NISTIR 5146

Detailed Design Specification for Conformance Testing of Computer Graphics Metafile (CGM) Interpreter Products

Daniel R. Benigni, Editor

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Computer Systems Laboratory
Information Systems Engineering Division
Gaithersburg, MD 20899

NIST

NISTIR 5146

Detailed Design Specification for Conformance Testing of Computer Graphics Metafile (CGM) Interpreter Products

Daniel R. Benigni, Editor

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Computer Systems Laboratory
Information Systems Engineering Division
Gaithersburg, MD 20899

March 1993

U.S. DEPARTMENT OF COMMERCE
Ronald H. Brown, Secretary

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Raymond G. Kammer, Acting Director

Detailed Design Specification for Conformance Testing of Computer Graphics Metafile (CGM) Interpreter Products

TABLE OF CONTENTS

EXECUTIVE SUMMARY

PURPOSE

This report was prepared by the National Institute of Standards and Technology/Computer Systems Laboratory (NIST/CSL) in support of the Computer-aided Acquisition and Logistics Support (CALS) initiative. It represents in particular a NIST/CSL FY92 contract deliverable task to develop a detailed design specification for determining conformance of CGM (Computer Graphics Metafile) Interpreter Products to the requirements of FIPS 128, the CGM standard, and the Military Specification MIL-D-28003A.

BACKGROUND

From CALS early beginnings in 1986, it was known that the CGM standard offered no conformance statements concerning either generators (writers) or interpreters (readers) of metafiles. An international workshop on CGM certification convened in the UK in 1987 concluded that 'a CGM Testing Architecture must include testing for CGM generators and interpreters.' So early on it was realized that providing the standard would not be enough--conformance testing for CGM files, as well as for CGM generators and interpreters, would be required to fulfill CALS requirements.

To meet these requirements, NIST/CSL first concentrated its efforts on developing the Application Profile for CGM in CALS (namely MIL-D-28003), to guarantee 100% exchange of CGM files between DoD and contractors. This has been an ongoing task, resulting at present with version A of MIL-D-28003, and an amendment to it. Then to meet the conformance testing requirements, NIST/CSL developed a test requirements document to test metafiles to both the Federal Information Processing Standard for CGM (FIPS 128) and the MIL-D-28003. Test tools were developed, and a conformance testing service for CGM metafiles began in May 1991. This was the first of the three parts of a total CGM conformance test suite.

Next NIST/CSL emphasis switched to conformance testing of CGM generator products, and procedures were developed by NIST/CSL for verifying that a CGM generator product produces conforming metafiles which accurately and correctly define the intended picture. Currently, these procedures are being used in a Beta test environment, with CGM generator product testing to formally begin shortly.

The last part of the total CGM conformance testing environment is to ensure that a CGM interpreter product can correctly and completely parse any CGM file and produce the intended picture. The detailed design specification developed herein by NIST/CSL is the first step in accomplishing this goal.

DISCUSSION

The objective of the CGM interpreter product testing program is to determine whether a given product, in this case a CGM interpreter, can correctly and completely parse any CGM file (that satisfies both FIPS for CGM and MIL-D-28003A), and produce the intended picture.

The approach developed by NIST/CSL is a traditional one based on falsification testing. A set of test CGM files is translated using the CGM interpreter product to be tested. The resulting translation is compared to a hard-copy set of reference pictures and a pass/fail decision is made based on the criteria described in a test script. This strategy is similar to one successfully used in testing conformance of implementations to the GKS (Graphical Kernel System) standard. The testing strategy is based on:

- a set of CGM files;
- a script; and
- a set of hard copy reference pictures.

A design-to-cost approach was adopted in selecting those aspects of CGM interpreter products to be tested. This is in recognition of several facts:

- a very large number of test cases is required for thorough, exhaustive testing to the CGM standard and MIL-D-28003;
- many simpler CGM elements are implemented correctly by most CGM interpreter products; and
- where successful interchange is not possible today, it is almost universally because of one or more of a small set of defects.

Therefore, the detailed strategy selects and prioritizes test cases based on their impact on effective interchange and on the prevalence of incorrect implementations.

CALS USE/IMPACT

This report provides a detailed design specification for testing conformance of CGM interpreter products.

In addition this report provides a prioritized list as well as a description of all the required test cases for a full interpreter testing service. A complete, detailed test case design is included only for the 100 highest priority tests, but a generic description of the remaining cases together with estimates of how many test files and associated images (at least another 100) are required is included.

RECOMMENDATIONS

NIST/CSL recommends strongly that a full testing program for MIL-D-28003A requires testing of CGM interpreters, and that testing must be done to up-to-date versions of both the CGM standard and MIL-D-28003. As new versions of the CGM standard and MIL-D-28003 are released, a maintenance effort is required to update this design, test suite, and supporting test tools to be of use to CALS in the future.

NIST/CSL also recommends that CALS provide funding to complete the additional test cases, test files, and associated hardcopy references and scripts required for a total CGM interpreter product test service.

In addition NIST/CSL recommends that this design and the test cases be used in an actual testing environment to determine its effectiveness in testing CGM interpreters.

ACKNOWLEDGMENT

The editor would like to acknowledge the major technical contributor to this report, Dr. G. Steven Carson of GSC Associates, Inc.

1 Introduction

The task for this CALS deliverable was to develop a strategy and detailed design for CGM Interpreter product testing. The strategy developed provides a prioritized list of CGM capabilities and functions to test. The prioritization is based on:

- impact on effective interchange; and
- prevalence of incorrect implementations.

This report provides a prioritized list as well as a description of all the required test cases for a full interpreter testing service. A complete, detailed test case design is included only for the 100 highest priority tests, but a generic description of the remaining cases together with estimates of how many test files and associated images are required is included. The reference hard copy pictures that should result from the successful translation of these 100 files, and the test script intended for use with the reference pictures to verify acceptable translation of each file, are not included in this report.

This detailed design uses a testing strategy based on:

- a set of CGM files;
- a script; and
- a set of hard copy reference pictures (integrated with the test script).

Section 5 addresses the impact on this testing strategy of:

- versions of the CGM standard;
- color levels in 28003A; and
- combinations and interactions of CGM elements.

Section 6 gives a plan for developing a complete test service, including manpower and time estimates to develop the remaining test cases.

2 Testing strategy

The approach developed by NIST/CSL is a traditional one based on falsification testing. A set of test CGM files is translated using the CGM interpreter product to be tested. The resulting translation is compared to a hard-copy set of reference pictures and a pass/fail decision is made based on the criteria described in a test script. This strategy is similar to one successfully used in the testing of GKS implementations. Thus, the testing strategy based on:

- a set of CGM files;
- a script; and
- a set of hard copy reference pictures.

A design-to-cost approach was adopted in selecting those aspects of CGM interpreter products to be tested. This is in recognition of several facts:

- A very large number of test cases is required for thorough, exhaustive testing to the CGM standard and MIL-D-28003;
- Many simpler CGM elements are implemented correctly by most CGM interpreter products; and
- Where successful interchange is not possible today, it is almost universally because of one or more of a small set of defects.

Therefore, the detailed strategy selects and prioritizes test cases based on their impact on effective interchange and on the prevalence of incorrect implementations.

2.1 Basic principles

Rather than using a few test cases, each of which tests numerous aspects, test cases should mostly focus on a single aspect. This is necessary to allow realistic test scripts with simple pass/fail criteria to be developed. It also makes interaction with the "vendor under test" simpler since errors can be clearly identified and files illustrating them can be supplied.

Another important principle is that, where possible, self-documenting and internally-validating tests should be used. This means that the test images themselves contain labels and marking points that serve to validate the output and reduce the possibility of erroneous test result analysis. For example, tests of circular arcs might use 2 point line segments to indicate the extent of the expected arcs while tests of edge attributes might use test labels to state the expected values.

A final principle is that of incremental testing. If tests are properly divided into sets, then more advanced (and time-consuming) testing need not be done until simpler test sets are passed. For example, extensive testing of a graphical primitive should be deferred until the testing service is sure that the CGM interpreter product correctly implements the basic and abstract CGM data types, as well as all coordinate types and precisions. To do otherwise could waste the testing service's time and lead to hard to diagnose problems and misleading results.

2.2 CGM files

The general principles that were followed in generating CGM test files are as follows:

- Descriptive file names of 8 or fewer characters were used due to file name length restrictions in some primitive systems. For similar reasons all capital letters were used. Standard computer graphics abbreviations like those used in language bindings were used where possible. For example, a file with the name VDCEXT01 might test a VDC arrangement with an inverted Y axis.

- Each test file should itself be a CALS-compliant file (except if error handling is deliberately being tested with non-compliant files).

- Default precisions and types are used as much as possible in test cases. If precisions, types and specification modes are independently varied as graphical primitives and attributes are tested, it is impossible to determine whether problems that are observed are due to the failure to implement the feature being tested or to a failure to correctly implement some underlying data type that it requires.

2.3 Test script

Even with largely self-documenting tests, a test script is essential to insure that relevant aspects are validated in a uniform and consistent manner. Here are the general principles used in developing the test scripts:

- Begin with a simple declarative instruction such as "Interpret file XXX.CGM".
- Number each step sequentially so that a failure report can show exactly which steps failed.

- Include a place to indicate step accomplishment where no pass/fail criteria is involved. This helps the tester keep track of what steps have been accomplished.
- List all items on the output to be validated and include a place to indicate pass/fail results.
- Provide a place to check off that intermediate steps have been accomplished.

An example fragment illustrating these principles is given below:

Step 2. Interpret file VDCEXT01.CGM. _____

Step 2.1 Verify that a blue line is present from the lower left to the upper right of the picture. _____/_____
pass fail

2.4 Hard copy reference pictures

There are several problems inherent in the production of hard copy reference pictures. These are:

- High quality color hard copy is expensive.
- The process of mapping a local representation of the test file to a local printer can itself introduce approximations and distortions that must be accounted for and documented.
- It is highly desirable to be able to produce multiple copies of the test script with included hard copy on demand and at low cost. It is also desirable that single copies of reference images be cheaply producible so that they may be provided to the "vendor under test" to document failures.

Therefore, this approach maximizes the use of black and white test files and uses as little color as possible. Use of a broad spectrum of colors and of non-primary colors is also limited so that almost all test hard copy can be produced quickly and simply on low-cost widely-available dot-matrix and ink-jet printers.

3 Classes of Tests

3.1 Introduction

Some test suites are formulated by tracking through a standard paragraph by paragraph and constructing tests for all requirements. Some problems encountered in creating test suites by this method are:

- they contain many tests cases;
- they check things that are almost always correctly implemented;
- they give equal weight to little-used and widely-used features; and
- they can fail (just as standards themselves sometimes do) to recognize the interactions between parts of the standard.

Therefore these tests are divided into three classes:

- structural capabilities;
- primitive capabilities; and
- attribute capabilities.

Structural capability tests check that the CGM interpreter product can read basic and abstract data types correctly. They also test that delimiter elements, metafile descriptor elements and picture descriptor elements can be understood and processed correctly. Finally, they test that certain concepts with near universal effect on picture production are correct. These include clipping and VDC extent. ***Primitive capability*** tests check the correct implementation of graphical primitives with a focus on basic geometry and specification. ***Attribute capability*** tests focus on determining if attribute elements are correctly interpreted and their values applied to the correct graphical primitives.

3.2 Structural Capability Tests

Structural capability tests check features that affect the interpretation of many different elements. Many CGM interpreter products have serious errors in this category that are a prime barrier to inter-operability. These tests should produce simple, easily verified graphical output. Some incorrect implementations will crash or otherwise behave unpredictably when interpreting some of these test files.

In order of increasing difficulty of test case development, the capabilities to be tested are:

- minimal CGM;
- CGM with no pictures;
- CGM with an empty picture;
- CGM with multiple pictures;
- background color;
- partitioned elements;
- minimal MDR;
- maximal MDR;
- default values;
- coordinate system options (VDC Extent);
- coordinate types;
- precisions;
- specification and selection modes (indexed vs. direct color; bundled vs. individual attributes);
- MFD and PD attribute order;
- clipping; and
- all elements interpretable.

3.3 Primitive Capability Tests

This category of tests is organized into groups based upon specification similarity. The groups are:

- lines;
- markers;
- circular and elliptical specifications;
- circles and ellipses;
- arcs;
- closed arcs;
- filled areas;
- text; and
- cell array.

The tests should be structured to insure that these CGM concepts are correctly implemented:

- all primitives are implemented;
- basic geometry;
- specification points (the values used for fundamental things like the centers of circles and the locations of text strings are right);
- directions and extents (generally vector information giving things like the up direction of text and the extent of arcs);
- limits;
- geometric consistency among primitives (e.g. text is drawn inside a rectangle that coincides with a restricted text extent); and
- degenerate cases.

3.4 Attributes Capability Tests

These tests should be organized into groups based upon specification similarity:

- lines;
- edges;
- markers;
- text;
- filled areas; and
- cell array (note that cell arrays have no attributes per se, but it must be insured that usage of different color specification modes leads to correct cell arrays.)

Within each group, tests should be structured to insure:

- basic effectiveness (i.e. check a few of each to insure that they work);
- all values of finite ranges implemented (e.g., all marker types are implemented);
- default values are implemented correctly; and
- there is geometric consistency among primitives (e.g. width 1 lines overlaid on width 1 rectangles do not "come apart").

Finally, bundled attributes should be tested. These are not often used in CGM generator products in the United States, but can be found in some European products. CALS is of such importance in the European defense community that CGMs from European products may find their way into US systems. Implementation of the default MIL-D-28003 bundle tables should be tested.

3.5 Prioritized list of capabilities

Experience with CGM interpreter products has helped in preparing the prioritized list of capabilities to be tested. Most capabilities require multiple test cases. Each test case will consist of a single CGM file, a test script "fragment", and a reference hard copy. The intent is that the test script fragments can eventually be combined into a single test script.

The initial 100 tests will be derived from the capabilities below, starting in the first priority category at the top and progressing down the list until 100 tests cases have been reached. Section 4.0 expands on the material in this section, giving a detailed design of each test case.

3.5.1 Highest priority capabilities — known shortcomings of products

The highest priority capabilities are listed in order of relative importance with the highest priority item being capability number one (1). A brief justification is given for each high priority capability. The names of specific products with these shortcomings are omitted.

failure to implement primitives or geometric attributes:

1. **restricted and append text** Many popular interpreters fail to implement these elements. All high-quality CGM generators use them.

2. **rotated text** Either rotated text is not supported at all, or only some angles are supported, or certain angles are implemented incorrectly.

3. **text path** Many interpreters support only text path correctly.

4. **elliptical elements** Most implementation have errors, often in arc sense. Technical illustrations from CAD system commonly use this element.

5. **circular elements** Many implementation have errors, often in arc sense.

6. **cell array** Cell array is not supported by most interpreters. It is generated by all graphic arts quality generators.

7. **polygon set** Polygon set is not supported by many interpreters. It is used by some graphics arts quality generators when they degenerate text into polygons. It is also used to represent certain clipped polygons.

failure to implement attributes:

8. **fill styles** Most interpreters do not support patterns. Most graphic arts quality generators use patterns. Also many interpreters do not support empty and hollow styles correctly - often using an incorrect edge color.

failure to support modes and precisions:

9. **integer and real values of all precisions** Most interpreters fail to support real precision. Many CALS CGMs are from engineering systems and use real coordinates.

10. **fixed point and floating point coordinates** At least one generator uses fixed point coordinates.

11. **direct color** Most graphics arts systems use direct color.

12. **font list** Most interpreters do not implement this element.

minimum capabilities not provided:

13. **1024 point polylines and polygons** Several products have much smaller limits.

14. **256 color table entries** Several products have much smaller limits.

15. **254 characters in a string** One product has a limit of 32.

failure to implement structural features correctly:

16. **Metafile Defaults Replacement (MDR)** Many interpreters do not support this element at all. Others do not correctly set all defaults.

17. **partitioned elements** These are very important for two reasons. First, the limited curve and closed figure capabilities of CGM Version 1 force many generators to degenerate complex native objects into polygons or polylines with many points. This in turn forces the use of partitioned elements. Second, large cell arrays must be partitioned. Most CGM interpreter products do not implement partitioned elements correctly.

18. **inverted y axis** Inverted y axes are present in several commercial products, especially ones based on X-windows or Macintosh systems.

19. **long-form string counts** Even though the maximum number of characters in a single string is 254, and that count can be encoded in a single octet, at least one CGM generator product always uses long form strings.

Metafile Descriptor (MFD), Picture Descriptor (PD) and Control elements:

20. **cannot interpret elements in any order** Elements are not handled at all or are handled incorrectly if they do not appear in the exact order listed in the CGM standard.

21. **ignore certain settings or only support a subset** The following important CGM elements not covered elsewhere in the highest priority tests are very often not implemented or are implemented incorrectly: scaling mode, colour value extent, and various combinations of index precision, color index precision, color precision, and color value extent.

22. **failure to correctly reset defaults at the beginning of a picture.**

3.5.2 Second priority capabilities — structural

The second priority capabilities are the remaining structural capabilities not included in the highest priority tests. In priority order, the capabilities to be tested (and specifically the CGM elements to be covered) are:

1. **all elements interpretable** No legal CGM element should cause a CGM interpreter product to crash, malfunction, or generate a message that states the element is "not supported"; this includes testing for the ability to recognize and skip ESCAPE and GDP elements, even though these are not allowed in MIL-D-28003, as well as NO-OP and MESSAGE elements.

2. **clipping** Clipping rectangles must be effective on all graphical primitives. Elements to be tested are:

   CLIP RECTANGLE
   CLIP INDICATOR

3. **CGM with multiple pictures** The product must provide a way to extract each picture.

4. **background color** The product must be able to read and use background colors. The element to be tested is:

   BACKGROUND COLOUR

5. **specification and selection modes** Bundled attributes are often not supported. The elements to be tested are:

   ASPECT SOURCE FLAGS

6. **remaining MFD, PD and control elements and values** This includes all 16 fonts allowed in the font list, to insure they are supported and mapped to "equivalent" fonts. The elements to be tested are:

   Metafile Descriptor Elements:

METAFILE VERSION
METAFILE DESCRIPTION
VDC TYPE
INTEGER PRECISION
REAL PRECISION
INDEX PRECISION
COLOUR PRECISION
COLOUR INDEX PRECISION
MAXIMUM COLOUR INDEX
COLOUR VALUE EXTENT
METAFILE ELEMENT LIST
FONT LIST
CHARACTER SET LIST
CHARACTER CODING ANNOUNCER

Picture Descriptor Elements:

SCALING MODE

Control Elements:

AUXILIARY COLOUR
TRANSPARENCY

7. **minimal CGM** Default values should be safely used when none are explicitly specified.

8. **coordinate system options** Only inverted Y axis was tested with highest priority test cases. Various values and combinations of VDC extent must be allowed and correctly mapped to the full extent of the picture. Any coordinate types and precisions missed in the highest priority tests should be picked up here. The elements to be tested are:

   VDC EXTENT
   VDC INTEGER PRECISION
   VDC REAL PRECISION

9. **MDR testing, including multiple MDR elements** Minimal MDR must be tested in highest priority cases. The element to be tested is:

   METAFILE DEFAULTS REPLACEMENT

10. **CGM with no pictures**

11. **CGMs with an empty picture** These include those with no graphical primitives and ones with no picture body.

12. **default values** Resetting defaults at beginning of picture must be tested in highest priority cases.

3.5.3 Third priority capabilities — graphical primitives

The third priority capabilities to be tested are the remaining graphical primitives not included in the highest priority tests. This category of tests will be organized into groups based upon specification similarity. The tests will be structured to insure that:

- o all primitives are implemented;
- o basic geometry is correctly implemented;
- o specification points (the values used for fundamental things like the centers of circles and the locations of text strings) are faithfully translated;
- o directions and extents (mostly vector information giving things like the up direction of text, the extent of arcs, and the radius of circular elements) are correctly implemented;
- o there is geometric consistency among primitives (e.g. restricted text is drawn inside a rectangle that coincides with the restricted text extent.); and
- o degenerate cases are handled gracefully.

The capability groups and the primitives to be tested under each are:

1. **lines** Check that the line drawn is centered on the geometric locus, and check line width. Elements to be tested are:

   POLYLINE
   DISJOINT POLYLINE

2. **markers** Check that markers are centered on the specification point, and check marker size. The element to be tested is:

   POLYMARKER

4. **circles and ellipses** Check geometric integrity. Elements to be tested are:

   CIRCLE
   ELLIPSE

5. **arcs** Test the effect of more complex VDC spaces, especially an inverted y-axis, on arc sense, and check geometric integrity. Elements to be tested are:

   CIRCULAR ARC CENTRE
   CIRCULAR ARC 3 POINT
   ELLIPTICAL ARC

6. **closed arcs** Basics are tested with highest priority tests, and check closure types. Elements to be tested are:

   CIRCULAR ARC 3 POINT CLOSE
   ELLIPTICAL ARC CLOSE
   CIRCULAR ARC CENTRE CLOSE

7. **filled areas** Test geometric integrity. Elements to be tested are:

   POLYGON
   POLYGON SET
   RECTANGLE

8. **text** Test append text here, including append text with a single null character; and test geometric text attributes like CHARACTER SPACING and CHARACTER HEIGHT. Elements to be tested are:

   TEXT
   RESTRICTED TEXT
   APPEND TEXT

9. **cell array** Test the "path" of the cells and their "line progression." At least one generator inverts cell array direction even though it uses an upright coordinate system for other primitives. The element to be tested is:

   CELL ARRAY

3.5.4 Fourth priority capabilities — graphical attributes

The fourth priority capabilities to be tested are the remaining attributes not included in the first priority tests. Within each group, tests will insure:

- basic effectiveness (i.e. check a few of each to insure that they work);
- all values of finite ranges are implemented (e.g. all marker types are implemented); and
- the default values are correct.

Finally, additional tests should be run on bundled attributes. (Basic tests were conducted under structural tests.)

Tests of capabilities in this category are organized into groups based upon specification similarity. A list of attributes that might be tested if not covered thoroughly elsewhere is given.

1. **lines** Elements to be tested are:

   LINE BUNDLE INDEX
   LINE TYPE
   LINE WIDTH
   LINE COLOUR

2. **edges** Elements to be tested are:

   EDGE BUNDLE INDEX
   EDGE TYPE
   EDGE WIDTH
   EDGE COLOUR
   EDGE VISIBILITY

3. **markers** Elements to be tested are:

   MARKER BUNDLE INDEX
   MARKER TYPE
   MARKER SIZE
   MARKER COLOUR

4. **text** Elements to be tested are:

   TEXT BUNDLE INDEX
   TEXT FONT INDEX
   TEXT PRECISION
   CHARACTER EXPANSION FACTOR
   CHARACTER SPACING
   TEXT COLOUR
   CHARACTER HEIGHT
   CHARACTER ORIENTATION
   TEXT PATH
   TEXT ALIGNMENT
   CHARACTER SET INDEX
   ALTERNATE CHARACTER SET INDEX

5. **filled areas** Elements to be tested are:

   FILL BUNDLE INDEX
   HATCH INDEX
   PATTERN INDEX
   FILL REFERENCE POINT (never used in commercial products)
   PATTERN TABLE
   PATTERN SIZE

6. **test for misapplied attributes** Test that inheritance of attributes is correctly applied to primitives.

4 Detailed Design of First 100 Test Cases

This section describes the first 100 test cases. A description of how each test case is to be created is given and an illustration of the test picture is included where it is helpful. At the end of the title of each section the total number of test cases in that section is given in parenthesis. The purpose of each test case is described and each is assigned a unique 8 character name. In addition, the cases are numbered from 1 through 100.

The total numbers of tests by category are:

Category	Number of tests
Highest priority test cases:	
Failure to implement primitives	22
Failure to implement attributes	5
Failure to support modes and precisions	12
Minimum capabilities not provided	4
Failure to implement structural features	4
MFD and PD elements	5
Subtotal:	52
Second priority test cases:	
All elements interpretable	2
Clipping	4
CGM with multiple pictures	1
Background color	2
Aspect source flags	10
Remaining MFD, PD & control element values	13
Minimal CGM	1
Coordinate system options	7
MDR testing, including multiple MDR el's	2
CGM with no pictures	1
CGM with empty pictures	3
Default values	1
Subtotal:	48
Total	100

4.1 Failure to implement primitives

4.1.1 Restricted and append text (2 test cases)

Case 1 (RESTXT01): This case will draw a single string of restricted text to detect the failure to implement the RESTRICTED TEXT primitive. Width 1 lines should be exactly aligned on the restricted text extent. Width 1 lines should identify the text position point. Figure 1 illustrates this test case.

Case 2 (APNTXT01): This case will be similar to RESTXT01, only using at least one piece of append text added to the initial restricted text string. It will detect the failure to implement APPEND TEXT.

Figure 1. Restricted text test

4.1.2 Rotated text (3 test cases)

Implementations often make mistakes only for certain values of character orientation due to errors in trigonometric computations. Thus, it is important to test some angles in each quadrant as well as the cardinal cases. Further, some implementations do not support restricted or appended rotated text correctly, so a test case should be included for that too. Note that typographic quality systems, such as those used in CALS, use only a 90 degree angle between the character up and character base vectors, making it unnecessary to test "skewed" orientations.

Case 3 (ROTTXT01): This case will test the four cardinal orientations.

Case 4 (ROTTXT02): This case will test one "random" angle in each of the four quadrants.

Case 5 (ROTTXT03): This case will test one cardinal and one rotated angle of restricted and of restricted plus append text.

Positioning lines should be overlaid on all text indicating the character base and character up directions. Figure 2 illustrates the first test case while Figure 3 illustrates the second.

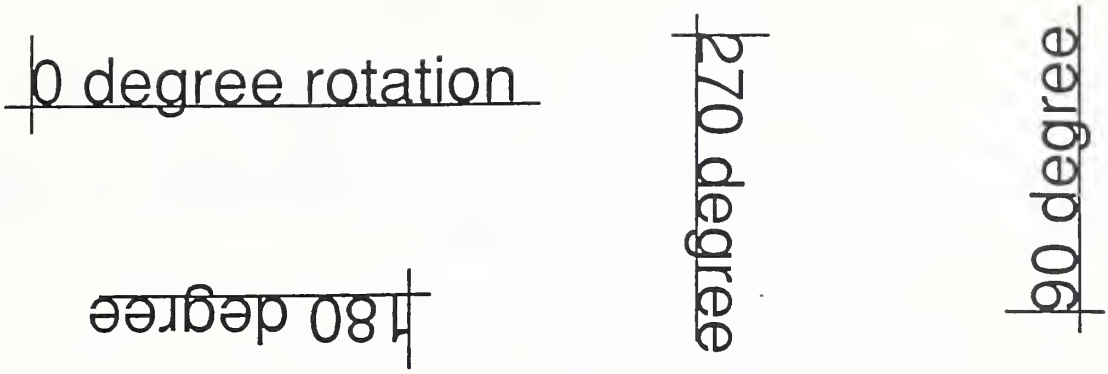


Figure 2. Cardinal text rotations

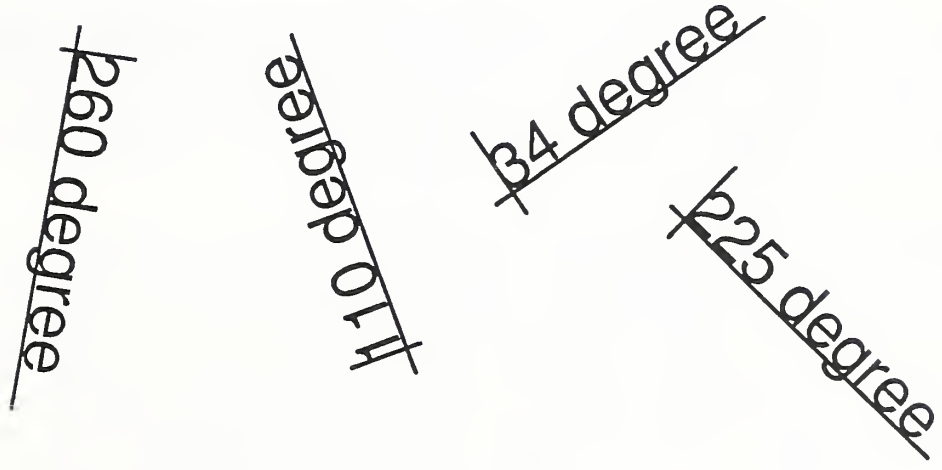


Figure 3. "Random" text rotations

4.1.3 Text path (2 test cases)

Case 6 (TXTPTH01): This case will include one example of each of the four possible text paths to detect the failure to implement this attribute.

Case 7 (TXTPTH02): This case will check text path in combination with RESTRICTED TEXT and rotated text. These cases are more difficult than the four simplest cases tested in TXTPTH01. Consequently, more interpreters are likely to do them incorrectly.

4.1.4 Elliptical elements (7 test cases)

These tests are designed to uncover first the failure to implement the primitive and second any incorrect implementation in critical areas. Since some implementations may act unpredictably when unsupported elements are encountered, test files should be dedicated to testing a single type of element. The key things to be checked for are:

1. Is the primitive implemented?
2. Is the geometry about right?
3. Are the extents of the arcs correct?
4. Is closure implemented correctly?

The three CGM elements to be tested are:

ELLIPSE
ELLIPTICAL ARC
ELLIPTICAL ARC CLOSE

Case 8 (ELLIPS01): This case will check the ELLIPSE element itself. It is important to check both "rotated" ellipses (i.e., ones whose major and minor axes are not parallel to the VDC axes) and "skewed" ellipses (i.e., ones whose major and minor axes are not perpendicular). More interpreters fail to implement the latter two correctly.

The second, third and fourth test cases will check the ELLIPTICAL ARC element. There are several very common mistakes in implementing this element. First, test that arc sense is derived from the CDP1 to CDP2 direction and not from the start vector to end vector direction. Second, test that arcs of various lengths that start and end in various quadrants are all implemented correctly. It is not unusual for trigonometry mistakes to become evident only for certain combinations of angles. Finally, arcs based on ellipses of various orientations and skewness should be checked. This will require 3 test cases since there are so many combinations to be checked.

Case 9 (ELLARC01): This case tests ellipses whose axes are perpendicular and oriented parallel to the x and y axes. It attempts to detect failure to implement the primitive or to do certain combinations of start and end angles correctly.

Case 10 (ELLARC02): This case will check rotated, but non-skewed ellipses.

Case 11 (ELLARC03): This case will check a few skewed ellipses. Figures 4 and 5 illustrate these test cases.

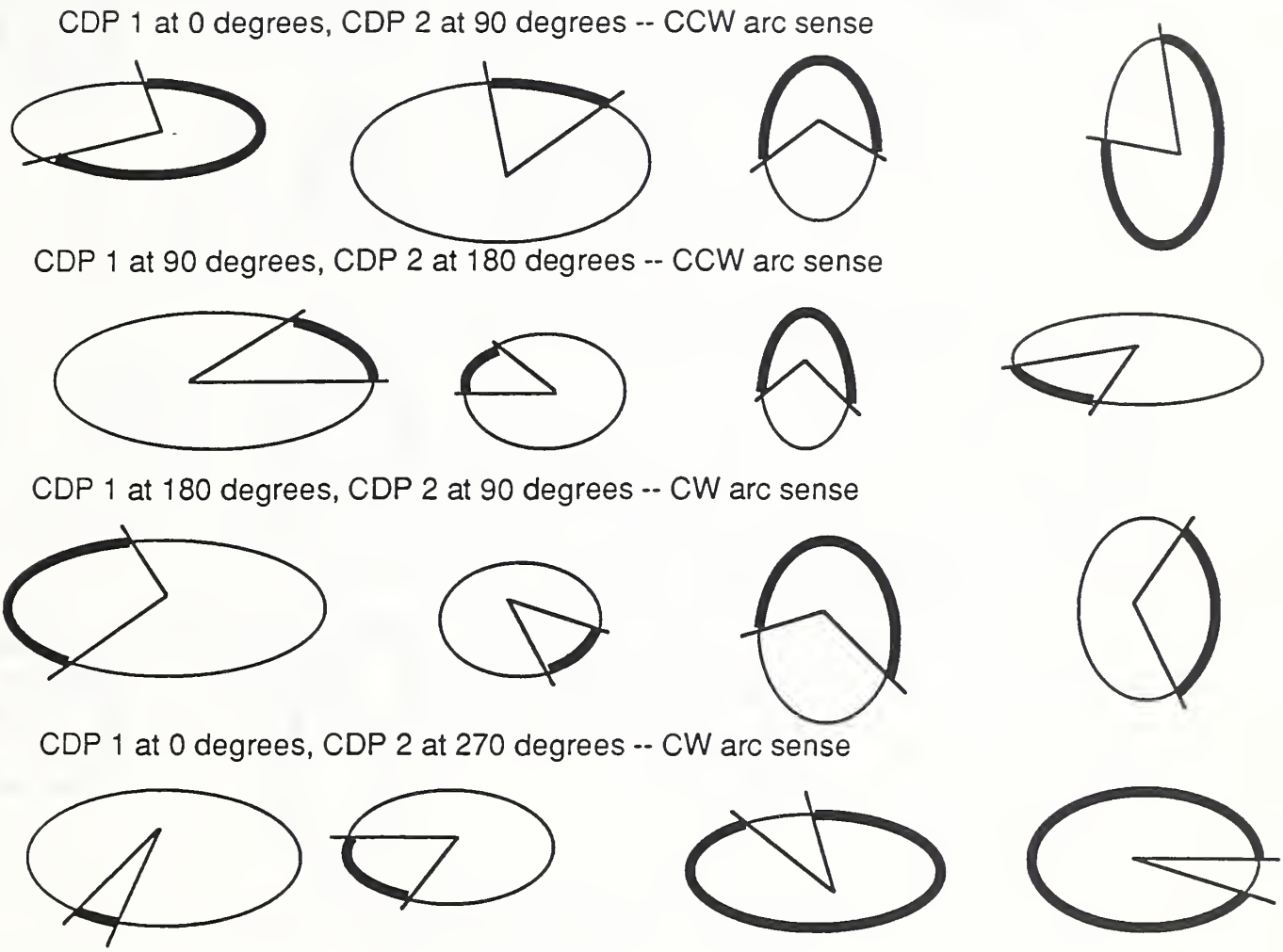


Figure 4. Normal elliptical arcs

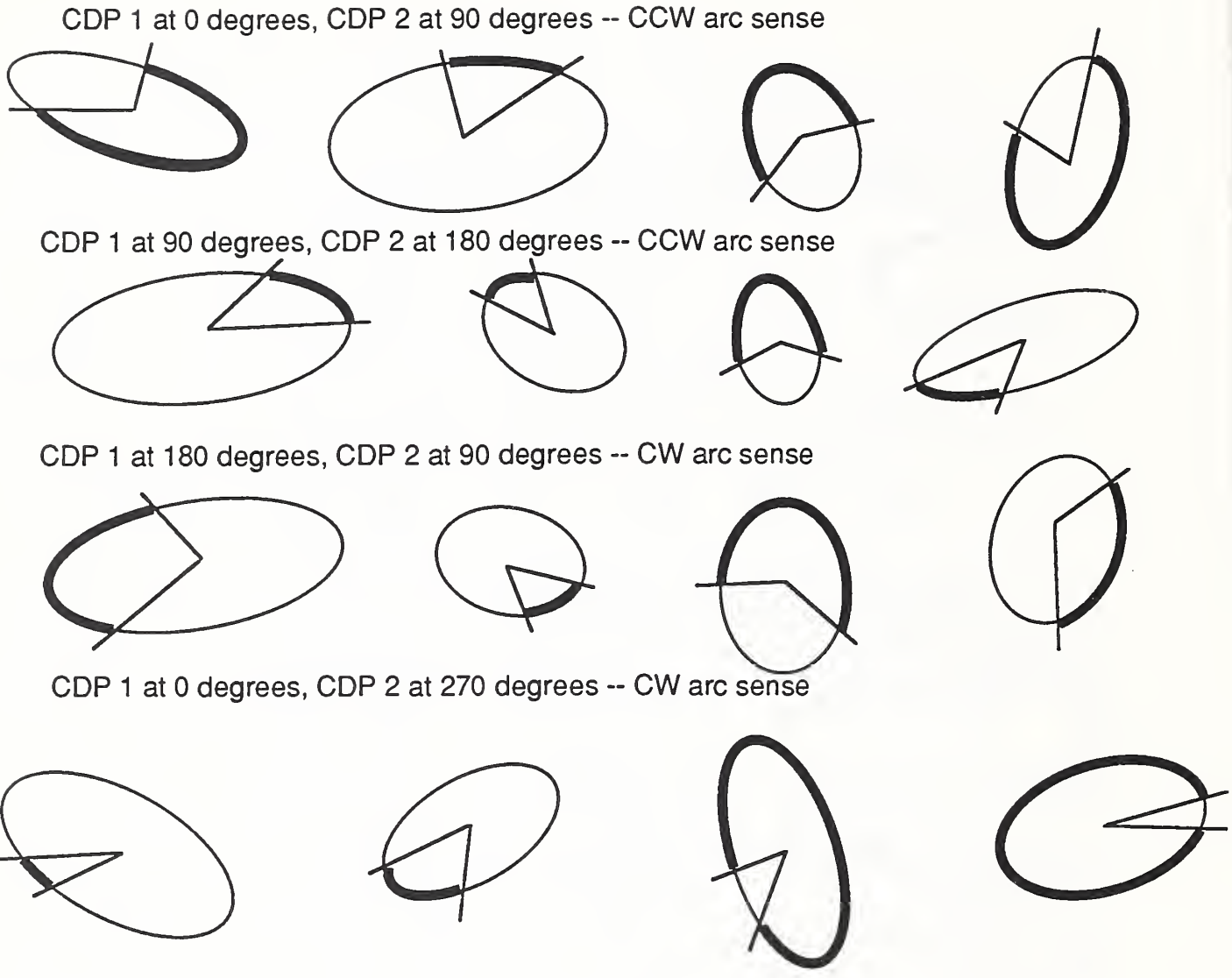


Figure 5. Rotated elliptical arcs

The fifth, sixth and seventh test cases will check the ELLIPTICAL ARC CLOSE element. They will be based on the ELLIPTICAL ARC tests, adding closure types and fills in a random way.

Case 12 (ELARCC01): This case will determine if the ELLIPTICAL ARC CLOSE element is implemented. Both pie and chord closures should be checked as well as various interior styles.

Case 13 (ELARCC02): This case will check a few rotated ELLIPTICAL ARC CLOSE elements.

Case 14 (ELARCC03): This case will check a few skewed ELLIPTICAL ARC CLOSE elements.

4.1.5 Circular elements (5 test cases)

These tests are designed to uncover both the failure to implement the primitive and any incorrect implementation in critical areas. Since some implementations may act unpredictably when unsupported elements are encountered, test files should be dedicated to testing a single type of element. The key things to be checked for are:

1. Is the primitive implemented?
2. Is the geometric specification faithfully translated?
3. Are the extents of the arcs correct?
4. Is closure implemented correctly?

The five CGM elements to be tested are:

CIRCLE
CIRCULAR ARC CENTRE
CIRCULAR ARC 3 POINT
CIRCULAR ARC 3 POINT CLOSE
CIRCULAR ARC CENTRE CLOSE

Additional coverage of the CIRCLE element will be gained since it is used to test the geometric integrity of the arc elements.

Case 15 (CIRCLE01): This case will check that the circle primitive is implemented, and that the center and radius are faithfully translated. This can be accomplished by using width 1 lines to mark the center and extents of the circle. Figure 6 illustrates this test case.

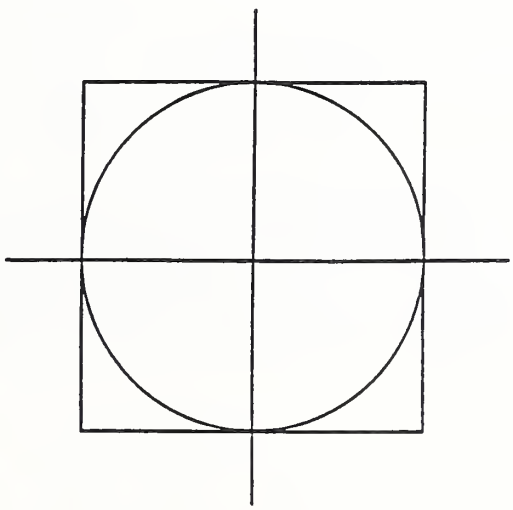


Figure 6. Basic CIRCLE test

Case 16 (CIRARC01): This case will check the CIRCULAR ARC CENTRE element. It is important to test various combinations of start and end vector as well as start and end vectors whose "end points" do not lie on the circle itself. Lines and circles should be used to indicate arc extent and make the pictures self-validating. Figure 7 illustrates this test case.

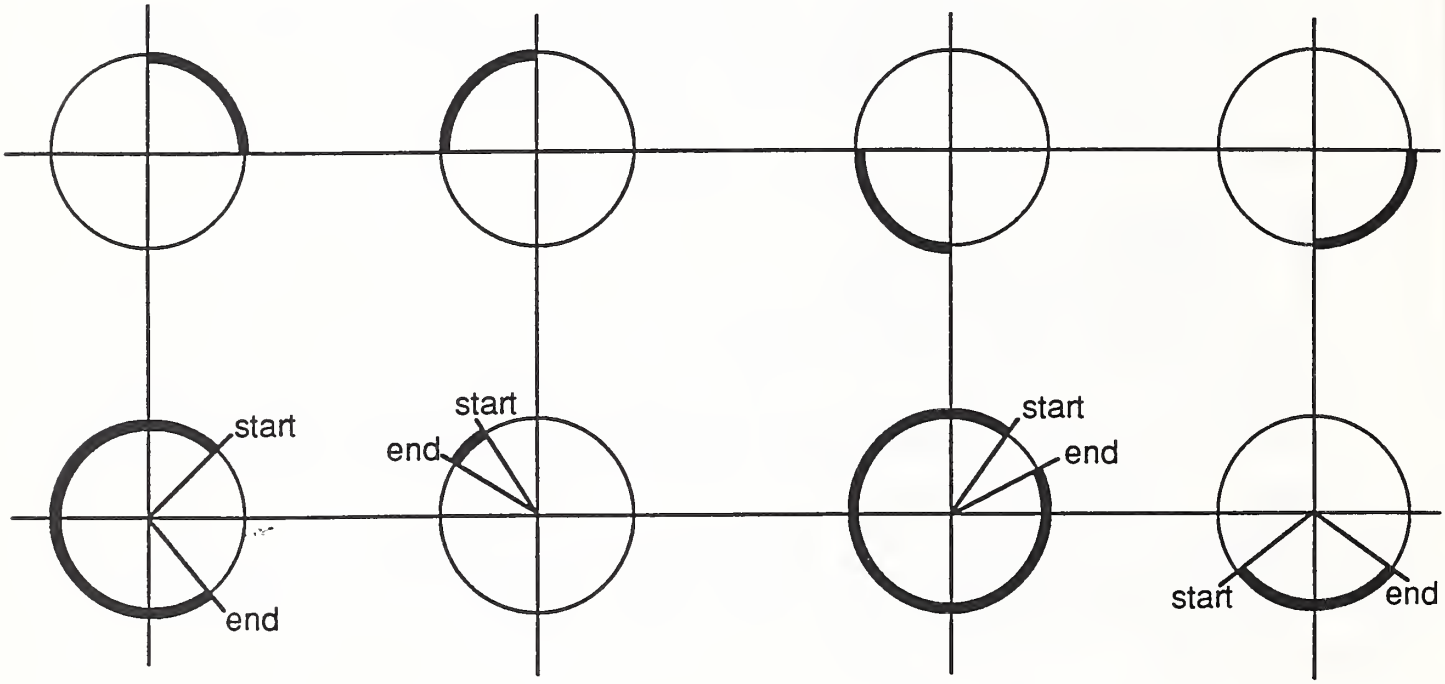


Figure 7. CIRCULAR ARC CENTRE test

Case 17 (CRARCC01): This case will check the CIRCULAR ARC CENTRE CLOSE element. It is important to test various combinations of start and end vector as well as start and end vectors whose "end points" do not lie on the circle itself. Lines and circles should be used to indicate arc extent and make the pictures self-validating. Figure 8 illustrates this test case.

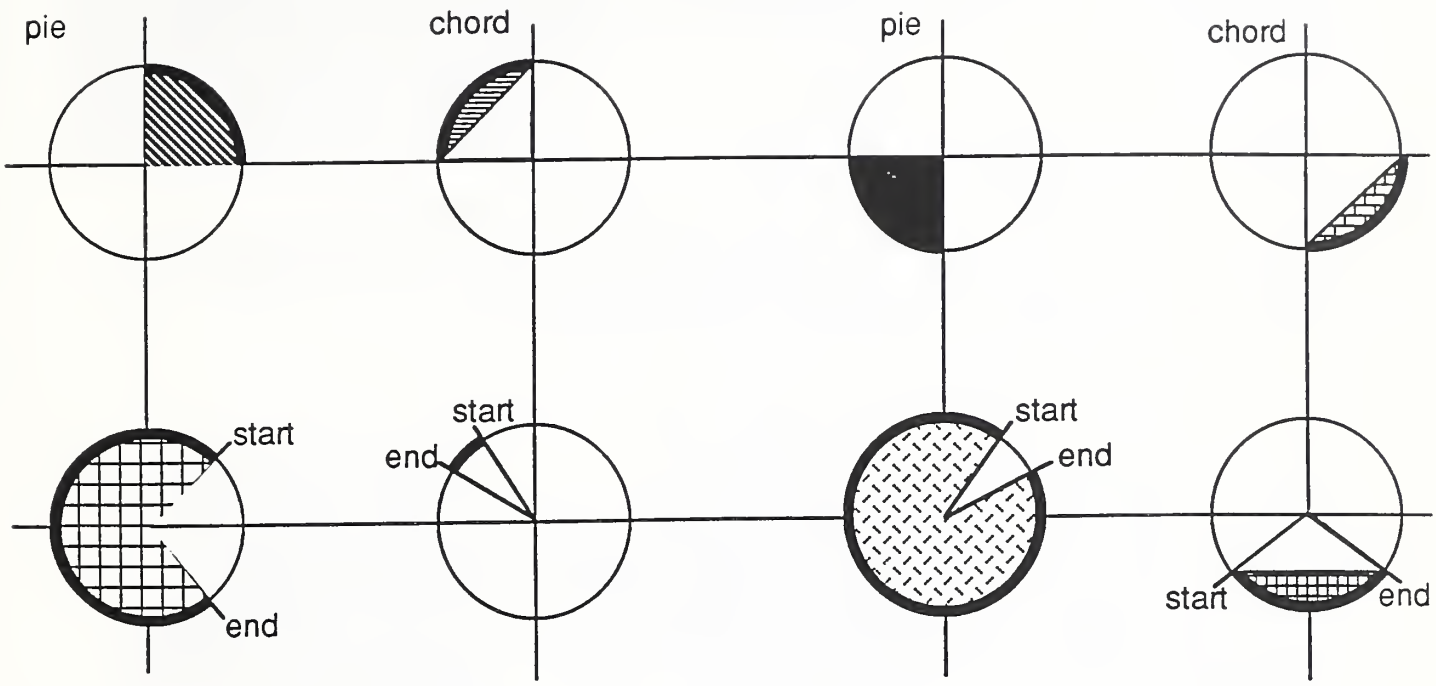


Figure 8. CIRCULAR ARC CENTRE CLOSE test

Case 18 (CRARC301): This case will check the CIRCULAR ARC 3 POINT element. The basic test picture looks like the CIRCULAR ARC CENTRE test. In this case, it is important that both clockwise (CW) and counterclockwise (CCW) directions be tested, since the arc sense is determined by the arrangement of the points and not the positive angular direction.

Case 19 (CRAR3C01): This case will check the CIRCULAR ARC 3 POINT CLOSE element. The basic test picture looks like the CIRCULAR ARC CENTRE CLOSE test. In this case, however, it is important that both CW and CCW directions be tested, since the arc sense is determined by the arrangement of the points and not the positive angular direction.

4.1.6 Cell array (2 test cases)

Case 20 (CELARY01): This case will check a simple black and white cell array. Since we are looking for failure to implement the primitive, the test need not be complex or include a lot of points.

Case 21 (CELARY02): This case will check a color cell array. Again, this test need not be large or complex since it is designed to uncover the failure to implement the primitive at all.

4.1.7 Polygon set (1 test case)

Case 22 (PLGSET01): This case will include a polygon set using all four possible values of the edge out flag. It need not be complex since again we are looking at this stage of testing for a complete failure to implement the primitive.

4.2 Failure to implement attributes

4.2.1 Interior styles (5 test cases)

These should be 5 test cases, one for each of the 5 interior styles allowed in the CGM standard. Each test case should contain two copies of three objects — a rectangle, a polygon, and a circle — one with edge visibility on and the other with edge visibility off. These tests must be done in color since correct implementation of hollow and empty styles can only be checked if distinct fill and edge colors are used. Figure 9 shows the intended picture from the pattern test case.

Case 23 (INTSTL01): This case should test solid interior style.

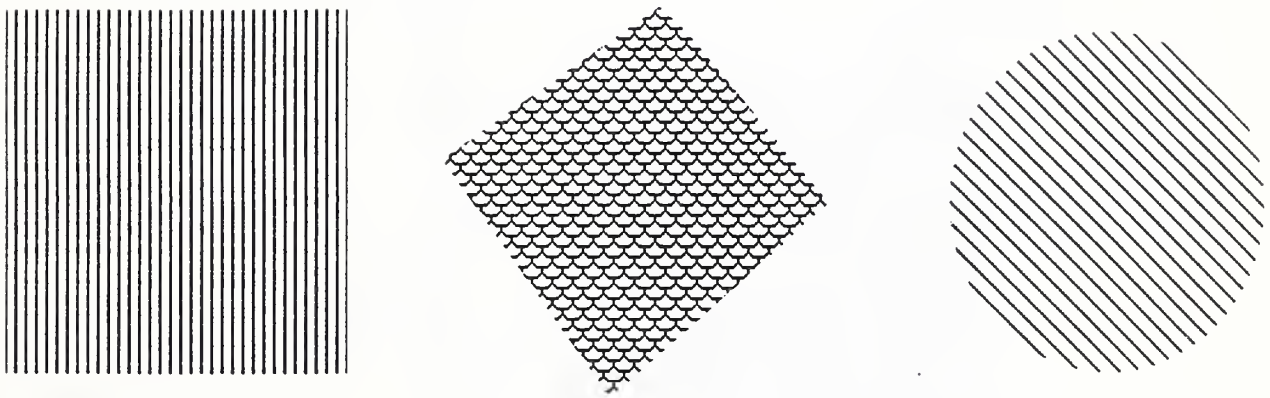
Case 24 (INTSTL02): This case should test hollow interior style.

Case 25 (INTSTL03): This case should test empty interior style.

Case 26 (INTSTL04): This case should test hatch interior style.

Case 27 (INTSTL05): This case should test pattern interior style.

Edge visibility on



Edge visibility off

Figure 9. Pattern fill tests

4.3 Failure to support modes and precisions

4.3.1 Integer and real values of all precisions (6 test files)

The relevant CGM elements and values whose correct interpretation is to be ascertained are:

REAL PRECISION
(1,16,16) fixed point and (0,9,23) floating point
COLOUR PRECISION
8 and 16
COLOUR INDEX PRECISION
8 and 16

Separate test files should be used for each element to be tested with the remaining elements set to default values. This will

allow isolation of the problem(s) if several elements are not correctly implemented. Each test file should contain some simple output whose correct translation relies on the precision being tested. Scaled line widths and marker sizes are used since they are convenient, testable capabilities that depend on real numbers.

Case 28 (REALPR01): This case will check fixed point (16,16) reals.

Case 29 (REALPR02): This case will check floating point (0,9,23) reals.

Case 30 (COLRPR01): This case will check 8 bit colour precision and 8 bit colour index precision.

Case 31 (COLRPR02): This case will check 8 bit colour precision and 16 bit colour index precision.

Case 32 (COLIPR01): This case will check 16 bit colour precision and 8 bit colour index precision.

Case 33 (COLIPR02): This case will check 16 bit colour precision and 16 bit colour index precision.

4.3.2 Fixed point and floating point coordinates (3 test cases)

The relevant CGM elements and values whose correct interpretation is to be ascertained are:

VDC TYPE
integer and real
VDC INTEGER PRECISION
16 and 32
VDC REAL PRECISION
(1,16,16) fixed point and (0,9,23) floating point

Separate test files should be used for each element to be tested with the remaining elements set to default values. This will allow isolation of the problem(s) if several elements are not correctly implemented. A single line drawn from lower left to upper right will detect the failure to implement these values.

Case 34 (VDCINT01): This case will check 32 bit VDC INTEGER PRECISION. (16 bit precision is covered in most other cases since it is the default.)

Case 35 (VDCRPR01): This case will check 32 bit fixed point (16,16) VDC REAL PRECISION.

Case 36 (VDCRPR02): This case will check 32 bit floating point (0,9,23) VDC REAL PRECISION.

4.3.3 Direct color (1 test case)

Case 37 (DIRCOL01): This case will set a color selection mode of direct. Then several objects should be drawn as the directly specified colors are switched. The test can be done in black and white as illustrated in Figure 10.

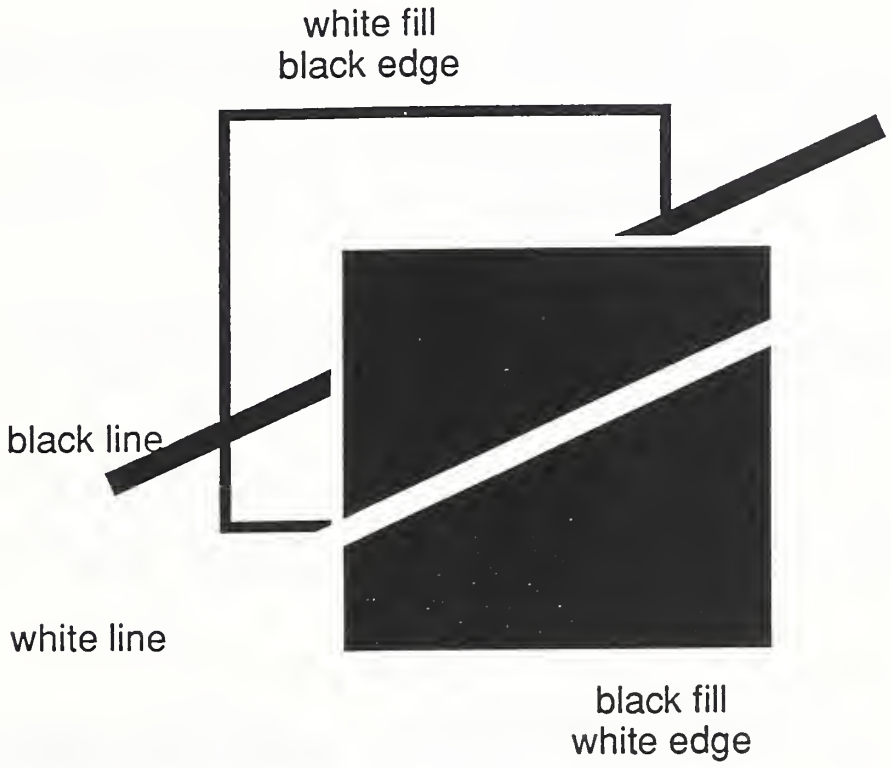


Figure 10. Direct color test

4.3.4 Font lists (2 test cases)

Several of the 16 Hershey fonts listed in MIL-D-28003, such as the German, Cyrillic, and Gothic, are not likely to be seen in DoD technical manuals and are not worth testing for at this highest priority level. What is important to test for are that:

- the serif and sans serif fonts are mapped to local equivalents if available;
- attributes such as relative boldness and italic slant are mapped to local equivalents if available; and
- the Greek characters are mapped to a local Greek font if one is available.

Further, it is unnecessary to exhaustively test the mapping of each character in the repertoire at this level. The check is for the basic capability to recognize and process the FONT LIST element. Thus, the mapping of the following Hershey fonts.

Case 38 (FNTLST01): This case will set a FONT LIST containing these four fonts:

- SIMPLEX ROMAN
- DUPLEX ROMAN
- COMPLEX ROMAN
- TRIPLEX ROMAN

Case 39 (FNTLST02): This case will set a FONT LIST containing these four fonts:

- COMPLEX ITALIC
- TRIPLEX ITALIC
- SIMPLEX GREEK
- COMPLEX GREEK

The pictures created by the test files are shown in Figures 11 and 12. Note that only four simultaneous fonts are supported by the overly-restrictive AP.

Figure 11. Font list test 1

Figure 12. Font list test 2

4.4 Minimum capabilities not provided

4.4.1 1024 point polylines and polygons (2 test cases)

Case 40 (POLYLN01): This case will contain a 1024 point polyline. By choosing points that make a basic shape, such as a circular arc, testing can readily show if points have been ignored.

Case 41 (POLYGN01): This case will contain a 1024 point polyline. By choosing points that make a basic shape, such as a circle, testing can readily show if points have been ignored.

Note that either of these tests will also necessarily test partitioned elements.

4.4.2 256 color table entries (1 test case)

Case 42 (COLTAB01): This case will contain a colour table with 256 entries. It should produce some simple self-described graphical output using colour table entries near the end of the table. The color used should be one that is not likely to be the default primary colour. If this test case is done in black and white, then two color table entries—one set to black and the other set to white—should be used. Figure 13 illustrates a picture from a test case that accomplishes this.

Figure 13. Color table test

4.4.3 254 characters in a string (1 test case)

Case 43 (TXTLEN01): This case will contain a TEXT element with 254 characters. Note that this string can and should be coded with the short form string count.

4.5 Failure to implement structural features correctly

4.5.1 MDR (1 test case)

Case 44 (MDRTST01): This case will contain a simple, minimal MDR element to discern whether the implementation supports processing this element. The MDR element should set the default marker type to a circle (instead of an asterisk) since this change is easily distinguished. The test should then draw a single marker (with text explaining that it should be a circle).

4.5.2 Partitioned elements (1 test case)

Case 45 (PARTEL01): This case will test proper implementation of partitioned elements independent of element buffer size. (Proper handling of partitioned elements will also be tested at the same time that 1024 point polygons and polylines are tested.) To insure that the failure is not caused by an overflow, a small element that need not be partitioned should be used in the test.

4.5.3 Inverted y axis (1 test case)

Case 46 (VDCEXT01): This case will use a VDC element with a lower left (first corner) value of (0,1000) and an upper right (second corner) value of (1000,0). The file should contain a single line drawn from (0,1000) to (1000,0). A default orientation text string should be included so that the image is not accidentally inverted. Figure 14 illustrates the design of this test case (on the left) and the common incorrect translation (on the right).

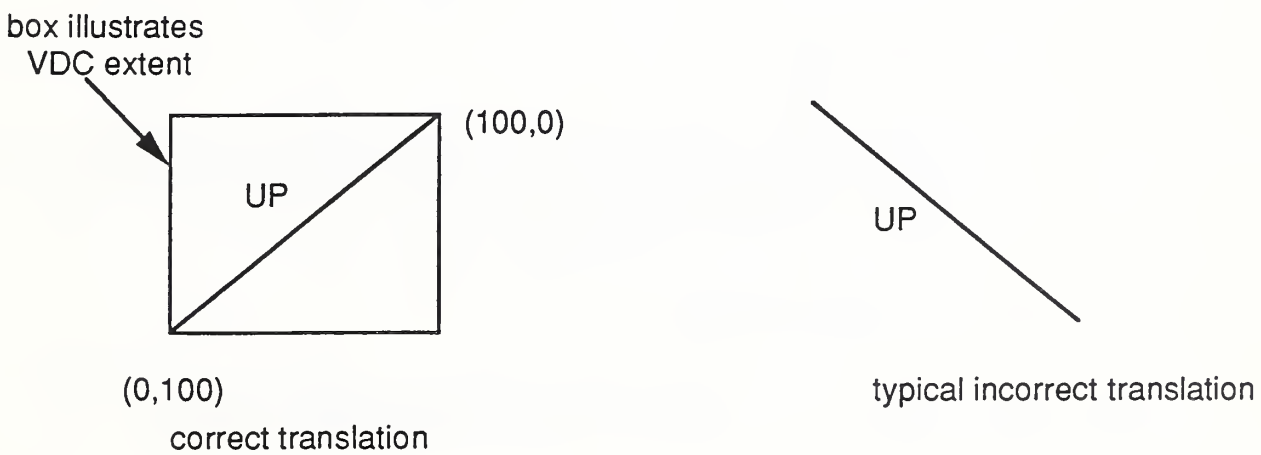


Figure 14. Inverted Y axis

4.5.4 Extended string counts (1 test case)

Case 47 (EXTSTR01): This case will check a CGM file with short text string (about 10 characters) coded with a long form count (that is, the first octet of the string is set to FF(hex) indicating that the next two octets contain the actual count.)

4.6 MFD, PD and Control elements

4.6.1 Cannot interpret elements in any order (1 test case)

The CGM standard lists metafile descriptor elements in a particular order from 1 through 15. Similarly, the seven picture descriptor elements are listed in a given order. Several CGM interpreter products can only accept the elements in that order. For example, these interpreters get confused if INDEX PRECISION (the sixth element) comes after COLOUR PRECISION (the seventh element).

Case 48 (DESORD01): This case will contain metafile and picture descriptors elements not in the most common order. It will do some simple output.

4.6.2 Ignore certain settings or only support a subset (3 test cases)

The test files for this case will each contain all metafile descriptor and picture descriptor elements. The elements will be set to their default values except that the value of one element per test case will be set to a less common value.

4.6.2.1 SCALING MODE

Case 49 (SCLMOD01): This case will set the scaling mode to metric. A simple rectangle should be drawn with dimensions 4 cm. on each side. The output can then be measured to see if the metric instruction is faithfully followed.

4.6.2.2 SPECIFICATION MODES

The two possibilities are absolute values in VDC units and as scaled "multiples" of a nominal device-dependent value.

Case 50 (SPECMD01): This case will set LINE WIDTH, MARKER SIZE, and EDGE WIDTH SPECIFICATION MODES to absolute. The picture should be self-validating through use of geometry points to validate the widths and sizes. Tests of the default values (1/100th of the longest size of VDC EXTENT) can be included. Figure 15 shows the picture from this test case.

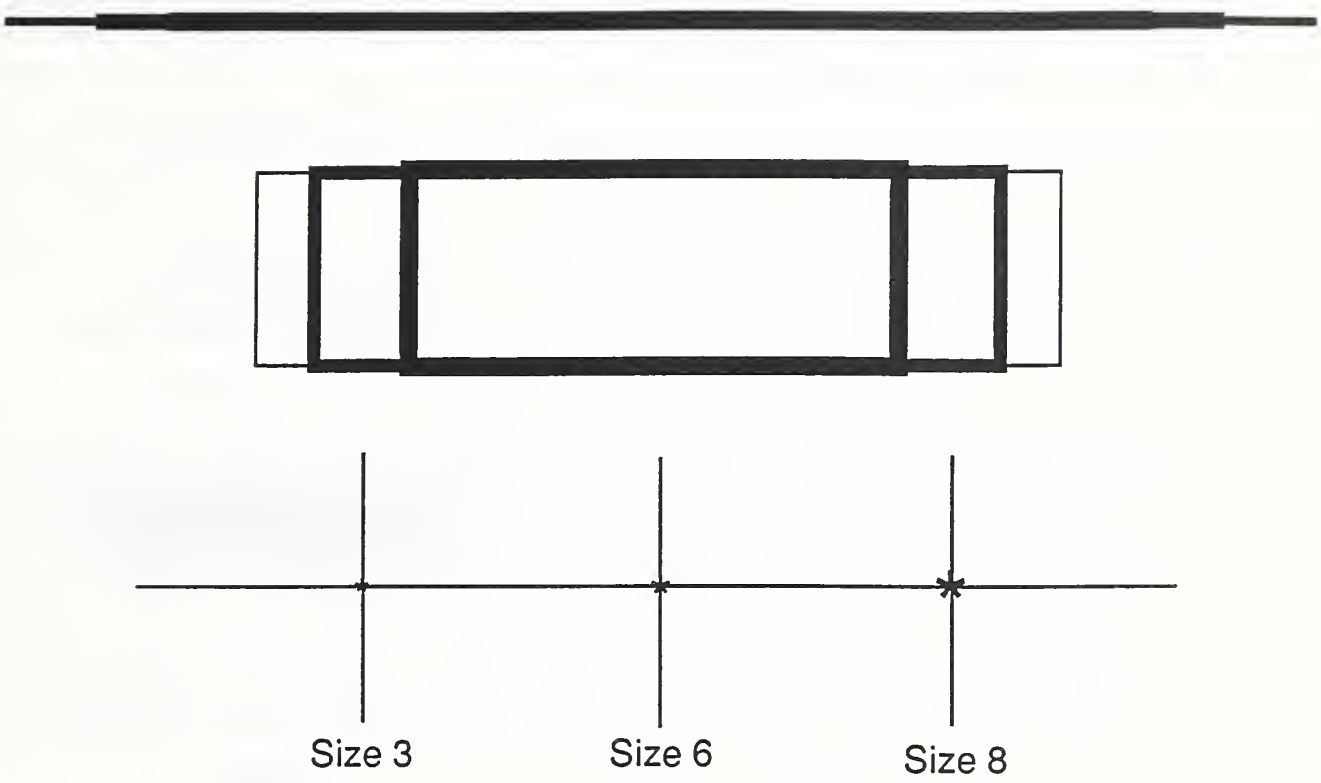


Figure 15. Specification modes

Case 51 (SPECMD02): This case will set LINE WIDTH, MARKER SIZE, and EDGE WIDTH SPECIFICATION MODES all be set to scaled (the default). The same picture as in case SPECMD01 should be drawn.

4.6.3 Failure to correctly reset defaults at the beginning of a picture (1 test case)

Case 52 (DEFALT01): This case will contain two pictures. The first should draw a few objects without setting attributes (to determine the default settings) and then change attributes in a way that the default ones would not be selected for all the objects. The second picture should then draw the same objects with default attributes as the first picture. Figure 16 illustrates the image from this test case. The second picture need only contain the objects on the left side.

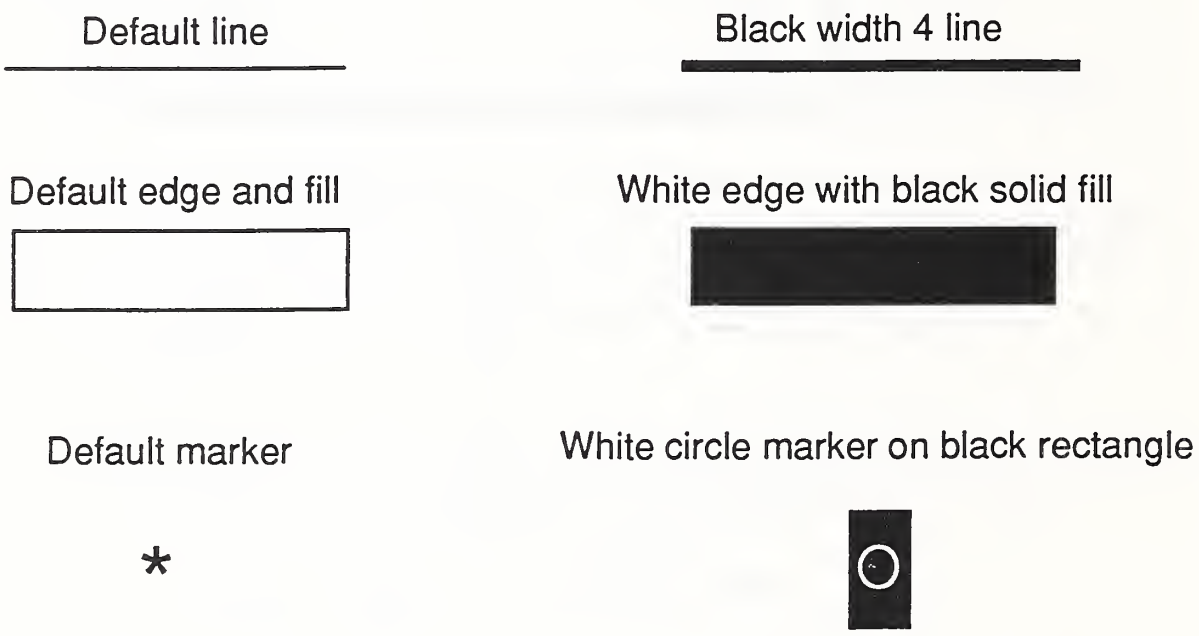


Figure 16. Defaults reset test

4.7 Second priority capabilities — structural

Fifty test cases were required to test the highest priority capabilities. Additional test cases are now drawn from the second priority capabilities list. These are the remaining structural capabilities not included in the highest priority tests.

4.7.1 All elements interpretable (2 test cases)

This test case verifies that no legal CGM element causes a CGM interpreter product to crash, malfunction, or generate a message that states the element is "not supported". Test are included that show a legal ESCAPE or Graphical Drawing Primitive (GDP) element can be recognized and skipped, even though it is not allowed in MIL-D-28003. NO-OP and MESSAGE elements are tested at this time, too.

Case 53 (ALLELM01): This case will contain one instance of each legal CGM element (except NO-OP) to verify that all elements can be accepted.

Case 54 (NOPTST01): This case will contains long strings of NO-OP elements interspersed in all sections of the CGM file.

4.7.2 Clipping (4 test cases)

These test cases verify that the clipping rectangle is settable and has an effect on all primitives. They also verify that clipping can be turned on and off. There will be four test cases, all of which contain identical graphical primitives from each "class" - lines, markers, filled areas, text, and cell array. The first two cases will have the default clipping rectangle (VDC extent rectangle) and one will have the CLIP INDICATOR on while the other has it off. The last two cases will set a smaller clip rectangle that is chosen so that some part of each object should be clipped off.

Case 55 (CLIPNG01): This case will set a clipping rectangle equal to all of VDC Extent but will leave clipping off. Nothing should be clipped.

Case 56 (CLIPNG02): This case will set a clipping rectangle equal to all of VDC Extent and will turn clipping on. Again, nothing should be clipped.

Case 57 (CLIPNG03): This case will set a clipping rectangle equal to a subset of VDC Extent chosen so that some of each primitive in the picture will be clipped when clipping is on. This file will leave clipping off. Nothing should be clipped.

Case 58 (CLIPNG04): This case will set a clipping rectangle equal to a subset of VDC Extent chosen so that some of each primitive in the picture will be clipped when clipping is on. This file will turn clipping off. Part of each primitive should be clipped.

4.7.3 CGM with multiple pictures (1 test case)

Case 59 (MULPIC04): This case will contain a CGM file with multiple, distinguishable pictures. The script will require that all be translated.

4.7.4 Background color (2 test files)

This test case will verify that the product can read and use background colors. Since the default background color will commonly be black or white, two test cases are required.

Case 60 (BGCOLR01): This case will set a black background color and will draw a white object on it.

Case 61 (BGCOLR02): This case will set a white background color and will draw a black object on it.

4.7.5 Aspect source flags (10 test cases)

These test cases will verify that the ASPECT SOURCE FLAGS element functions correctly. Ten test cases are necessary to verify both the static and dynamic behavior of these elements. The first five test cases will tests the default bundle tables for the line, marker, text, fill and edge bundles respectively. Appropriate graphical output in each case will be designed to show that bundle indices 1 through 5 are implemented. Figure 17 illustrates the test case for the line bundle.

Test cases six through 10 will test the dynamic behavior of bundles for each of the five classes of bundles. First, a bundle index will be selected and a default object will be drawn in that index. Next the ASPECT SOURCE FLAGS element will be used to set some attributes to individual. That attribute will be directly set to a non-default value and appropriate output produced to verify that the attribute is taken from the direct value and not the default bundle table. Next all attributes will be set to individual and some output produced. Finally all attributes will be set to bundles again and some more output produced.

Case 62 (ASFTST01): This case will verify the correct implementation of the line bundles.

Case 63 (ASFTST02): This case will verify the correct implementation of the marker bundles.

Case 64 (ASFTST03): This case will verify the correct implementation of the text bundles.

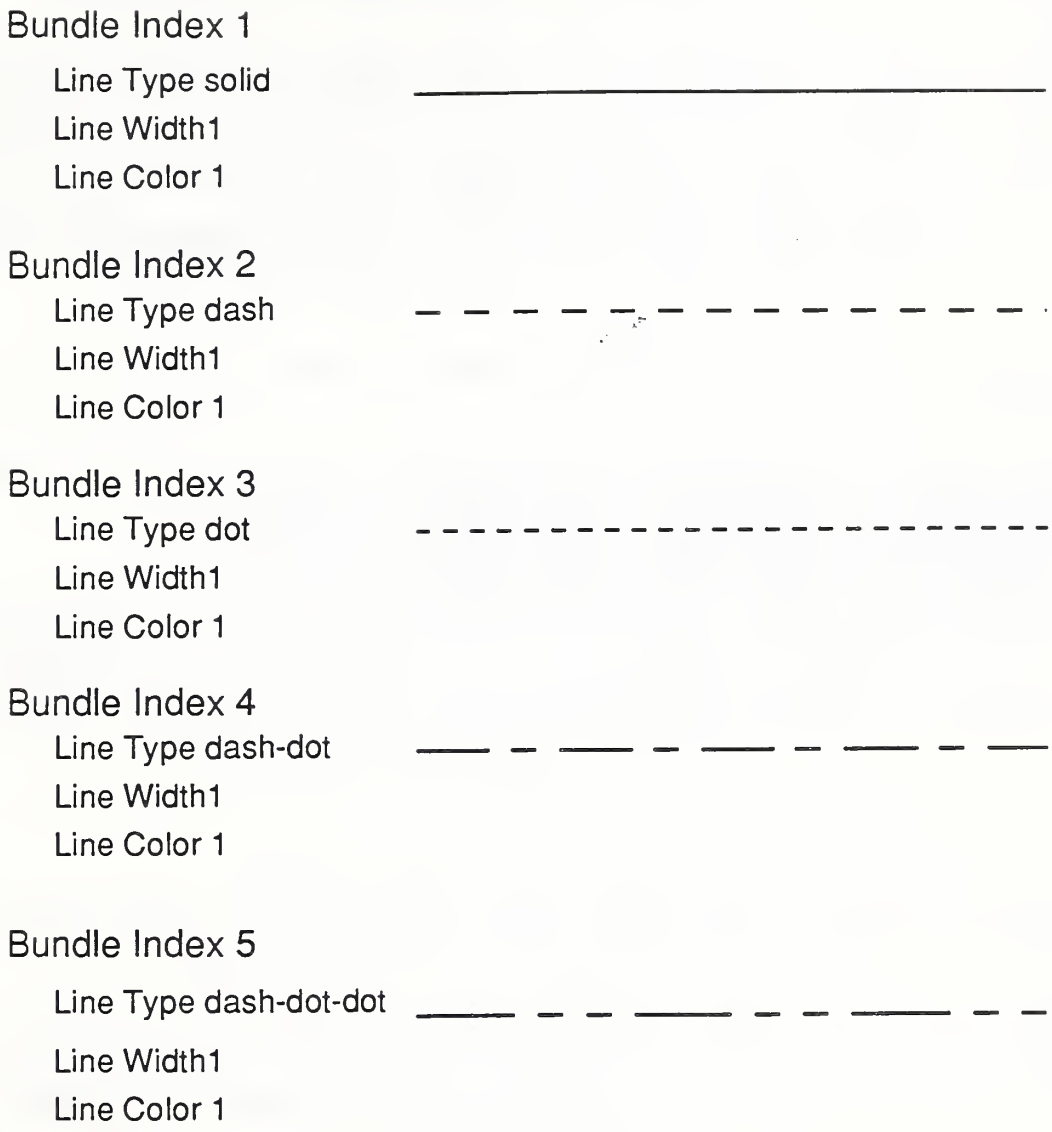


Figure 17. Default line bundle test

Case 65 (ASFTST04): This case will verify the correct implementation of the fill bundles.

Case 66 (ASFTST05): This case will verify the correct implementation of the edge bundles.

Case 67 (ASFTST06): This case will verify the correct dynamic behavior of individual and bundled line attributes.

Case 68 (ASFTST07): This case will verify the correct dynamic behavior of individual and bundled marker attributes.

Case 69 (ASFTST08): This case will verify the correct dynamic behavior of individual and bundled text attributes.

Case 70 (ASFTST09): This case will verify the correct dynamic behavior of individual and bundled fill attributes.

Case 71 (ASFTST10): This case will verify the correct dynamic behavior of individual and bundled edge attributes.

4.7.6 Remaining MFD, PD and control element values (16 test cases)

The next 16 test cases will check MFD, PD and control elements not already tested.

4.7.6.1 METAFILE VERSION (1 test case)

Case 72 (MFVERS01): This case will set metafile version to 5 to ascertain that this element is processed and that a version too "advanced" for the implementation can be recognized. This is the only way to test that this element is processed.

4.7.6.2 METAFILE ELEMENT LIST (3 test cases)

These three test cases will test the ability of the product to accept the most common legal values of this element.

Case 73 (MFELLS01): This case will use a METAFILE ELEMENT LIST containing the code for each possible value.

Case 74 (MFELLS02): This case will use a METAFILE ELEMENT LIST containing the code for the drawing set.

Case 75 (MFELLS03): This case will use a METAFILE ELEMENT LIST containing the code for the drawing-plus-control set.

4.7.6.3 METAFILE DESCRIPTION (1 test case)

Case 76 (MFDESC01): This case will contain a long METAFILE DESCRIPTION with the CALS profile indicative string embedded inside rather than being at the front to verify that it can be recognized inside the string. One way to ascertain that the profile has been recognized is to test some profile-dependent information. One example is the default bundle tables.

4.7.6.4 COLOUR VALUE EXTENT (4 test cases)

These four test cases will check that the COLOUR VALUE EXTENT element is correctly implemented. Two cases will check 8 bit COLOUR PRECISION and two others will test 16 bit COLOUR PRECISION. All four cases will set two "random" color value extents less than the full range allowed in 8 or 16 bits respectively. The test case will draw a set of rectangles filled in solid gray-scale colors ranging from (min-red, min-green, min-blue) through (max-red, max-green, max-blue).

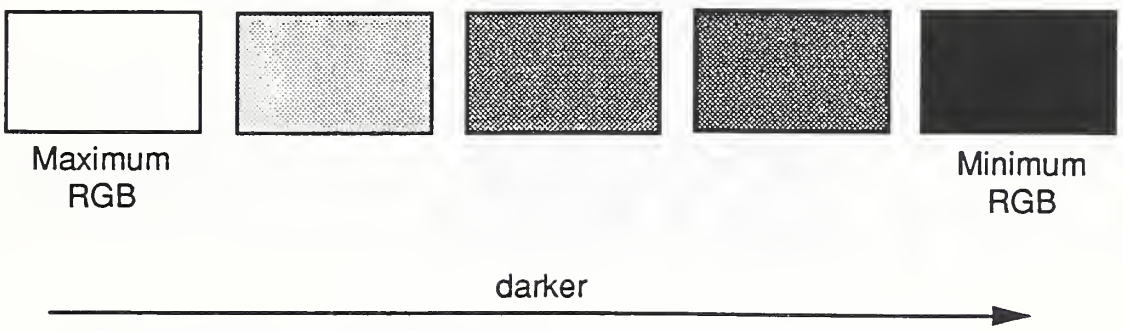


Figure 18. Colour value extent tests

Case 77 (COLVAL01): This case will use 8 bit colour precision. The maximum range is 0 - 255 in this case. This case will set a subset range (colour value extent) from 0 - 100.

Case 78 (COLVAL02): This case will use 8 bit colour precision. The maximum range is 0 - 255 in this case. This case will set a subset range (colour value extent) from 100 - 255.

Case 79 (COLVAL03): This case will use 16 bit colour precision. The maximum range is 0 - 65,535 in this case. This case will set a subset range (colour value extent) from 0 - 1200.

Case 80 (COLVAL04): This case will use 16 bit colour precision. The maximum range is 0 - 65,535 in this case. This case will set a subset range (colour value extent) from 20,100 - 50,000.

4.7.6.5 FONT LIST (2 test cases)

Eight of the fonts were tested in the highest priority capabilities. The remaining eight will be tested in these two test cases. Note that the character sets for these fonts are not defined in MIL-D-28003 and many systems will have no close local equivalent. The best that can be hoped for is a mapping to a local equivalent.

Case 81 (FNTLST03): This case will test fonts 9 through 12 of Table VI of MIL-D-28003.

Case 82 (FNTLST04): This case will test fonts 13 through 16 of Table VI of MIL-D-28003.

4.7.6.6 CHARACTER SET LIST (1 test case)

Case 83 (CHRSET01): This case will verify that characters from the "right-hand" part of the code table can be successfully translated. It should use a single string in a standard font (like SIMPLEX ROMAN) that contains characters—such as Ä, Á, and Â —from the right hand part of the table.

4.7.6.7 AUXILIARY COLOUR and TRANSPARENCY (2 test cases)

Even though MIL-D-28003 has inadequacies that hinder their effective use, these elements should be tested for completeness. MIL-D-28003 allows transparency to be set only to on, even though the "native" mode of most quality graphics arts systems has transparency set to off. Thus, the contents of the AUXILIARY COLOUR element should be ignored since they are only used when transparency is off. By overlaying the figures on a black background the effectiveness of these elements can be ascertained.

Case 84 (TRANSP01): This case will set transparency to on and auxiliary color to white. Several objects will be drawn over a black background so that any use of auxiliary color in their rendition can be detected. This test case is illustrated in Figure 19.

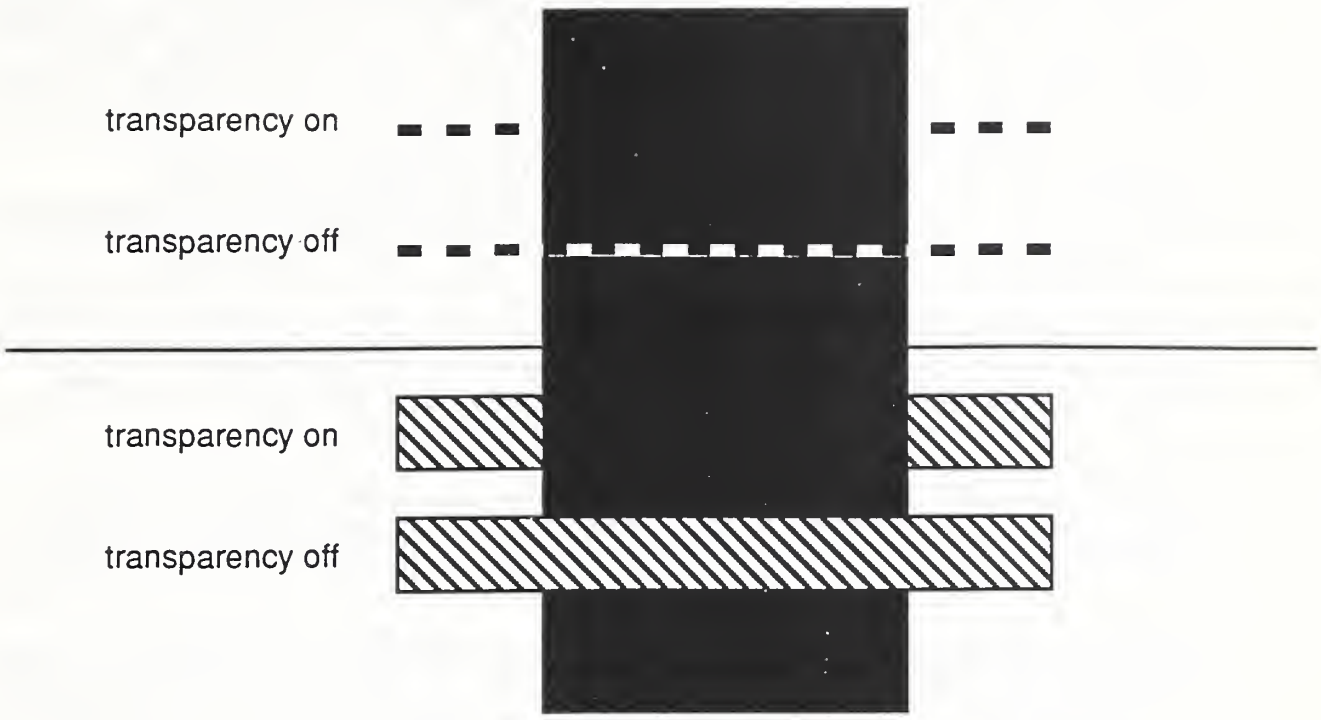


Figure 19. AUXILIARY COLOUR and TRANSPARENCY tests

4.7.7 Minimal CGM (1 test case)

Case 85 (MINCGM01): This case will use a CGM file with no non-required metafile descriptor or picture descriptor elements. It will draw a single line. The purpose is to verify that default values are used when none are specified.

4.7.8 Coordinate system options (7 test cases)

The highest priority test cases checked for implementation of an inverted Y axis. These test cases will check other combinations of VDC extent. They will also verify that the full extent of VDC is correctly mapped to the full extent of the picture. Both integer and real VDC will be checked in each case since interpreters often use separate code in each case to compute internal scale factors. The same sort of simple output used in the inverted Y axis case (a line drawn from lower left to upper right) can be used to distinguish mistakes in these other cases.

Case 86 (VDCEXT02): This case will use integer VDC in normal orientation and will have a picture that draws a single, rectangle with edge width 1 just inside the VDC extent rectangle. The script will verify that it is visible and fills the entire picture area.

Case 87 (VDCEXT03): This case will use real VDC in normal orientation and will have a picture that draws a single, rectangle with edge width 1 just inside the VDC extent rectangle. The script will verify that it is visible and fills the entire picture area.

Case 88 (VDCEXT04): This case will use integer VDC with an x-axis inverted from the default and will draw a single width 1 line from the lower left to the upper right. The script will verify that it is visible has the correct orientation.

Case 89 (VDCEXT05): This case will use real VDC with an x-axis inverted from the default and will draw a single width 1 line from the lower left to the upper right. The script will verify that it is visible has the correct orientation.

Case 90 (VDCEXT06): This case will use integer VDC with an x-axis inverted from the default and a y-axis inverted from the default. The picture will have a single width 1 line from the lower left to the upper right. The script will verify that it is visible has the correct orientation.

Case 91 (VDCEXT07): This case will use real VDC with an x-axis inverted from the default and a y-axis inverted from the default. The picture will have a single width 1 line from the lower left to the upper right. The script will verify that it is visible has the correct orientation.

Case 92 (VDCEXT08): This case will use real VDC with a y-axis inverted from the default. The picture will have a single width 1 line from the lower left to the upper right. The script will verify that it is visible and that it has the correct orientation.

4.7.9 MDR testing, including multiple MDR elements (2 test cases)

A minimal MDR was tested in highest priority cases. This test is expanded on to determine not only that MDR elements are allowed, but that their values are correctly processed. Also the ability to correctly process more than one MDR element will be verified

Case 93 (MDRTST02): This case will include a MDR element that includes most picture descriptor, control and attribute elements. Attributes will be set to a non-default value. The picture will contain no attribute elements and at least one primitive from

each class (line, marker, filled area, cell array) and verify that the values set in the MDR element are used for widths, styles, and colors.

Case 94 (MDRTST03): This case will be identical to MDRTST02, only the MDR data will be split into multiple MDR elements.

4.7.10 CGM with no pictures (1 test case)

Case 95 (NOPICT01): This case will contain a CGM with no picture. The CGM interpreter product should handle it gracefully and should not produce a blank picture.

4.7.11 CGM with an empty picture (3 test cases)

This test case will contain a CGM with a single picture that contains no graphical primitive elements. The CGM interpreter product should handle it gracefully, producing an empty picture where appropriate.

Case 96 (EMTPCT01): This case will have only BEGIN PICTURE and END PICTURE with no BEGIN PICTURE BODY. No picture should be produced.

Case 97 (EMTPCT02): This case will have BEGIN PICTURE, END PICTURE, and BEGIN PICTURE BODY. It will include control and attribute elements but no graphical primitives. A blank picture should be produced.

Case 98 (EMTPCT03): This case will have BEGIN PICTURE, END PICTURE, and BEGIN PICTURE BODY, but no other elements within the picture. A blank picture should be produced.

4.7.12 Default values (1 test case)

Case 99 (DEFVAL01): This case will be a minimal CGM with no non-required MFD or PD elements and no attribute elements. The picture will contain at least one primitive from each class (line, marker, filled area, cell array) and verify that correct default values are used for widths, styles, and colors.

4.7.13 Graphical attribute tests (1 test case only for now)

To round out the "top 100" test cases, case 100 begins the graphical primitive tests.

Case 100 (POLTLN02): This case will verify that the five line types are implemented correctly.

5 Impact of Versions and Levels

This section briefly discusses the impact on this testing strategy of:

- versions of the CGM standard;
- color levels in 28003A; and
- combinations and interactions of CGM elements.

5.1 Versions

Testing later versions of the CGM standard means adding additional test cases to cover the new elements. This is not yet necessary since no commercial products support features beyond those in Version 1 (as of the published date of this report). So far as is known, no previously developed tests will need to be changed to test CGM interpreter products that support later versions of the CGM standard or MIL-D-28003A.

Adding test cases for the new features in CGM:1992 and MIL-D-28003A will be an iterative process since no products yet support them. Tests can be developed and tested incrementally as features of Version 2 and Version 3 of CGM:1992 become implemented in commercial products.

5.2 Color levels in 28003A

Since many of the capabilities can be tested using black and white only, the effects of testing color levels in MIL-D-28003A is minimized. Additional tests which address full color and gray scale must be developed. The major impact will be in organizing the test cases for the level of color supported and ensuring that testing coverage at each level is complete.

5.3 Combinations and interactions of elements

The testing strategy developed here will account for the most obvious effects of the interactions of CGM elements. Experience has shown that the most common error in this category is that some implementations misapply attributes. For example, a product might use the edge color as both the line color and the edge color.

A second type of error that might be considered to be in this category involves the effect of picture descriptor and metafile descriptor information, such as integer precision, on the interpretation of graphical elements later in the file. The test strategy that NIST/CSL has developed explicitly tests for this type of interaction.

6 Test Case Development Plan - Version 1

This section summarizes information about the number of test cases remaining to be developed for CGM Version 1 interpreter testing. It also estimates the time required to develop the remaining tests and to establish a test service based on them.

6.1 Graphical primitive tests

The general strategy of these tests is to thoroughly and exhaustively test the geometric integrity of the primitive. The goal is to establish that the "right" locations are selected to be part of the primitive and that all geometric distances, such as widths of lines and edges, are correct. Here is an expanded description of the tests to be done and an estimate of the test cases required.

Description	Test cases
lines	
check that the line drawn is centered on the geometric locus	1
test disjoint polyline	1
check the geometric integrity of line width	1
markers	
check that all markers are centered on the specification point	1
circles and ellipses	
test geometric integrity, i.e. is the correct locus of points used	4
check edge location and edge width	2
arcs	
test geometric integrity, i.e. is the correct locus of points used	2
test the effect of more complex VDC spaces—especially an inverted y-axis—on arc sense	4
closed arcs	
test geometric integrity, i.e. is the correct locus of points used	4
test the effect of more complex VDC spaces—especially an inverted y-axis—on arc sense	4

filled areas (POLYGON, POLYGON SET, RECTANGLE)	
test geometric integrity, i.e. is the correct locus of points used	3
do more thorough test of POLYGON SET	2
do more thorough test of RECTANGLE	1
test geometric integrity, especially the relationship of edge location and edge width	3
test implicit and explicit polygon closure (i.e. check cases where the first and last point are equal and where they are not)	2
text	
test append text, including append text with a single null character	2
test the effect of more complex VDC spaces—especially an inverted y-axis—on character up direction and text path	4
test geometric integrity of TEXT ALIGNMENT	10
test geometric integrity of CHARACTER SPACING	1
test geometric integrity of CHARACTER HEIGHT	2
cell array	
test “path” of the cells and their ”line progression "	3
do a more thorough test of CELL ARRAY	3
Total:	**60**

6.2 Graphical attribute tests

The emphasis in these tests is checking “non-geometric” attributes not yet tested. Here is an expanded description of the tests to be done and an estimate of the test cases required.

Description	**Test cases**
lines	
test LINE COLOUR	1
edges	
test all 5 edge types	2
test EDGE COLOUR	2
markers	
test all 5 marker types	1
test MARKER COLOUR	1
text	
test attribute changes in restricted strings	2
test TEXT COLOUR	1

filled areas	
test that hatches 1-6 are supported	1
test that 8 distinct patterns are supported	1
test FILL REFERENCE POINT	1
test PATTERN TABLE options, including dimensions and local color precision	6
test PATTERN SIZE	2
misapplied attributes	
several test files that intermix primitives with attribute changes to detect e.g. line attributes mistakenly applied to edges	5
Total	**27**

6.3 Summary for CGM Version 1 Interpreter Testing

Initial estimates show that 87 additional test cases are required for Version 1 CGM:1992 and MIL-D-28003 interpreter testing. Allowing for a few additional tests whose need might be discovered during detailed design, an estimate of 100 additional test cases is reasonable. Following this effort it will be necessary to beta test the interpreter testing service based on the suite of 200 tests.

In addition, test tools and the test suite for MIL-D-28003A Version 1 must be upgraded. Finally, the test suite will need to be upgraded to Versions 2 and 3 of CGM:1992 as products become available.