graph, surface chart, histogram, bar chart, pie chart, Venn diagram, polar chart, table chart, or tower chart.

choice device. A logical input device that enables the application program to identify keys pressed by the terminal operator.

CICS/VS. Customer Information Control System/Virtual Storage. A subsystem of MVS or VSE under which GDDM can be used.

clipping. In computer graphics, removing parts of a display image that lie outside a viewport. Synonymous with **scissoring.**

CMS. Conversational Monitor System. A time-sharing subsystem that runs under VM/SP.

COBOL. One of the programming languages supported by GDDM.

code page. Defines the relationship between a set of code points and graphic characters. This relationship covers both the standard alphanumeric characters and the national language variations. GDDM supports a set of code pages used in conjunction with typographic fonts for the IBM 4250 printer.

code point. Synonym for **character code.**

column chart. See **bar chart.**

compass keys. In the GDDM Vector and Image Symbol Editors, a set of PF keys predefined to draw a line in vector symbols or add a dot in image symbols, in directions corresponding to points of the compass.

component (data). In GDDM-PGF, synonym for **data group.** One line on a line graph, for example, or one set of bars on a bar chart.

composed page printer. A printer, such as the IBM 4250 or IBM 3800 Model 3, to which the host computer transmits data in the form of a succession of formatted pages. Such devices will print pictorial data as well as text, and will position all output to pixel accuracy. The density of pixels and the general print quality are often high enough for the output to be used as camera-ready copy for publications.

composite bar chart. In GDDM-PGF, a bar chart in which multiple y values for the same x value or x label are stacked one on top of another. Contrast with **multiple bar chart.** See also **floating bar chart.**

compressed data stream. A data stream that has been made more compact by use of a data-compression algorithm.

constant data. In GDDM, data that is defined in a map and need not be known to the application program.

constant outline. In GDDM-PGF, the ability to specify that all data on the chart (the bars in a bar chart, for example) is outlined in the same color and line type.

correlation. The translation (by GDDM) of a screen position into a part of the user's picture. The action following a pick operation.

current partition. The partition selected for processing by the application program. Contrast with **active partition.**

current position. In GDDM, the end of the previously drawn primitive. Unless a "move" is performed, this position will also be the start of the next primitive.

current vector. In the GDDM Vector Symbol Editor, the vector (displayed in red) that is currently being addressed.

current + 1 vector. In the GDDM Vector Symbol Editor , the vector (displayed in blue) that follows the current vector.

cursor. A physical indicator that may be moved around a display screen. See **alphanumeric cursor** and **graphics cursor.**

curve construction line. In the GDDM Vector Symbol Editor, one of a series of vectors that is used in the construction of a curve.

curve fitting. The construction of a smooth curve through a sequence of plot points, as opposed to their connection by straight lines. In GDDM-PGF or the ICU, curve fitting may be requested for line graphs or polar charts.

D

data group. In GDDM-PGF, one set of y values corresponding to a given set of x values (for example, the data values for one line on a line graph). Synonymous with **component.**

data indexing. In ICU, the display of y values relative to other y values in the same chart, rather than as originally specified. For example, all bars might be displayed as a percentage of the set of bars at X = 1979.

data label. In ICU, text specified on the data entry panel rather than on the x-axis label panel. If the chart has axes, the label is displayed on a tick mark that is close to the matching numeric x value. Data labels are attached to the sectors on a pie chart and to the circles and overlap area of a Venn diagram. Contrast with **axis label.**

data-stream compatibility (DSC). In 8100 systems, the facility that provides access to System/370 applications that communicate with 3270 Information Display System terminals.

data-stream compression. The shortening of an I/O data stream for the purpose of more efficient transmission between link-attached units.

data values. In GDDM-PGF, the x and y values that are plotted on a business chart.

datum line. In GDDM-PGF, a line drawn parallel to a chart axis, through a specified value along the other axis. See also **datum reference line.**

datum reference line. In GDDM-PGF, a datum line that also acts as a shading boundary for the first component of a surface chart, histogram or composite bar chart, or for all the components of a polar chart or multiple bar chart. If no datum reference line is present, such components are shaded from the x axis.

data set. The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access.

DBCS. Double-byte character set.

DCT. Destination control table (CICS/VS).

default value. A value chosen by GDDM when no value is explicitly specified by the user. For example, the default line type is a solid line.

designator character. The first byte of a light-pen-detectable field that indicates whether or not the field has been selected.

device class. In IMD, a classification that groups devices according to the size of their presentation areas (rows and columns).

device echo. A visual identification of the position of the graphics cursor. The form of the device echo is defined by the application program.

device family. In GDDM, a device classification that governs the general way in which I/O will be processed. See also **processing options.** For example:

- Family 1: 3270 display or printer
- Family 2: queued printer
- Family 3: system printer (alphanumerics only)
- Family 4: high-resolution printer

device image. In GDDM, an image contained in a device or GDDM page. Contrast with **application image.**

device suffix. In IMD, a suffix to a mapgroup name that indicates the device class.

device token. In GDDM, an 8-byte code giving entry to a table of pre-established device hardware characteristics that are required when the device is opened (initialized).

digital image. A two-dimensional array of picture elements (pixels) representing a picture. A digital image can be stored and processed by a computer, using bits to represent pixels. In GDDM, pixels have the value black or white. Often called simply **image.**

direct transmission. In GDDM image processing, the transfer of image data direct from a source outside GDDM to an image device, including manipulation by a projection in the device, and without GDDM maintaining a copy or buffer of the data.

display device. Any output unit that gives a visual representation of data. For example, a screen or printer. More commonly, the term is used to mean a screen as opposed to a printer.

display point. Synonym for **pixel.**

display-point matrix. Synonym for **dot matrix.**

display terminal. An input/output unit by which a user communicates with a data-processing system or subsystem. Usually includes a keyboard and always provides a visual presentation of data. For example, an IBM 3179 display.

DL/1. Data language 1. A language for data-base processing operations.

dot matrix. In computer graphics, a two-dimensional pattern of dots used for constructing a display image. This type of matrix can be used to represent characters by dots. Synonymous with **character matrix** and **display-point matrix.**

double-byte character set (DBCS). A set of characters in which each character occupies two byte positions in internal storage and in display buffers. Used for oriental languages.

DPCX. Distributed Processing Control Executive. An 8100 system control program.

DPPX. Distributed Processing Programming Executive. An 8100 system control program.

DSC. Data-stream compatibility.

dual characters. In GDDM, characters that each occupy two bytes in internal storage and in display buffers. They are used to display Kanji or Hangeul symbols.

dummy device. An output destination for which GDDM does all the normal processing but for which no actual output is generated. Used, for example, to test programming for an unavailable output device.

E

echo. In interactive graphics, the visible form of the locator or other logical input device.

ECSA. Extended character set adapter.

edit. To enter, modify, or delete data.

editing grid. In the GDDM Image and Vector Symbol Editors, a grid used as a guide for editing a symbol. In the Image Symbol Editor, it is a dot matrix. In the Vector Symbol Editor, it is a grid of lines.

exploded pie chart. A pie chart in which one or more sectors have been moved outward from the center of the pie, to have a greater impact on the eye.

extended data stream. For 3179, 3278, 3279, and 3287 devices, input/output data formatted and encoded in support of color, programmed symbols, and extended highlighting. These features extend the 3270 data-stream architecture.

extended highlighting. The emphasizing of a displayed character's appearance by blinking, underscore, or reverse video.

external defaults. GDDM-supplied values that users can change to suit their own needs.

extracted image. In GDDM, an image on which transform element calls operate. It may imply the whole source image or just a part of it, depending on whether or not a define sub-image transform element has been applied in its derivation.

F

FCT. File control table (CICS/VS).

field. An area on the screen or the printed or plotted page. See **alphanumeric field, graphics field,** and **mapped field.**

field attributes. See **alphanumeric field attributes.**

fillet. A curve that is tangential to the end points of two adjoining lines.

flat file. A file that contains only data, that is, a file that is not part of a hierarchical data structure. A flat file can contain fixed or variable length records.

floating area. The part of a page reserved for floating maps.

floating bar chart. In GDDM-PGF, a special type of composite bar chart in which the first data group is not displayed. The stacks of bars representing the remaining data groups therefore appear to "float".

floating map. A map whose absolute position on the GDDM page is not fixed. During execution, a floating map takes the next available space that satisfies its specification.

floating-point feature. A processing unit feature that provides four 64-bit floating-point registers to perform floating-point arithmetic calculations.

foil. A transparency for overhead projection.

font. A particular style of typeface (for example, Gothic English). In GDDM, a font may exist as a programmed symbol set.

FORTRAN. One of the programming languages supported by GDDM.

four-button cursor. A hand-held device, with cross-hair sight, for indicating positions on the surface of a tablet. Synonymous with **puck.**

frame. In IMD, synonym for **panel.**

free data. In GDDM-PGF, data that has a separate set of x points for each component. Formerly known as paired data. Contrast with **tied data.**

full-screen alphanumeric operation. Full-screen processing operations on alphanumeric fields.

full-screen mode. A form of screen presentation in which the contents of an entire terminal screen can be displayed at once. Full-screen mode is often used for fill-the-blanks prompting, and is an alternative to line-by-line I/O.

full-screen processor. A host software component that, together with display terminal functions, supports display terminal input/output in full-screen mode.

G

GDDM. Graphical Data Display Manager.

GDDM storage. The portion of host computer main storage used by GDDM.

GDF. Graphics data format.

general graphics. The methods and techniques for converting data to or from graphics display in mathematical, scientific, or engineering applications; that is, any application other than business graphics. See also **business graphics.**

generated mapgroup. The output produced when a source IMD mapgroup is generated. It contains the information needed by GDDM at execution to position the mapped fields on the GDDM page.

graphics. A picture defined in terms of graphics primitives and graphics attributes.

graphics area. Part of a mapped field that is reserved for later insertion of graphics.

graphics attributes. In GDDM, comprise color selection, color mix, line type, line width, graphics text attributes, marker symbol, and shading pattern definition.

graphics cursor. A physical indicator that can be moved (often with a joystick, mouse, or stylus) to any position on the screen.

graphics data format (GDF). A picture definition in an encoded order format used internally by GDDM and, optionally, providing the user with a lower-level programming interface than the GDDM API.

graphics data stream. The data stream that produces graphics on the screen, printer, or plotter.

graphics field. A rectangular area of a screen or printer page, used for graphics. Contrast with **alphanumeric field.**

graphics input queue. A queue associated with the graphics field onto which elements arrive from logical input devices. The program may remove elements from the queue by issuing a graphics read.

graphics primitive. A single item of drawn graphics, such as a line, arc, or graphics text string. See also **graphics segment.**

graphics read. A form of read that solicits graphics input or removes existing elements from the graphics input queue.

graphics segment. A group of graphics primitives (lines, arcs, and text) that have a common window and a common viewport and associated attributes. Graphics segments allow a group of primitives to be subject to various operations. See also **graphics primitive.**

graphics text attributes. In GDDM, comprise symbol (character) set to be used, character box size, character angle, character mode, character shear angle, and character direction.

gray-level. A digitally encoded shade of gray, normally (and always in GDDM) in a range 0 through 255. See also **gray-scale image.**

gray-scale image. An image in which the gradations between black and white are represented by discrete gray-levels. Each pixel of the image therefore has a value in the range 0 through 255. Contrast with **bi-level image** and **halftone image.**

grid coordinates. In the GDDM-PGF Vector Symbol Editor and GDDM Image Symbol Editor, the x and y coordinates of a point on the editing grid.

grid lines. In GDDM-PGF, lines drawn parallel to one axis and through the major scale marks of the other axis.

H

halftone image. A bi-level image in which intermediate shades of grey are simulated by patterns of adjacent black and white pixels. Contrast with **gray-scale image.**

hardware cell. Synonym for **character cell.**

hardware characters. Synonym for **hardware symbols.**

hardware symbols. The characters that are supplied with the device. The term is loosely used also for GDDM mode-1 symbols that are loaded into a PS store for subsequent display. Synonymous with **hardware characters.**

help panel. A panel presenting tutorial text to assist the terminal user. All the GDDM interactive utilities possess comprehensive help panels.

hidden bars. See **overlapping bar chart.**

high-resolution image file. An intermediate form, residing on disk, of a picture destined for a high-resolution printer.

high-resolution printer. A printer, such as the 4250 or 3800-3, that has a high density of pixels to the inch and therefore produces output of good quality.

histogram. A chart in which each value of the dependent variable corresponds to a range of values of the independent variable (represented by the width of the associated bar). Such a chart might display the number of persons in various age ranges, for example.

home panel. The first panel that is displayed by the ICU. It is the starting point for access to the other panels.

I

ICU. Interactive Chart Utility.

identity projection. In GDDM image processing, a null projection, that is, one which results in no change to the image.

image. synonym for **digital image.**

image field. A rectangular area of a screen or printer page, used for image. Contrast with **alphanumeric field** and **graphics field.**

image symbol. A character or symbol defined as a dot pattern.

Image Symbol Editor (ISE). A GDDM-supplied interactive editor that lets users create or modify their own image symbol sets (ISS).

image symbol set (ISS). A set of symbols each of which was created as a pattern of dots. Contrast with **vector symbol set (VSS).**

IMD. Interactive Map Definition.

IMS/VS. Information Management System/Virtual Storage. A subsystem of MVS under which GDDM can be used.

include member. A collection of source statements stored as a library member for later inclusion in a compilation.

indexing. In ICU, see **data indexing.**

input queue. See **graphics input queue.**

integer. A whole number (for example, -2, 3, 457).

Interactive Chart Utility (ICU). A GDDM-PGF menu-driven program that allows business charts to be created interactively by nonprogrammers.

interactive graphics. In GDDM, those graphics that can be moved or manipulated by a user at a terminal.

Interactive Map Definition. A member of the GDDM family of program products. It enables users to create alphanumeric layouts at the terminal. The operator defines the position of each field within the layout and may assign attributes, default data, and associated variable names to each field. The resultant map may be tested from within the utility.

interactive mode. A mode of application operation in which each entry receives a response from a system or program, as in an inquiry system or an airline reservation system. An interactive system may also be conversational, implying a continuous dialog between the user and the system.

interactive subsystem. (1) One or more terminals, printers, and any associated local controllers capable of operation in interactive mode. (2) One or more system programs or program products that enable user applications to operate in interactive mode. For example, CICS/VS.

intercept. In a chart, a method of describing the position of one axis relative to another. For example, the x axis can be specified so that it intercepts (crosses) the y axis at the bottom, middle, or top of the plotting area of a chart.

inter-device copy. The ability to copy a page or the graphics field from the current primary device to another device. The target device is known as the alternate device.

ISE. Image Symbol Editor.

ISS. Image symbol set.

J

JCL. Job Control Language.

joystick. A lever that can pivot in all directions, used as a locator device.

K

Kanji. A character set of symbols used in Japanese ideographic alphabets.

key. In a legend, a symbol and an associated data group name. A key might, for example, indicate that the pink line on a graph represents "Predicted Profit". See also **legend**.

key symbol. A small part of a line (from a line graph) or an area (from a shaded chart) used in a legend to identify the various data groups.

L

legend. A set of symbolic keys used to identify the data groups in a business chart.

line attributes. In GDDM, color, line type, and line width.

line graph. In GDDM-PGF, a chart in which the plotted points (each optionally represented by a marker) are joined by straight or curved lines. If only the markers are displayed, the chart is known as a scatter plot.

link-attached. Pertaining to devices that are connected to a controlling unit by a data link. Synonymous with remote. Contrast with **channel-attached**.

link edit. To create a loadable computer program by means of a linkage editor.

load module. A program unit that is suitable for loading into main storage for execution; it is usually the output of a linkage editor.

local. Synonym for **channel-attached**.

local character set identifier. A hexadecimal value stored with a GDDM symbol set, which may be used by symbol-set-loading means other than GDDM in the context of local copy on a printer.

locator. A logical input device used to indicate a position on the screen. Its physical form may be the alphanumeric cursor or a graphics cursor moved by a joystick.

logarithmic axis. In GDDM-PGF, an axis on which ascending powers of 10 are equally spaced.

logical input device. A concept that allows application programs to be written in a device independent manner. The logical input devices to which the program refers may be subsequently associated with different physical parts of a terminal, depending on which device is used at run-time.

LTERM. In IMS/VS, logical terminal.

M

Manhattan chart. Synonym for **tower chart.**

map. A predefined alphanumeric layout, defining the position, attributes, and default data for each constituent alphanumeric field.

map specification library (MSL). The data set in which maps are held in their source form.

mapgroup. A data item that contains a number of maps and information about the device on which those maps will be used. All maps on a GDDM page must come from the same mapgroup.

mapped alphanumerics. The creation of alphanumeric displays via predefined maps. Contrast with **procedural alphanumerics.**

mapped field. An area of a page whose layout is defined by a map.

mapped graphics. Graphics placed in a graphics area within a mapped field.

mapped page. A GDDM page whose content is defined by maps in a mapgroup.

mapping. The use of a map to produce a panel from an output record, or an input record from a panel.

marker. In GDDM, a symbol centered on a point. Line graphs and polar charts may use markers to indicate the plotted points.

MDT. Modified data tag.

menu. A displayed list of logically grouped functions from which the operator may make a selection.

menu-driven. Describes a program that is driven by an operator responding to one or more displayed menus.

MFS. Message format service.

missing values. In GDDM-PGF or the ICU, x or y values that are omitted from a chart. For example, one line on a graph might represent a sales forecast and extend to the end of the year on the x axis, while a second line might represent actual sales and extend only to the current month.

mixed character string. A string containing a mixture of Latin (one-byte) and Kanji (two-byte) characters.

mixed chart. In GDDM-PGF or the ICU, the combination of more than one chart type in a business graph. For example, the overlaying of a line graph on top of a bar chart.

mode 1/2/3 characters. See **character mode.**

mountain shading. A method of shading surface charts where each component is shaded separately from the base line, instead of being shaded from the data line of the previous component.

mouse. A hand-held device (the IBM 5277 Mouse) that is moved around a locator pad to position the graphics cursor on the screen.

MSHP. Maintain system history program.

MSL. Map specification library.

multicomponent chart. In GDDM-PGF, a chart presenting more than one data group.

multiple axis chart. In GDDM-PGF, a chart in which more than one x axis or y axis, or both, is used. See also **secondary axis.**

multiple bar chart. In GDDM-PGF, a form of bar chart in which the bars at a given x value or label are placed side by side. Contrast with **composite bar chart.**

multiple charts. Two or more charts appearing together on the display screen or page. Multiple charts can be of the same type or different types, and can be derived from one or more sets of data.

N

National Language (NL) feature. The translations of the ICU panels and GDDM messages into a variety of languages other than English.

negate. In bi-level image data, setting zero bits to one and one bits to zero.

neutral color. White on a display, black on a printer. Contrast with **background color.**

nickname. In GDDM, a quick and easy means of referring to a device, the characteristics and identity of which have been predefined.

non-paired data.. See **tied data.**

null character. An empty character represented by X'00' in the EBCDIC code. In GDDM-PGF, such a character does not occupy a screen position. The trailing positions of short keys or labels may be filled with nulls. Contrast with **blank character**.

numeric label. In GDDM-PGF, an axis major scale mark label derived directly from the data value at that scale mark. Contrast with **alphanumeric label**.

O

object code. Output from a compiler or assembler that is in itself executable machine code or is suitable for processing to produce executable machine code.

object deck. Synonym for **object module**.

object libraries. An area on a direct access storage device used to store object programs and routines.

object module. A module that is the output of an assembler or a compiler and is input to a linkage editor.

off-point. A pixel that has been turned off by the user of the Image Symbol Editor.

on-point. A pixel that has been turned on by the user of the Image Symbol Editor.

operator reply mode. In GDDM, the mode of interaction available to the operator (display terminal user) with respect to the modification (or not) of alphanumeric character attributes for an input field.

outbound structured field. An element in 3270 data streams from host to terminal with formatting that permits variable-length and multiple-field data to be sequentially translated by the receiver into its component fields without having to examine every byte.

overlapping bar chart. A form of business chart where adjacent bars partly overlap each other. Overlapping bars are sometimes called **hidden bars**.

P

page. In GDDM, the main unit of output and input. All specified alphanumerics and graphics are added to the current page. An output statement always sends the current page to the device, and an input statement always receives the current page from the device.

pageable (main storage). In a virtual storage system, fixed-length blocks that have virtual addresses and that can be transferred between real (main) storage and auxiliary storage.

paired data. See **free data**.

panel. A predefined display that defines the locations and characteristics of alphanumeric fields on a display terminal. When the panel offers the operator a selection of alternatives it may be called a menu. Synonymous with **frame**.

partition. Part of the display screen's surface on which a page of GDDM output can be shown. Two or more partitions can be created, each displaying a page of output.

partition set. A grouping of partitions that are intended for simultaneous display on a screen.

partitioned data set (PDS). A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data. Synonymous with **program library**.

PCB. Program communication block (IMS/VS).

PCT. Program control table (CICS/VS).

PDS. In OS/TSO, a partitioned data set.

pel. Synonym for **pixel**.

PGF. Presentation Graphics Facility.

pick. The action of the operator selecting part of a graphics display by placing the graphics cursor over it.

pick aperture. A rectangular or square box that is moved across the screen by the graphics cursor. An item must lie at least partially within the pick aperture before it can be picked.

pick device. A logical input device that allows the application to determine which part of the picture was selected (or picked) by the operator.

picture element. Synonym for **pixel**.

picture interchange format (PIF) file. In graphics systems, the type of file, containing picture data, that can be transferred between GDDM and a 3270-PC/G or 3270-PC/GX work station.

picture space. In GDDM, an area of specified aspect ratio that lies within the graphics field. It is centered on the graphics field and defines the part of the graphics field in which graphics will be drawn.

pie chart. A chart that takes the form of one or more circles divided into sectors, the angles of which represent the contributions of each data value to the group total.

PIF. Picture interchange format (PIF) file.

pixel. The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonymous with **display point, pel,** and **picture element.**

PL/I. One of the programming languages supported by GDDM.

plotter. An output device that uses pens to draw its output on paper or transparency foils.

polar chart. A form of business chart where the x axis is circular and the y axis is radial.

polyfillet. In GDDM, a curve based on a sequence of lines. It is tangential to the end points of the first and last lines; and tangential also to the midpoints of all other lines.

polyline. A sequence of adjoining lines.

PPT. Processing program control table (CICS/VS).

presentation graphics. Computer graphics products or systems, the functions of which are primarily concerned with graphics output presentation. For example, the display of business planning bar charts.

Presentation Graphics Facility (PGF). A member of the GDDM family of program products. It is concerned with business graphics, as opposed to general graphics.

preview chart. A small version of the current chart that may be displayed on ICU panels.

primary device. In GDDM, the main destination device for the application program's output, usually a display terminal. The default primary device is the user console. See also **alternate device.**

primitive. See **graphics primitive.**

primitive attribute. A specifiable characteristic of a graphics primitive. See **graphics attributes** and **graphics text attributes.**

print utility. A subsystem-dependent utility that sends print files from various origins to a queued printer.

procedural alphanumerics. The creation of alphanumeric displays using the GDDM alphanumeric API. Contrast with **mapped alphanumerics.**

processing options. Describe how a device's I/O will be processed. These device-family-dependent and subsystem-dependent options are specified when the device is opened. An example is the choice between CMS attention-handling protocols.

program library. (1) A collection of available computer programs and routines. (2) An organized collection of computer programs. (3) Synonym for **partitioned data set.**

programmed symbols (PS). Dot patterns loaded by GDDM into the PS stores of an output device.

projection. In GDDM image processing, an application-defined function which specifies operations to be performed on data extracted from a source image. Consists of one or more **transforms,** which see. See also **transform element.**

PS. Programmed symbols.

PS overflow. A condition where the graphics cannot be displayed in its entirety because the picture is too complex to be contained in the device's PS stores.

PSB. In IMS/VS, a program specification block.

puck. Synonym for **four-button cursor.**

Q

QSAM. Queued sequential access method.

QTAM. Queued telecommunications access method.

queued printer. A printer belonging to the subsystem under which GDDM runs, to which output is sent indirectly by means of the GDDM Print Utility program. In some subsystems, this may allow the printer to be shared between multiple users. Contrast with **system printer.**

R

RAS. Reliability, availability, serviceability.

raster device. A device with a display area consisting of dots. Contrast with **vector device.**

rastering. The transforming of graphics primitives into a dot pattern for line-by-line sequential use. In GDDM PS devices, this is done by transforming the primitives into a series of programmed symbols (PS).

RCP. Request control parameter.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference line. See **datum reference line.**

reference symbol. In the GDDM Image and Vector Symbol Editors, a previously defined symbol displayed with the symbol currently being edited, for the purpose of achieving consistent symbol sizes and shapes within a given symbol set.

regression line. In ICU, the conversion of a set of y values into other values that form a straight line most closely resembling the original values.

relative data. In GDDM-PGF, real y-data values that are to be presented in a stacked chart-type. The actual points to be plotted for a particular component are obtained by adding the y data of that component to the y data of the previous components. Contrast with **absolute data.**

remote. Synonym for **link-attached.**

reply mode. See **operator reply mode.**

resolution. In graphics and image processing, the number of pixels per unit of measure (inch or meter).

reverse clipping. Where one graphics primitive overlaps another, removing any parts of the underlying primitive that are overpainted by the overlying primitive.

reverse video. A form of alphanumeric highlighting for a character, field, or cursor, in which its color is exchanged with that of its background. For example, changing a red character on a black background to a black character on a red background.

S

scalable markers. In GDDM-PGF, (vector) markers on a line graph or polar chart that may be varied in size.

scale marks. In GDDM-PGF, markings spaced at equal intervals along an axis. Each pair of "major scale marks" may have one or more "minor scale marks" in between. Synonymous with **tick marks.**

scanner. A device that produces a digital image from a document.

scatter plot. In GDDM-PGF, a variety of line graph in which only the marked points, and not their joining lines, are drawn.

scissoring. Synonym for **clipping.**

scrolling. In computer graphics, moving a display image vertically or horizontally in a manner such that new data appears at one edge as existing data disappears at the opposite edge.

SCS. SNA character string.

secondary axis. In GDDM-PGF, an x- or y-axis line drawn parallel to the primary axis and capable of having scale marks, labels and title different from those of the primary axis. Permits the combination of two business charts.

segment. See **graphics segment.**

segment attributes. Attributes that apply to the segment as an entity, rather than to the individual primitives within the segment. For example, the visibility, transformability, or detectability of a segment.

segment library. The portion of auxiliary storage where segment definitions are held. These definitions are GDDM objects in graphics data format (GDF) and are managed by means of GDDM API calls. GDDM handles the file accesses to and from auxiliary storage.

segment priority. The order in which segments will be drawn, also the order in which they will be detected.

segment transform. The means to rotate, scale, and reposition segments without re-creating them.

selector adjunct. A subfield of an application data structure that qualifies a data field.

shear. The action of tilting graphics text so that each character leans to the left or right while retaining a horizontal baseline.

skyscraper chart. Synonym for **tower chart.**

SMF. System management facilities.

SMP. System management program.

SNA. Systems network architecture

source image. An image that is the data input to image processing or transfer.

SPI. System programmer interface.

SPIB. System programmer interface block.

spider labels. In GDDM-PGF, labels that annotate pie-chart sectors. Each label is joined to its associated sector by a line, giving the resulting chart a spider-like appearance.

stacked chart type. A surface chart, composite bar chart, or histogram where the data components are stacked one on top of another. The data value of a particular component is indicated by the depth of the band at that point. See also **relative data.**

stand-alone (mode). Operation that is independent of another device, program, or system.

state-1. In GDDM-PGF, the state of a business graphics program, before the first plot has been made.

state-2. In GDDM-PGF, the state of a business graphics program after the first plot has been made, thereby constructing the axes.

string device. A logical input device that enables an application program to process character data entered by the terminal operator.

stroke device. A logical input device that enables an application program to process a sequence of x,y coordinate data entered by the terminal operator.

stylus. A pen-like pointer for indicating positions on the surface of a tablet.

surface chart. A chart similar to a line graph, except that no markers appear and the areas between successive lines are shaded.

swathe. A horizontal slice of printer output, forming part of a complete picture. High-resolution printer images are often constructed in swathes to reduce the amount of storage required.

symbol. Synonymous with **character.** For example, the following terms all have the same meaning: vector symbols, vector characters, vector text.

symbol cell. Synonym for **character cell.**

symbol matrix. Synonym for **dot matrix.**

symbol set. A collection of symbols, usually but not necessarily forming a font. GDDM applications may use the hardware device's own symbol set. Alternatively, they can use image or vector symbol sets, which the user may have created.

symbol set identifier. In GDDM, an integer (or the equivalent EBCDIC character) by which the programmer refers to a loaded symbol set.

system printer. A printer belonging to the subsystem under which GDDM runs, to which output is sent indirectly by means of system spooling facilities. Contrast with **queued printer.**

T

table chart. In GDDM-PGF, a chart in which the data is presented as numbers arranged in rows and columns.

tablet. (1) A locator device with a flat surface and a mechanism that converts indicated positions on the surface into coordinate data. (2) The IBM 5083 Tablet Model 2, which, with a four-button cursor or stylus, allows positions on the screen to be addressed and the graphics cursor to be moved without use of the keyboard.

tag. In interactive graphics, an identifier associated with one or more primitives that is returned to the program if such primitives are subsequently picked.

target image. An image which is the destination of processed or transferred data.

target position. In the GDDM Vector Symbol Editor, the grid coordinates of a point on the editing grid to which a vector is to be drawn.

TCT. Terminal control table (CICS/VS).

temporary graphics. Graphics created outside a segment.

terminal. A device, usually equipped with a keyboard and a display unit, capable of sending and receiving information over a link. See also **display terminal.**

test symbol. In the GDDM Image and Vector Symbol Editors, an area on the Symbol Edit panel in which the currently chosen symbol is displayed.

text. Characters or symbols sent to the device. GDDM provides alphanumeric text and graphics text.

text attributes. See **graphics text attributes.**

tick marks. In GDDM-PGF, synonym for **scale marks.**

tied data. In GDDM-PGF, data that shares the same set of x points for each component. This is the most common form of data. It was formerly known as **non-paired data.** Contrast with **free data.**

tilted pie chart. A pie chart drawn in three dimensions, which has been tilted away from full face to reveal its three-dimensional properties.

tower chart. A form of business chart in which rows of towers stand on a two-dimensional base. Synonymous with **Manhattan chart** and **skyscraper chart.**

trailing attribute byte. The screen position following an alphanumeric field. This attribute byte may specify, for example, that the cursor should auto-skip to the next field when the current field is filled.

transfer operation. In GDDM image processing, an operation in which a projection is applied to a source image, and the result placed in a target image. The source and target images can be device or application images in any combination, or one or other of them (but not both) can be image data within the application program.

transform. (1) The action of modifying a picture for display; for example, by scaling, rotating, shearing, or displacing. (2) The object that performs or defines such a modification; also referred to as a **transformation**. (3) In GDDM image processing, a definition of three aspects of the data manipulation to be done by a projection:

1. a transform element or sequence of transform elements, and
2. a resolution conversion or scaling algorithm, and
3. a location within the target image for the result.

Only the last item is mandatory.

See also **projection** and **transform element**.

transform element. In GDDM image processing, a specific function in a transform, which may be one of the following: define sub-image, scale, orient, reflect, negate, define place in target image.

A given transform element may be used only once in a **transform**, which see.

transformable. A segment must be defined as transformable if it will subsequently be moved, scaled, or rotated.

transparency. (1) A document on transparent material suitable for overhead projection. (2) An alphanumeric attribute that allows underlying graphics to show.

TSO. Time sharing option. A subsystem of OS/VS under which GDDM can be used.

TWA. Transaction work area.

U

UDS. User default specification.

UDSL. A list of user default specifications (UDSs).

unformatted data. In GDDM image processing, compressed or uncompressed binary image data that has no headers, trailers, or embedded control fields other than any defined by the compression algorithm, if applicable. The data is in row major order, beginning with the top left of the picture.

user control. A GDDM function that allows the terminal operator to perform some actions without the need for application programming. The actions include panning and zooming graphics, manipulating windows, and printing, plotting, and saving pictures.

user default specification (UDS). The means of changing a GDDM default value. The default values that a UDS can change are those of the GDDM or subsystem environment, GDDM user exits, and device definitions.

user exit. A point in GDDM execution where a user routine will gain control if such has been requested.

V

variable cell size. In most devices, the hardware cell size is fixed. But the 3290 Information Panel has a cell size that can be varied. This in turn causes the number of rows or columns on the device to alter.

VCNA. VTAM communications network application.

vector. (1) In computer graphics, a directed line segment. (2) In the GDDM-PGF Vector Symbol Editor, a straight line between two points.

vector device. A device capable of displaying lines and curves directly. Contrast with **raster device**.

vector symbol. A character or shape made up of a series of lines or curves.

Vector Symbol Editor. A program supplied with GDDM-PGF, the function of which is to create and edit vector symbol sets (VSS).

vector symbol set (VSS). A set of symbols each of which was originally created as a series of lines and curves.

Venn diagram. A form of business chart in which two populations and their intersection are represented by two overlapping circles.

viewport. A subdivision of the picture space, most often used when two separate pictures are to be displayed together.

VM/SP CMS. IBM Virtual Machine/System Product Conversational Monitor System. A system under which GDDM can be used.

VSE. Virtual storage extended. An operating system consisting of VSE/Advanced Functions and other IBM programs. In GDDM, the abbreviation VSE has sometimes been used to refer to the Vector Symbol Editor, but to avoid confusion, this usage is deprecated.

VSS. Vector symbol set.

VTAM. Virtual Telecommunications Access Method

W

window. (1) In GDDM, a defined section of world coordinates. The window can be regarded as a set of coordinates that are overlaid on the viewport. (2) In GDDM, the "graphics window" is the set of coordinates used for defining the primitives that make up a graphics display. By default, both x and y coordinates run from 0 through 100. (3) In GDDM, an "operator window" is an independent rectangular subdivision of the screen. Several can exist at the same time, and each can receive output from, and send input to, either a separate GDDM program or a separate function of a single GDDM program. (4) In GDDM, the "page window" defines which part of a deep or wide page should currently be displayed.

work station. A display screen together with attachments such as a local copy device or a tablet.

world coordinates. The user application-oriented coordinates used for drawing graphics. See also **window.**

wrap-around field. An alphanumeric field that extends to the right-hand edge of the page and continues at the start of the next row.

WTP. Write-to-programmer.

Index

GDDM Performance Guide

READER'S COMMENT FORM

Order No. SC33-0324-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Number of your latest Technical Newsletter for this publication . . .

If you want an acknowledgement, give your name and address below.

Name . . . .

Job Title . . . . Company . . . .

Address. . . . .

. . . . Zip . . . .

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: *Staples can cause problems with automated mail-sorting equipment.*
Please use pressure-sensitive or other gummed tape to seal this form.

SC33-0324-0

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 6R1H,
180 Kost Road,
Mechanicsburg, PA 17055, USA

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

SC33-0324-0
Version 2 Release 1

GDDM Performance Guide Printed in USA SC33-0324-0

SC33-0324-00