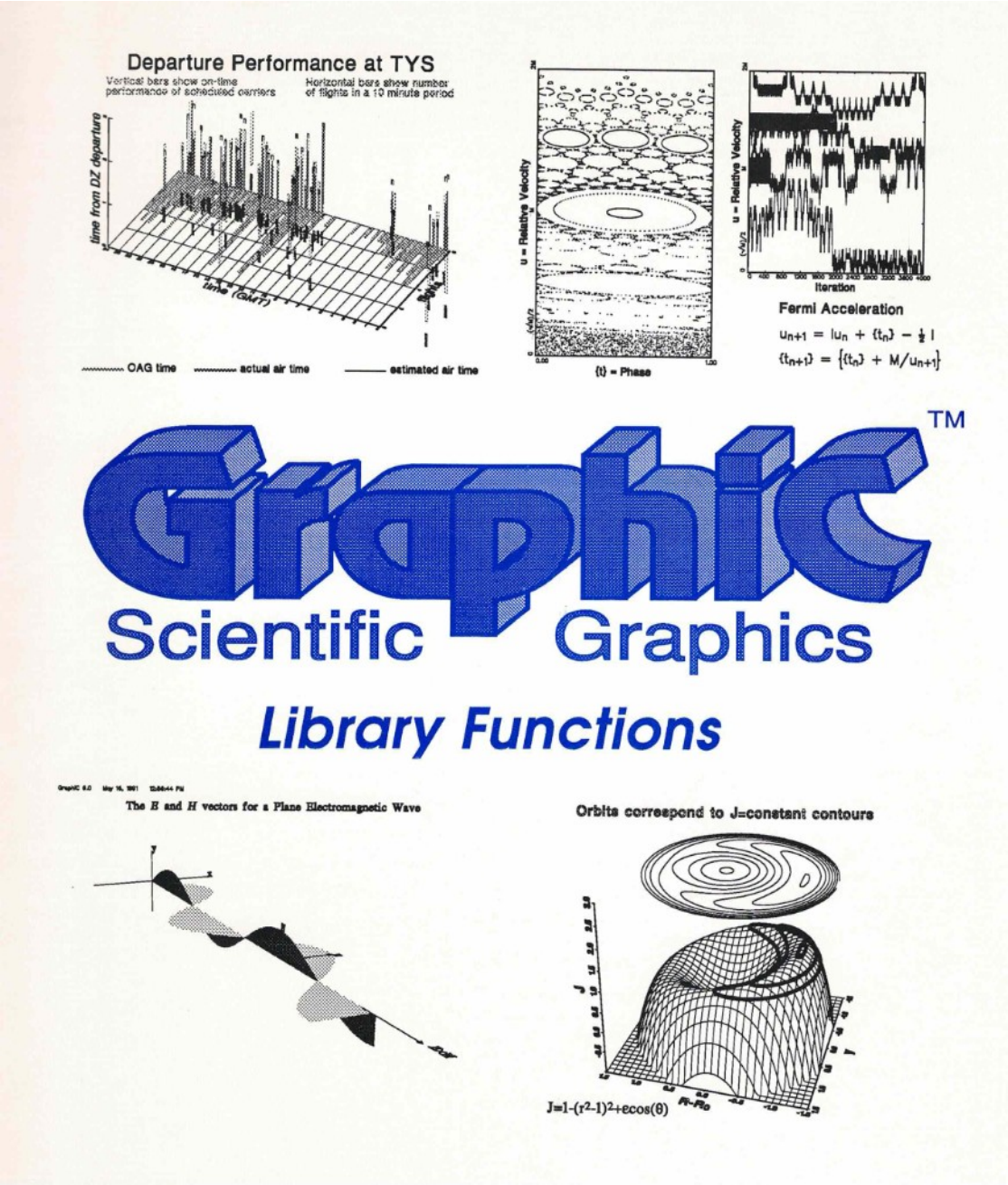
GraphiC 2026 — The Professional Scientific and Technical Graphics Library

A Modern 64-Bit Multi-Language Vector Graphics Engine



Original 1993 GraphiC Cover

In Memoriam to my collaborators: George Kelley and Steve Kelley

Thanks to David Dillow, who implemented TrueType and Postscript font support

GraphiC™ Software License Agreement

jamesrome.com
Oak Ridge, TN 37830
Website: https://jamesrome.com • Email: support@jamesrome.com

PLEASE READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING THIS SOFTWARE. USE OF THE SOFTWARE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS.

jamesrome.com provides this software and licenses its use worldwide. You assume responsibility for the selection of the software to achieve your intended results and for the installation, use, and results obtained from the software.

Software License Grant

You may: 1. **System & Machine Installation**: Install and execute the software on any computer, workstation, server, or cloud node in your possession. 2. **Backups & Reproductions**: Copy the software into any machine-readable or printed form for backup, research, educational, or modification purposes in support of your use. 3. **Modification & Integration**: Modify the software or merge it into another program for your personal, scientific, institutional, or academic use. Any portion of this software merged into another program will continue to be subject to the terms and conditions of this agreement. 4. **License Transfer**: Transfer the software and license to another party if the other party agrees to accept the terms and conditions of this agreement. If you transfer the software, you must at the same time either transfer all copies (printed and digital) to the same party or destroy any copies not transferred; this includes all modifications and portions merged into other programs.

Source Code & Redistribution

THE GRAPHIC SOURCE CODE IS PROVIDED FOR SCIENTIFIC, RESEARCH, EDUCATIONAL, AND PROFESSIONAL ENGINEERING USE. REDISTRIBUTION OF GRAPHIC SOURCE CODE, COMPILED OBJECT MODULES, OR DERIVATIVE RUNTIME ENGINES WITHIN COMMERCIAL APPLICATION PACKAGES FOR CONVEYANCE TO THIRD PARTIES REQUIRES AN EXTENDED LICENSING AGREEMENT WITH JAMESROME.COM.

You must reproduce and include the original copyright notices on any copy, modification, or portion merged into another program. You may not use, copy, modify, or transfer the program, in whole or in part, except as expressly provided for in this license.

Term & Governing Law

This license is effective until terminated. You may terminate it at any time by destroying the software together with all copies, modifications, and merged portions. It will also

terminate if you fail to comply with any term of this agreement.
This Agreement is governed by the laws of the State of Tennessee, United States of America.

Limited Warranty & Disclaimer

jamesrome.com provides the GraphiC library source code, modern 64-bit drivers, language packages, and technical documentation substantially in accordance with this reference manual. Because GraphiC is a foundational suite of scientific programming tools, rendering precision and system behavior depend to a significant extent upon the host compiler, operating platform, and skills of the user.

jamesrome.com shall not in any case be liable for special, incidental, consequential, indirect, or similar damages arising from the use or inability to use this software, even if advised of the possibility of such damages. This includes, without limitation, loss of profits or revenue, loss of computer time, loss of data, costs of recreating lost data, hardware damage, or claims by third parties.

Limitation of Remedies

If within 30 days of acquisition the GraphiC tools do not perform substantially in accordance with the manual, remedies shall be limited to the provision of corrected source code, updated documentation, or a refund of any license fees paid directly to jamesrome.com for the software. In no case shall liability exceed the amount of license fees paid for the right to use the software.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT REPRESENTS THE COMPLETE AND EXCLUSIVE AGREEMENT SUPERSEDING ANY PRIOR PROPOSAL OR ORAL/WRITTEN COMMUNICATION.

Table of Contents

1. Executive Summary & Historical Background

The IBM PC was released in 1981. At the time, George Kelley and I (James Rome) were plasma physicists working in the Thermonuclear Division at Oak Ridge National Laboratory. We realized that generating graphical plots would give us critical insight into plasma equilibria and physical behavior, but the original IBM PC had very low-resolution graphics (CGA 320×200). To overcome this, we each purchased a Corona PC, which offered an amazing 640×480 pixel display.

However, we soon discovered that there was no software available to produce publication-grade scientific plots on microcomputers. We had been using the DISSPLA™ Fortran package on mainframes and were heavily influenced by its methodology. In both DISSPLA and GraphiC, plotting is a structured, procedural state machine:

1. **Initialize Session**: `bgnplot(device, mode, file)`
2. **Page Dimensions & Units**: `page(width, height)`
3. **Start Page Canvas**: `startplot(bgcolor)`
4. **Set Plot Area & Origin**: `area2d(xlen, ylen)` and `physor(x, y)`
5. **Calibrate Axes & Labels**: `box()`, `heading(title)`, `xname(x)`, `yname(y)`, `graf(...)`
6. **Execute Plot Routines**: Calling the authentic GraphiC routines across all languages (`conmat`, `curve`, `surmat`, `smdraw`, `polgrid`, `xlog`, etc.) without obscuring them behind generic OOP abstractions.
7. **Close Frame & Subsystem**: Always conclude with `endplot()` followed by `stopplot()`.

We insisted on vector graphics—where everything is a discrete point, stroke, or filled polygon—so that we could zoom in to inspect fine details on our computer screens and reproduce plots cleanly across pen plotters and laser printers.

We decided that the C language was best suited to run efficiently on a personal computer, but we had to learn it from scratch! We also studied 8086 assembly language and hand-coded critical inner loops in assembly to maximize rendering speed, since numerical math co-processors (like the 8087) were not yet available. It took us several years to perfect our lean, mean, fast C graphics engine. We subsequently founded **Scientific Endeavors Corporation** to market GraphiC worldwide to scientists and other technical workers.

After George passed away, his son Steve Kelley took his place in the company. Steve cleaned up numerous bugs, expanded features, and managed dedicated customer support. When Steve suffered an untimely death, we terminated Scientific Endeavors. And there the codebase sat, untouched, for 30 years…

The 2026 Renaissance: Enter AI and Google Antigravity

In 2026, working in close collaboration with me, Google's **Antigravity** autonomous AI coding agent resurrected the original GraphiC C archives.

Over the course of this modernization: Archaic 16-bit DOS segmented pointers, expanded memory (EMS) page banking, and disk overlays were eliminated in favor of a clean, flat 64-bit POSIX/C99 architecture running natively across macOS (Apple Silicon ARM64 & Intel), Linux, and Windows. Obsolete pen plotter support and the Tektronix storage file format were replaced by a modern W3C Scalable Vector Graphics (SVG) driver with an interactive browser-based viewer and live monitor streaming. And, Complete multi-language parity was established—delivering first-class bindings for **Modern Fortran (2003/2018/legacy F77)**, **Python 3**, and **Java 22 (Foreign Function & Memory API)**.

Most importantly, the core mathematical soul of GraphiC remains unaltered: the exact procedural state machine, analytical 3D hidden-surface horizon algorithms, Hershey stroke vector fonts (and now Postscript and TrueType), and precision coordinate transforms developed four decades ago are now preserved and running at native 64-bit speed for the next generation of scientific research.

---

2. Architectural Overview

GraphiC modernizes the classic UNIX philosophy: a lean, fast, rock-solid native C core paired with clean, zero-overhead bindings for modern dynamic and managed languages.

2.1 Native C99 / POSIX Core Engine (`Source/`, `Include/`)

- Implemented in clean, portable C99 with standard POSIX headers.
- Completely free of external third-party dependencies; links strictly with the standard C runtime (`libc`) and mathematics library (`libm`).
- Builds simultaneously as a shared library (`lib/libgraphic.dylib` on macOS, `libgraphic.so` on Linux, `graphic.dll` on Windows) and static archive (`lib/libgraphic.a`).
- Manages 2D/3D transformations, viewport clipping, polygon filling, contour elevation calculation, and drawing pipelines.

2.2 Modern Vector SVG Driver & Interactive Web Viewer

- Replaces legacy Tektronix 4014 / 4105 vector formats with clean, standalone W3C Scalable Vector Graphics (SVG).
- Features resolution-independent vector lines, crisp circle/arc primitives, inline `<defs>` vector hatch patterns, and automatic `viewBox` scaling.
- Generates fully responsive output that renders natively in modern web browsers, Microsoft Word, LaTeX/PDF documents, and publication vector editors (Inkscape, Adobe Illustrator).

2.3 Python 3 Language Package (`python/graphic/`)

- Provides zero-copy foreign function interface via standard library `ctypes`.
- Offers pythonic ergonomics with context managers:

```
from graphic import Figure

with Figure(8.0, 6.0, background="white", output_path="output.svg") as
fig:
    fig.title("Damped Oscillator")
    fig.setup_axes(0.0, 1.0, 10.0, -1.0, 0.5, 1.0)
    fig.box()
    fig.plot(x, y, color="blue")
```

- Transparently accepts Python lists, tuples, or NumPy 1D and 2D arrays.

2.4 Java 22 Foreign Function & Memory (FFM) Package (`com.sciend.graphic`)

- **Zero-Overhead Panama FFM Architecture**: Built upon the modern JDK 22 Foreign Function & Memory API (`java.lang.foreign`). Directly downcalls native C symbols via `Linker.nativeLinker()` and off-heap `Arena` memory segments, eliminating legacy JNI boilerplate, C glue code, and runtime compilation wrappers.

- **Authentic GraphiC Procedural State Machine**: Provides direct static access via `com.sciend.graphic.Graphic` using authentic GraphiC routine names:

Java

```
import com.sciend.graphic.Graphic;
import static com.sciend.graphic.Graphic.*;

// 1. Initialize session & page canvas
bgnplot(0, 'S', "oscillator.svg");
page(8.0f, 6.0f);
startplot(BRT_WHITE);

// 2. Define plot window & origin
area2d(6.5f, 4.5f);
physor(0.75f, 0.75f);

// 3. Calibrate axes, labels & borders
box();
heading("Damped Oscillator");
xname("Time (seconds)");
yname("Amplitude (volts)");
graf("%3.1f", 0.0f, 1.0f, 10.0f, "%3.1f", -1.0f, 0.5f, 1.0f);
// 4. Plot vector curves & conclude
color(BLUE);
curve(x, y, x.length, 0);
endplot();   // or stopplot()
```

- **Deterministic Resource Management (`AutoCloseable`)**: For modern Java workflows, `Figure` implements `AutoCloseable` with automatic cleanup while maintaining authentic GraphiC routine names:

Java

```
import com.sciend.graphic.Figure;
import static com.sciend.graphic.Color.*;

try (Figure fig = new Figure(8.0f, 6.0f, WHITE, "oscillator.svg")) {
    fig.heading("Damped Oscillator");
    fig.xname("Time (seconds)");
    fig.yname("Amplitude (volts)");
    fig.graf("%3.1f", 0.0f, 1.0f, 10.0f, "%3.1f", -1.0f, 0.5f, 1.0f);
    fig.box();
    fig.color(BLUE);
    fig.curve(x, y);
}
```

2.5 Modern & Legacy Fortran Packages (`fortran/src/` and `Source/gpcfor_compat.c`)

Fortran has been an essential scientific computing language throughout GraphiC's history (originally supported on 32-bit Windows via `gpcfor.dll`). The modernized library provides two distinct Fortran interop layers:

1. **Modern Fortran 2003/2008/2018/2023 (`iso_c_binding`)**:
    - `fortran/src/graphic_iso.f90`: Type-safe standard C interoperability interface declarations mapping C entry points directly into Fortran with zero runtime glue overhead.

- `fortran/src/graphic.f90`: High-level idiomatic Fortran module providing constants (`COLOR_BLUE`, `STYLE_DASHED`), subroutines (`gpc_begin`, `gpc_graf`, `gpc_curve`, `gpc_surface`, `gpc_contour`), and automatic Fortran string null-termination (`trim(str) // c_null_char`).

2. **Legacy Fortran 77 ABI Compatibility Shim (`Source/gpcfor_compat.c`)**:

   Re-implements the classic `gpcfor.dll` pass-by-reference symbols in both uppercase (`CALL BGNPLOT(...)`, `CALL GRAF(...)`, `CALL CURVE(...)`) and lowercase trailing underscore (`bgnplot_`, `graf_`, `curve_`).

   Automatically handles fixed-length Fortran character array padding and hidden string lengths, allowing legacy scientific simulation and analysis codes to link without modification.

---

3. Memory Management: Legacy DOS vs Modern 64-Bit Systems

Historical 16-Bit Memory Architecture

In the original 1980s DOS implementation, IBM PC software operated under the infamous **640 KB conventional memory barrier** with 16-bit segmented real-mode pointers (`segment:offset`). Generating high-resolution 3D surface meshes (e.g., 100 × 100 floating-point arrays) or multi-thousand point curves easily exceeded available physical RAM.

To survive, GraphiC incorporated a custom virtual memory subsystem: - `getbuf()` and `freebuf()`: Custom dynamic allocators that partitioned small memory heaps. - Disk Paging & Temporary Spill Files: Array buffers were swapped to temporary scratch files on floppy disks or slow early hard drives. - Expanded / Extended Memory (EMS / XMS): Routines to bank-switch 16 KB pages into DOS memory windows. - Byte-Offset Seek Files (`.OFF`): Auxilliary index files written alongside `.TKF` plot files so that multi-page plots could be randomly accessed by seeking to byte offsets without parsing the entire file.

Modern 64-Bit Architecture

In modern 64-bit computing environments:

1. **Flat 64-bit Virtual Address Space**: Operating systems provide 2^{48} to 2^{64} bytes of addressable memory per process. Datasets of millions of points fit effortlessly in active memory without paging overhead.

2. **Virtual Memory & Hardware MMU**: The kernel and hardware Memory Management Unit (MMU) automatically handle page faults, demand paging, and translation lookaside buffer (TLB) caching orders of magnitude faster than any disk-swapping algorithm.

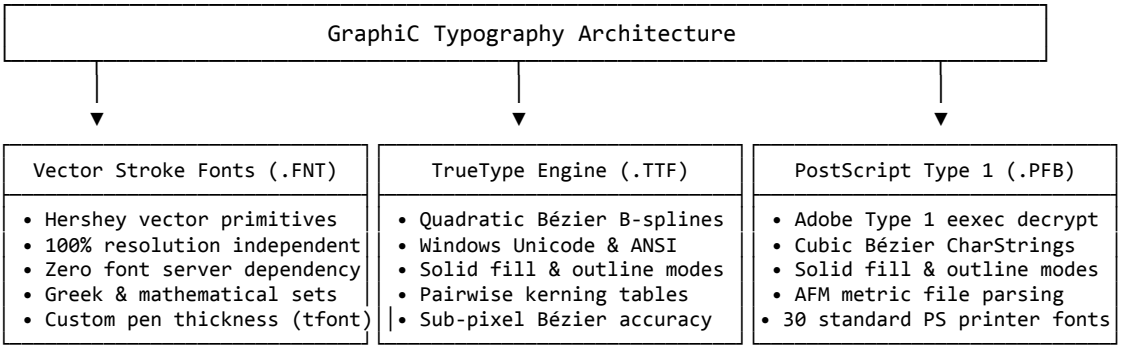
3. **Elimination of Obsolete `.OFF` Files**: Modern SVG files are self-contained XML documents. The legacy `.OFF` files served only DOS-era `PLAY.EXE` tools and have been permanently disabled, eliminating disk I/O overhead and directory clutter.

4. **Standard Heap Allocation**: The modern library uses standard C `malloc()`, `calloc()`, and `free()` for dynamic temporary buffers, ensuring thread safety, memory sanitizer (`ASan`) validation, and zero leaks.

---

4. Typography, Fonts, and Text Systems

GraphiC incorporates a comprehensive, multi-engine typographic subsystem engineered specifically for scientific plotting and technical publication. Rather than forcing all text through a single font model, GraphiC natively integrates three distinct font architectures—Hershey vector stroke fonts, standard TrueType (`.ttf`) outline fonts, and Adobe PostScript Type 1 (`.pfb` / `.pfa`) fonts. All three engines can be freely intermixed within the same plot, or switched dynamically in mid-string using in-line escape characters.



4.1 The Three Native Typography Engines

1. The Hershey Vector Stroke Font Engine (`.fnt`)

Originally compiled by Dr. Allen V. Hershey at the U.S. Naval Weapons Laboratory, Hershey fonts represent characters as connected vector line strokes:

- **Zero Dependencies**: Because characters are drawn directly with line segments, they require no operating system font managers, rasterizers, or display servers.
- **Arbitrary Scaling & Rotation**: Can be scaled to any cap height or rotated to arbitrary angles without raster blur, anti-aliasing artifacts, or loss of clarity.
- **Scientific Symbol Suites**: Complete coverage of uppercase and lowercase Greek alphabets, mathematical operators ($\int, \partial, \sum, \sqrt{\square}, \infty, \approx$), astronomical symbols, and accented characters.
- **Included Typefaces**: `simplex.fnt` (clean technical sans-serif), `complex.fnt` (double-stroke Roman serif), `duplex.fnt` (bold sans-serif), `triplex.fnt` (heavy Roman), `script.fnt` (flowing cursive), `italics.fnt` (slanted serif), and `simgrma.fnt` (Greek mathematical).

2. The Native TrueType Engine (`.ttf`)

Developed for GraphiC Version 7.0 by David Dillow (`Source/Truetype.c`, `Source/Ttfntld.c`, `Source/Ttfntplt.c`) and modernized for 64-bit platforms:

- **Direct TrueType File Parsing**: Reads standard Microsoft/Apple `.ttf` font files directly from disk via `gfopen()`.
- **64-Bit Clean Binary SFNT Ingestion**: Portable big-endian binary parsing of TrueType tables, removing legacy 64KB table limits. Parses `cmap` (Format 0 and Format 4 Microsoft Unicode BMP), `glyf` (Bézier geometry), `head`, `hhea`, `hmtx` (advance metrics), `kern`, `loca`, and `maxp`.
- **Transparent UTF-8 Sequence Decoding**: Multi-byte UTF-8 sequences in strings (Greek letters, math symbols, degree signs) are automatically decoded on the fly into Unicode code points and resolved through the `cmap` Format 4 table.
- **Quadratic B-Splines (`qspline`)**: Evaluates second-order Bézier curves into smooth polygonal contours with configurable tessellation tolerance (`SetTTMaxError`).
- **Solid Filling & Outlines with Even-Odd Rule**: Supports hollow vector outlines (`fillfont(0)`) and solid filled glyph interiors (`fillfont(1)`). In vector output (SVG, PDF), filled glyphs apply `fill-rule="evenodd"` so internal counters ('O', '0', 'B', '8', 'R', 'e', 'a') remain hollow.
- **Pairwise Kerning**: TrueType `kern` tables are automatically parsed to produce proportional, publication-quality letter spacing (`SetTTKerning(1, 0)`).

3. The PostScript Type 1 Engine (`.pfb` / `.pfa`) & GrafText

Also engineered by David Dillow (`Source/Psfont.c`):

- **Direct Adobe Type 1 Ingestion**: Reads binary (`.pfb`) and ASCII (`.pfa`) PostScript font files.
- **In-Memory `eexec` Decryption**: Automatically decrypts the encrypted private dictionary and character outlines using the standard Adobe Type 1 deciphering key (`55665`).
- **Cubic Bézier Curve Parsing**: Interprets PostScript `CharStrings` bytecodes and subroutines (`Subrs`), converting third-order Bézier splines into polygon outlines or solid filled glyphs.
- **Adobe Font Metrics (`.afm`) Integration**: Parses corresponding `.afm` metric files for character advance widths and pairwise kerning adjustments.

- **GrafText 30 Standard Typefaces**: For PostScript (`.ps`) and Encapsulated PostScript (`.eps`) printing, `GTfont(number)` selects from 30 standard printer-resident Adobe PostScript fonts.

4.2 Font Selection and Registration: `font()`

The public `font()` routine loads and registers up to `FNTNUM` (default 10) fonts in memory simultaneously:

```
void font(int nfont, ...);
```

Calling Syntax

The first parameter `nfont` specifies the total number of fonts to be loaded. It is followed by variable argument pairs: `font(nfont, font_path_1, escape_char_1, font_path_2, escape_char_2, ...)`: - `font_path` (`char *`): Path to the font file. The file extension automatically routes the font to the correct engine: - `.TTF` or `.ttf` → TrueType outline engine (`TRUETYPE`). - `.PFB` or `.pfb` → PostScript Type 1 filled engine (`PSFILLED`). - `.FNT` or `.fnt` → Hershey stroke vector engine (`REGULARFONT`). - `escape_char` (`char` or `int`): The character used as an in-string trigger to activate this font.

Practical Examples

Loading Mixed TrueType, PostScript, and Hershey Fonts:

```
/* Load 3 fonts: TrueType Times, PostScript Helvetica, and Hershey Greek */
font(3,
     "times.ttf",    '\\',    /* '\' triggers Times TrueType */
     "helv.pfb",     '|',     /* '|' triggers Helvetica PostScript */
     "simgrma.fnt",  '/'      /* '/' triggers Greek mathematical font */
);
fillfont(1);                  /* Enable solid fill for TrueType and
PostScript */
```

Reloading and Freeing Fonts: - Calling `font(-1)` reloads the previously registered font set without needing to repeat the arguments. - Calling `freememq()` unloads all loaded fonts and frees their heap buffers.

4.3 In-Line Multi-Font Switching & Mathematical Notation

Once fonts are registered with `font()`, they can be switched dynamically anywhere inside titles, axis labels, or annotations simply by including the designated escape character:

```
heading("Magnetic Field \\B(r) |[Tesla] /m/b");
```

- The text starts in the primary active font (`times.ttf`).
- `\\` switches to Font 0 (`times.ttf`).
- `|` switches to Font 1 (`helv.pfb`).
- `/` switches to Font 2 (`simgrma.fnt`), printing μ_B in Greek notation.

- Doubling the escape character (e.g., `//` or `\\\\`) prints the literal character without triggering a font change.

Subscripts and Superscripts

GraphiC features automatic baseline displacement and scaling for scientific scripts: - `^` enters superscript mode (configurable with `supsym(char)`). - `_` enters subscript mode (configurable with `subsym(char)`). - `scrht(factor)`: Adjusts the character height scale factor (default 0.7 × cap height). - `scrshft(factor)`: Adjusts the baseline shift percentage.

Example:

```
xname("Density n_e (10^1^9 m^-^3)");
yname("Electron Temperature T_e (keV)");
```

4.4 Typography Controls, Styling, and Effects

Function	Signature	Description
fillfont	void fillfont(int fill)	0 = hollow vector outlines; 1 = solid filled glyphs (default for TrueType & PostScript).
fontfill	void fontfill(int fill, int pat, int bdr)	Fills glyph interiors with pattern index pat and outlines characters in color bdr.
tifont	void tifont(float wid)	Sets stroke pen thickness in physical inches (e.g., 0.02f).
tfont	void tfont(int wid)	Sets stroke pen thickness in device pixels (0 = single pixel, 1–7 = bold).
widen	void widen(float factor)	Scales horizontal character aspect ratio (1.0 = normal, 1.4 = wide, 0.8 = condensed).
skew	void skew(float slant)	Slants the vertical stroke axis for oblique or simulated italic styling.
circtxt	void circtxt(float x, float y, float r, char *s, float h, float a)	Renders string s curved along a circular arc of radius r centered at (x, y).
SetTTKerning	void SetTTKerning(char onoff, char str)	1 = enables TrueType pairwise kerning table evaluation; 0 = disables kerning.
SetTTFull	void SetTTFull(char onoff, char str)	1 = enables full 256-character set (Windows ANSI/Latin-1); 0 =

Function	Signature	Description
		standard 128 ASCII.
SetTTMaxError	void SetTTMaxError(float err, char str)	Sets maximum Bézier curve tessellation error tolerance (lower values = smoother curves).
GTfont	void GTfont(int font_number)	Selects one of the 30 standard Adobe PostScript printer fonts for GrafText output.

4.5 Standard Adobe PostScript Printer Fonts (GrafText)

When exporting plots to Encapsulated PostScript (`.eps`) or routing to PostScript printer drivers, `GTfont(number)` provides direct access to the 30 standard PostScript printer fonts:

Index	Font Name	Index	Font Name	Index	Font Name
0	Times-Roman	**10**	Courier-Oblique	**20**	NewCenturySchlbk-Roman
1	Times-Bold	**11**	Courier-BoldOblique	**21**	NewCenturySchlbk-Bold
2	Times-Italic	**12**	Palatino-Roman	**22**	NewCenturySchlbk-Italic
3	Times-BoldItalic	**13**	Palatino-Bold	**23**	NewCenturySchlbk-BoldItalic
4	Helvetica	**14**	Palatino-Italic	**24**	Helvetica-Narrow-BoldOblique
5	Helvetica-Bold	**15**	Palatino-BoldItalic	**25**	Helvetica-Narrow
6	Helvetica-Oblique	**16**	Bookman-Demi	**26**	Helvetica-Narrow-Bold
7	Helvetica-BoldOblique	**17**	Bookman-DemiItalic	**27**	Helvetica-Narrow-Oblique
8	Courier	**18**	Bookman-Light	**28**	Symbol ($\alpha, \beta, \gamma, \infty, \int, \pm$)
9	Courier-Bold	**19**	Bookman-LightItalic	**29**	ZapfDingbats

4.6 Modern SVG Vector Typography (2026 Modernization)

In the modern 64-bit SVG driver, text is rendered with dual-mode precision: 1. **Geometric Vector Outlines & Filled Polygons**: When maximum archival fidelity and 100% device independence are required, TrueType, PostScript, and Hershey characters are decomposed into exact SVG `<path>` and `<polygon>` elements. The output is mathematically guaranteed

to render identically on every device, operating system, and vector graphics suite (Inkscape, Illustrator, Word) without requiring local font files. 2. **Semantic SVG `<text>` Elements**: When searchability, screen reader accessibility, and text selection in web browsers or PDF viewers are preferred, GraphiC emits semantic `<text>` nodes with standard CSS font family cascades (`font-family: "Times New Roman", Times, serif;`).

4.7 Native Unicode (UTF-8) and TrueType Typography

Modern 64-bit GraphiC introduces transparent Unicode (UTF-8) decoding alongside the fully modernized TrueType outline engine. Scientific users can now use standard UTF-8 string literals directly in C, C++, Python, Java, and Fortran code without relying on legacy character codes or awkward multi-font escape switches.

1. Transparent UTF-8 Multi-Byte Decoding

When a TrueType font (`.ttf`) is active, GraphiC's string measurement and rendering pipelines (`strsizq()` and `TrueTypeOut()` in `Source/Ttfntplt.c`) automatically decode UTF-8 multi-byte sequences on the fly into 16-bit and 32-bit Unicode scalar values:

- **2-Byte Sequences (0xC0–0xDF)**: Standard Greek characters (α, β, γ, δ, λ, μ, π, σ, ω), degree symbols (°C), plus-minus (±), and Latin accents.
- **3-Byte Sequences (0xE0–0xEF)**: Mathematical operators (≤, ≥, ≠, ≈, ≡, ⊂, ∈, ∞, ∇, ∂, √), arrows (→, ←), and geometric shapes.
- **4-Byte Sequences (0xF0–0xF7)**: Extended mathematical alphanumeric symbols and specialized notations.

The decoded Unicode value is queried directly against the font's OpenType/TrueType `cmap`table (Format 4 Unicode BMP), retrieving the exact vector glyph contour definitions without intermediate transliteration.

2. Character Counter Geometry & Even-Odd Hole Punching

TrueType glyphs represent character shapes using quadratic Bézier contour loops. Glyphs with enclosed "counters" or holes—such as **O**,**0**,**B**,**8**,**R**,**e**,**a**, %, and Greek letters like Δ and θ —contain an outer contour followed by one or more interior cutout contours.

In the modernized SVG vector driver, all filled TrueType glyphs are rendered with `fill-rule="evenodd"`. The standard even-odd fill rule ensures that the interior counters remain completely transparent and hollow, preventing glyph holes from being erroneously filled in by vector renderers, web browsers, or PDF converters.

3. TrueType Outline vs. Hershey Vector Fonts

GraphiC provides both font technologies so developers and researchers can choose the optimal approach for each technical application:

Feature	Hershey Vector Fonts (.fnt)	Modern TrueType Fonts (.ttf)
Architecture	Compact vector stroke tables	Quadratic Bézier B-spline outlines
Dependencies	**Zero (100% self-contained)**	Requires standard .ttf font files
Counter / Holes	Open vector line strokes	**Hollow via fill-rule="evenodd"**
Scaling / Zoom	Arbitrary linear scaling	Infinite Bézier curve resolution
Character Set	ASCII + Hershey Greek/Math plates	**Full Unicode BMP (Latin, Greek, Math)**
String Syntax	Escape characters (e.g. /a/b)	**Direct UTF-8 literals ("λ = 532 nm")**
Best Used For	Archival plots, headless servers, embedded	High-impact publications, journal figures

4. Scientific Example: Native UTF-8 Spectroscopy Plot

```
#include <graphic.h>

int main(void) {
float x[] = {0.0f, 1.0f, 2.0f, 3.0f};
float y[] = {25.0f, 30.0f, 45.0f, 60.0f};

bgnplot(1, 'g', "spectroscopy.svg");
startplot(WHITE);

/* Load system Arial font as TrueType Font 0 with trigger '\310' */
font(1, "/System/Library/Fonts/Supplemental/Arial.ttf", '\310');
fillfont(1); /* Enable solid glyph fill with even-odd counter hollowing */

page(8.0f, 6.0f);
area2d(6.5f, 4.5f);
color(BLUE);

/* Direct UTF-8 string literals: Greek symbols, math relations, degree signs */
heading("\310Laser Spectroscopy: λ = 532 nm, T = 25 °C, ΔV ≤ 5.0 V");
xname("\310Wavelength Offset Δλ (nm) [α = 0.05, β = 1.2, π ≈ 3.14]");
yname("\310Temperature T (°C) [±0.5 °C]");

graf("%-4.1f", 0.0f, 1.0f, 3.0f, "%-4.1f", 20.0f, 10.0f, 70.0f);
color(RED);
curve(x, y, 4, 0);
```

```
endplot();
stopplot();
return 0;
}
```

5. Directory Layout and Clean Workspace Policy

The repository is structured to maintain strict separation of concerns and a 100% clean root directory:

```
GraphiC/
├── c/
│   └── examples/          # 33 portable C example sources & generated .svg
plots
├── python/
│   ├── graphic/           # Python library package (bindings, figure,
constants)
│   └── examples/          # 19 Python example scripts & generated .svg plots
├── java/
│   ├── src/               # Java 22 library source code (com.sciend.graphic)
│   ├── bin/               # Compiled Java bytecode (.class files)
│   └── examples/          # 19 Java example classes & generated .svg plots
├── fortran/
│   ├── src/               # Modern Fortran modules (graphic_iso.f90,
graphic.f90)
│   ├── bin/               # Compiled Fortran modules and executables
│   └── examples/          # Fortran example sources & generated .svg plots
├── docs/
│   ├── images/            # High-resolution rendered PNG figures for manual
│   ├── MANUAL.md          # Comprehensive Markdown manual source
│   └── GraphiC_Modern_Manual.docx # Complete Microsoft Word manual
├── Include/               # C/C++ public and private header files
├── Source/                # Core C graphics engine source code
├── lib/                   # Compiled shared (.dylib) and static (.a)
libraries
└── bin/                   # Compiled C example executables
```

> **Clean Workspace Policy**: - No stray `.svg`, `.tkf`, `.html`, or `.off` files are ever permitted in the root repository directory. - All example executables and scripts must write their vector output files strictly inside their respective language directory (`c/examples/`, `python/examples/`, or `java/examples/`). - The obsolete `.OFF` file generation has been permanently disabled in the C driver.

6. Compilation, Quickstart & Live Monitor Display

Building C/C++ Libraries and Examples

```
cd <GraphiC_Dir>

# 1. Clean build artifacts
make clean

# 2. Build libgraphic.a, libgraphic.dylib, and play utility
make all

# 3. Compile all 33 portable C example executables into bin/
make examples

# 4. Run all C examples to generate SVGs in c/examples/
make run_c_examples
```

Running Python 3 Examples

```
export PYTHONPATH=<GraphiC_Dir>/python:$PYTHONPATH

# Run individual examples
python3 python/examples/d2test.py
python3 python/examples/d3test.py
python3 python/examples/cnttest.py
python3 python/examples/smtest.py
```

Compiling and Running Java 22 Examples

```
cd <GraphiC_Dir>/java

# Compile library and examples with Java 22 FFM preview enabled
javac --enable-preview --source 22 -d bin
src/main/java/com/sciend/graphic/*.java examples/*.java

# Run individual examples
java --enable-preview -cp bin examples.D2Test
java --enable-preview -cp bin examples.D3Test
java --enable-preview -cp bin examples.CntTest
java --enable-preview -cp bin examples.SmTest
```

Compiling and Running Fortran Examples

Fortran support requires `gfortran` (Homebrew GCC):

```
# Compile library, Fortran modules, and example executables
make fortran_examples
```

```
# Run all Fortran examples (generates SVGs in fortran/examples/)
make run_fortran_examples
```

Example modern Fortran usage:

```
program harmonic
  use graphic
  implicit none

  call gpc_begin(DRIVER_SVG, "fortran/examples/d2test.svg")
  call gpc_page(8.0, 6.0)
  call gpc_start(COLOR_WHITE)
  call gpc_area2d(5.5, 4.0)
  call gpc_physor(1.5, 1.2)
  call gpc_box()
  call gpc_title("Harmonic Oscillations", "Time (s)", "Amplitude (V)")
  call gpc_graf("%3.1f", 0.0, 1.0, 10.0, "%3.1f", -1.0, 0.5, 1.0)
  call gpc_color(COLOR_BLUE)
  call gpc_curve(x, y, 100, 0)
  call gpc_end()
  call gpc_stop()
end program harmonic
```

Real-Time Live Monitor Display & Early Run Abort

A major architectural advantage of GraphiC is its ability to draw graphics directly to the screen monitor *as data is generated* (`bgnplot(mode, ...)` with `mode = 0` for monitor only or `mode = 2` for dual monitor and file output).

Why Live Monitor Display Is Mission-Critical

In scientific computing, aerospace trajectory modeling, finite-element analysis, and laboratory data acquisition (DAQ), calculations often run for many minutes, hours, or days: 1. **Interactive Data Streaming**: Rather than running "blind" in batch mode and inspecting a static plot file hours later, the engineer can observe the physical state, orbital path, or voltage trace evolving point-by-point in real time. 2. **Early Error Detection & Run Cancellation**: If an integration step diverges, numerical instability emerges, boundary conditions are violated, or a physical sensor drops data, the anomaly is immediately visible on the monitor. The user can press `Ctrl+C` or click Stop to abort the run immediately, refine parameters, and restart—saving valuable time and computational resources. 3. **Dual-Channel Logging**: Mode `2` directs graphics to the interactive display while simultaneously logging the complete, publication-grade vector graphics file (`.svg` or `.tkf`) to disk for documentation and archival reporting.

Section 7. Complete GraphiC API Technical Reference

This reference documents the complete public API of the GraphiC library reconstructed from the original C source code implementation, parameter tables, and algorithmic commentary (`Source/*.c` and `Include/Proto.h`). Each routine is specified in the clean, high-contrast format of the original printed reference manual detailing its operational level, source mapping, coordinate systems, (new) multi-language signatures, parameter constraints, and algorithmic behavior.

Routine Level Classifications

As in the original printed GraphiC reference manual (Page L-01), routines are classified into functional operational levels:

Level 0 (Pre-Initialization): Global configuration routines invoked before `bgnplot()` (e.g., `rotate()`).

Level 1 (Basic / Primary): Foundational routines invoked immediately following session initialization (`bgnplot`, `startplot`, `page`, `area2d`, `graf`, `curve`, `surmat`, `conmat`, `smdraw`, `polgrid`, etc.).

Level 2 (Intermediate / Customization): Specialized styling, font attributes, custom tick spacing, grid controls, 3D viewing angles, legends, and statistical fitting.

Level 3 (Advanced / Low-Level & Interactive): Complex mathematical transformations, interactive crosshair digitizers, simultaneous subplot context switching, raw clipping windows, and coordinate conversion pipelines.

Initialization Routines

Covers global plotting subsystem initialization, screen monitor bindings, driver allocation, and mode configuration.

void bgnplot(char mode, char driver, char *tekfile) BPLT.C

Level 1

int mode	`0` = Monitor/Screen interactive display only; `1` = Vector file logging only (`.svg` / `.tkf`); `2` = Simultaneous live monitor display AND vector file logging (dual-channel).
char driver	Output driver / display target: `'S'` = SVG vector graphics (default modern); `'g'` = Live graphics monitor display; `'t'` = Text terminal; `'T'` = Tektronix stream.
char* tekfile	Output destination filename (e.g. `"plot.svg"`), or `"notek"` to suppress file output.

Initializes a graphics plotting session, allocates internal device state tables, and binds the output vector stream, display driver, and live screen monitor.

bgnplot() must be called prior to any other GraphiC routine (with the exception of static configuration routines like rotate()).

> [!IMPORTANT] > **Real-Time Simulation Monitoring & Early Run Abort**: > A vital design capability of GraphiC is rendering live graphics directly to the screen monitor while simultaneously streaming vector commands to a file. In intensive physics calculations, orbital trajectory integrations, aerospace simulations, and laboratory data acquisition, computations can run for hours. GraphiC draws each vector and curve incrementally as data arrives, enabling the user to immediately observe divergence, numerical instability, or sensor errors. If a run is going awry, the user can terminate the process early, saving hours of unnecessary compute time.

Multi-Language Signatures:

C	`void bgnplot(char mode, char driver, char *tekfile)`
Python 3	`graphic.bgnplot(mode: int, driver: str, tekfile: str) -> None`
Java 22	`GraphiC.bgnplot(int mode, char driver, String tekfile)`
Fortran	`call gpc_bgnplot(mode, driver, tekfile)`

void startplot(int bcolor) BPLT.C

Level 1

int bcolor Canvas background palette color index.

Begins a new plotting canvas page and initializes background fill.

Clears the active frame, resets clipping boundaries to full canvas dimensions, resets pen state to UP at origin, and sets background color. In multi-page sessions, each startplot() constitutes an independent page frame. Use WHITE or BRT_WHITE for no background color.

Multi-Language Signatures:

C	`void startplot(int bcolor)`
Python 3	`graphic.startplot(bcolor: int) -> None`
Java 22	`GraphiC.startplot(int bcolor)`
Fortran	`call gpc_startplot(bcolor)`

void NewFile(char *NewName) BPLT.C

Level 2

char* NewName Destination filename for subsequent plots.

Closes the current output file stream and redirects subsequent pages to NewName.

Allows generating a sequential series of separate vector graphics files from a single continuous calculation or simulation loop.

Multi-Language Signatures:

C	`void NewFile(char *NewName)`
Python 3	`graphic.NewFile(NewName: str) -> None`
Java 22	`GraphiC.NewFile(String NewName)`
Fortran	`call gpc_NewFile(NewName)`

Plot Termination Routines

Handles canvas page closure, vector buffer flushing, plot additions, and complete session termination.

void dateit(Uchar font) BPLT.C

Level 3

char font Font selector character; `'0'` turns date/time off.

Enables or disables automatic date and time stamping on the current plot page.

The timestamp is printed along the top edge of the active picture using the stroke font specified by the font selector character. Passing '0' disables timestamping (default).

or '0' to disable date and time output. Enclose in single quotes.

Multi-Language Signatures:

C	`void dateit(Uchar font)`
Python 3	`graphic.dateit(font: str) -> None`
Java 22	`GraphiC.dateit(byte font)`
Fortran	`call gpc_dateit(font)`

void plotadd(void) SVC.C

Level 3

Returns GraphiC to Level 1 while keeping the current page and screen active.

Used when multiple overlapping coordinate systems or independent curve overlays must be composited onto the same physical drawing region without re-erasing the screen or writing an end-of-page trailer.

See Also: `startplot, endplot`

Multi-Language Signatures:

C	`void plotadd(void)`
Python 3	`graphic.plotadd() -> None`
Java 22	`GraphiC.plotadd()`
Fortran	`call gpc_plotadd()`

int endplot(void) BPLT.C

Level 1

Completes drawing for the current page, flushes pending polygon and polyline buffers to the output driver, and closes active vector container elements.

In interactive environments, endplot() pauses for user inspection or keypress. When GPC_NONINTERACTIVE=1 or batch driver is active, it completes immediately.

@return [int] Status code (0 for success, non-zero for abort or error).

See Also: `startplot, stopplot`

Multi-Language Signatures:

C	`int endplot(void)`
Python 3	`graphic.endplot() -> int`
Java 22	`GraphiC.endplot() -> int`
Fortran	`status = gpc_endplot()`

void stopplot(void) BPLT.C

Level 1

Terminates the GraphiC session.

Flushes and closes the active file stream (e.g. final </svg> closing tag), frees dynamically allocated workspace memory (horizons, spline buffers, font tables), and resets terminal or display hardware to initial state.

See Also: `bgnplot, endplot`

Multi-Language Signatures:

C	`void stopplot(void)`
Python 3	`graphic.stopplot() -> None`
Java 22	`GraphiC.stopplot()`
Fortran	`call gpc_stopplot()`

Page and Axis Options

Defines physical sheet dimensions, active plotting viewports, curve cropping, axis offsets, borders, tick geometries, text heights, and legend positioning.

void acrop(char flag) SVC.C

Level 2

char flag 0 = crop curves at the edge of the plot region (default); 1 = turn off line cropping, extending vectors to the drawing page boundary.

The GraphiC default is to crop lines and curves at the edges of the plot region (i.e., the rectangular viewport defined by the call to area2d()). To turn off this feature, call acrop(1); this automatically moves the clipping boundary to the physical edge of the drawing page, allowing vectors and curves to span across the entire sheet. Line cropping is automatically reset to the default value (acrop(0)) after each call to curve drawing routines.

Multi-Language Signatures:

C	`void acrop(char flag);`
Python 3	`graphic.acrop(flag: int) -> None`
Java 22	`GraphiC.acrop(int flag)`
Fortran	`call gpc_acrop(flag)`

void area2d(float xinch, float yinch) BPLT.C

Level 1

float xinch Active plotting box width.

float yinch Active plotting box height.

Establishes the physical bounding dimensions of the active 2D plotting box.

The area2d dimensions define the length of the horizontal (X) and vertical (Y) axes. Data points outside this boundary are clipped unless clipping is overridden. Must be called prior to graf(), scales(), or box().

Multi-Language Signatures:

C	`void area2d(float xinch, float yinch)`
Python 3	`graphic.area2d(xinch: float, yinch: float) -> None`
Java 22	`GraphiC.area2d(float xinch, float yinch)`
Fortran	`call gpc_area2d(xinch, yinch)`

void physor(float xphys, float yphys) SVC.C

Level 1

float xphys Horizontal origin offset from canvas left edge.

float yphys Vertical origin offset from canvas bottom edge.

Establishes the physical offset of the lower-left corner of the plot area from the bottom-left corner of the canvas page in physical inches.

Allows leaving margins for axes numerical labels, axis titles, and page headings, or positioning multiple separate subplots across a single page.

Multi-Language Signatures:

C	`void physor(float xphys, float yphys)`
Python 3	`graphic.physor(xphys: float, yphys: float) -> None`
Java 22	`GraphiC.physor(float xphys, float yphys)`
Fortran	`call gpc_physor(xphys, yphys)`

void orel(float xorel, float yorel) SVC.C

Level 2

float xorel Horizontal relative displacement in inches.

float yorel Vertical relative displacement in inches.

Translates the current physical origin by relative increments dx, dy.

Useful for stepping through regular grid arrangements of subplots in multi-panel figures without recalculating absolute page coordinates.

Multi-Language Signatures:

C	`void orel(float xorel, float yorel)`
Python 3	`graphic.orel(xorel: float, yorel: float) -> None`
Java 22	`GraphiC.orel(float xorel, float yorel)`
Fortran	`call gpc_orel(xorel, yorel)`

void box(void) SVC.C

Level 1

Draws a rectangular border enclosing the active 2D plotting area defined by area2d().

Uses current line style, thickness, and color. Must be called after area2d() and physor().

See Also: `area2d, pbox`

Multi-Language Signatures:

C	`void box(void)`
Python 3	`graphic.box() -> None`
Java 22	`GraphiC.box()`
Fortran	`call gpc_box()`

void rotate(char isrot) BOTHPLT.C

Level 1

char isrot 0 for landscape, 1 for portrait.

Configures orientation of the physical drawing coordinate system.

0 = Upright / Landscape orientation. 1 = Rotated 90 degrees / Portrait orientation.

Must be called before bgnplot() or startplot().

Multi-Language Signatures:

C	`void rotate(char isrot)`
Python 3	`graphic.rotate(isrot: int) -> None`
Java 22	`GraphiC.rotate(char isrot)`
Fortran	`call gpc_rotate(isrot)`

void axesoff(int noaxes) SVC.C

Level 1

int noaxes Bitmask controlling axes visibility:

Selectively disables the rendering of axis lines and/or numeric labels.

Either scales() or graf() must still be called so that GraphiC can calculate data-to-pixel transformation matrices.

Bit 0 (0x01, AXESOFF): Turn all axes and labels off. Bit 1 (0x02, XAXIS): Suppress X-axis line. Bit 2 (0x04, YAXIS): Suppress Y-axis line. Bit 3 (0x08, XLAB): Suppress X-axis numerical labels. Bit 4 (0x10, YLAB): Suppress Y-axis numerical labels. Value 0 (AXESON): All axes and labels visible (default).

Multi-Language Signatures:

C	`void axesoff(int noaxes)`
Python 3	`graphic.axesoff(noaxes: int) -> None`
Java 22	`GraphiC.axesoff(int noaxes)`
Fortran	`call gpc_axesoff(noaxes)`

void frame(int frameon, int ftick) SVC.C

Level 1

int frameon 1 = draw bounding frame around axis area; 0 = off (default).

int ftick 1 = add tick marks along the entire perimeter frame; 0 = ticks on axes only.

Enables an enclosing rectangular frame around the 2D plot area.

Must be called before any axis-generating routines (graf or scales).

Multi-Language Signatures:

C	`void frame(int frameon, int ftick)`
Python 3	`graphic.frame(frameon: int, ftick: int) -> None`
Java 22	`GraphiC.frame(int frameon, int ftick)`
Fortran	`call gpc_frame(frameon, ftick)`

void cross(int iscross) SVC.C

Level 1

int iscross 1 = draw crossed axes passing through (0, 0); 0 = draw axes along left/bottom boundaries (default).

Selects between boxed perimeter axes and center-crossed axes.

Multi-Language Signatures:

C	`void cross(int iscross)`
Python 3	`graphic.cross(iscross: int) -> None`
Java 22	`GraphiC.cross(int iscross)`
Fortran	`call gpc_cross(iscross)`

void CrossAt(float x, float y) AXESDRAW.C

Level 1

float x X data coordinate of the vertical axis intersection.

float y Y data coordinate of the horizontal axis intersection.

Sets the intersection coordinates (x, y in data units) where axes intersect.

Rather than binding axes to the left and bottom perimeter of the plot box, CrossAt allows axes to cross at arbitrary values such as (0, 0) for Cartesian graphs. Must be called before graf().

Multi-Language Signatures:

C	`void CrossAt(float x, float y)`
Python 3	`graphic.CrossAt(x: float, y: float) -> None`
Java 22	`GraphiC.CrossAt(float x, float y)`
Fortran	`call gpc_CrossAt(x, y)`

void pgshift(float xshift, float yshift) BPLT.C

Level 1

float xshift Horizontal translation in inches (or cm).

float yshift Vertical translation in inches (or cm).

Shifts the position of the entire plot by xshift and yshift inches.

If rotate(1) was invoked, shifts are interpreted relative to the rotated screen coordinate frame.

Multi-Language Signatures:

C	`void pgshift(float xshift, float yshift)`
Python 3	`graphic.pgshift(xshift: float, yshift: float) -> None`
Java 22	`GraphiC.pgshift(float xshift, float yshift)`
Fortran	`call gpc_pgshift(xshift, yshift)`

void axnamht(float inch); void tickht(float height); void titlht(float inch); void numht(float inch); GRAPHIC.C

Level 1 & 2

float inch Desired size in physical inches (or cm if metricunits(1)).

Global dimensioning routines for axis typography and tick marks:

axnamht(inch): Sets height of axis title labels (xname, yname). Default 0.15 in. tickht(height): Sets physical height of tick marks. Default 0.08 in. titlht(inch): Sets height of plot title string (heading). Default 0.20 in. numht(inch): Sets height of numeric values along axes. Default 0.10 in.

Multi-Language Signatures:

C	`void axnamht(float inch); void tickht(float height); void titlht(float inch); void numht(float inch);`
Python 3	`graphic.axnamht(inch: float) -> None; graphic.tickht(height: float) -> None; graphic.titlht(inch: float) -> None; graphic.numht(inch: float) -> None`
Java 22	`GraphiC.axnamht(float inch); GraphiC.tickht(float height); GraphiC.titlht(float inch); GraphiC.numht(float inch);`
Fortran	`call gpc_axnamht(inch); call gpc_tickht(height); call gpc_titlht(inch); call gpc_numht(inch)`

void upright(char uplab) — SVC.C

Level 1

char / int uplab 1 = draw Y-axis numeric labels upright (horizontal);

Controls orientation of Y-axis numeric labels.

0 = rotate labels 90 degrees parallel to the Y-axis (default).

Multi-Language Signatures:

C	`void upright(char uplab)`
Python 3	`graphic.upright(uplab: int) -> None`
Java 22	`GraphiC.upright(byte uplab)`
Fortran	`call gpc_upright(uplab)`

void legpos(int entries, float xleg, float yleg, int border); void legend(int marker, char *string, int style, float strhgt, int scolor); — LEGEND.C

Level 2

int entries	Maximum number of entries in legend box.
int border	Border color (-1 for no border outline).
int marker	Symbol marker index (0 for line only).
char* string	Descriptive legend text.
int style	Line pattern style (0-8).
float strhgt	Character font height in inches.
int scolor	Color index.

Publication legend box builder.

legpos(entries, xleg, yleg, border): Establishes legend box position (xleg, yleg) in physical inches, allocates vertical spacing for entries items, and specifies border outline. legend(marker, string, style,

strhgt, scolor): Adds an annotated entry containing a sample line segment of style and color, centered marker symbol, and descriptive text label.

Multi-Language Signatures:

C	`void legpos(int entries, float xleg, float yleg, int border); void legend(int marker, char *string, int style, float strhgt, int scolor);`
Python 3	`graphic.legpos(entries, xleg, yleg, border); graphic.legend(marker, string, style, strhgt, scolor)`
Java 22	`GraphiC.legpos(int entries, float xleg, float yleg, int border); GraphiC.legend(...)`
Fortran	`call gpc_legpos(entries, xleg, yleg, border); call gpc_legend(...)`

void grid(int gridon); void fgrid(int fon, int ndiv); GRIDOPTS.C

Level 2

int gridon	1 = major grid on, 0 = major grid off.
int fon	Line pattern for fine grid (-1 for fine ticks only, 0-8 for styles).
int ndiv	Number of sub-divisions per major grid division.

Configures major and fine background grid lines across the 2D plotting box.

grid(gridon): If gridon=1, draws dotted grid lines aligned with major tick marks. fgrid(fon, ndiv): Subdivides each major interval into ndiv sub-grid lines using line style fon. Must be called before graf() or scales().

Multi-Language Signatures:

C	`void grid(int gridon); void fgrid(int fon, int ndiv);`
Python 3	`graphic.grid(gridon: int); graphic.fgrid(fon: int, ndiv: int)`
Java 22	`GraphiC.grid(int gridon); GraphiC.fgrid(int fon, int ndiv);`
Fortran	`call gpc_grid(gridon); call gpc_fgrid(fon, ndiv)`

void tickout(int is_out); void tickht(float height); GRIDOPTS.C

Level 2

int is_out	1 = point outward, 0 = point inward.
float height	Tick height in inches.

Controls axis tick mark appearance.

tickout(1): Reverses default inward-pointing ticks to point outward from the plot box. tickht(height): Sets physical length of tick marks in inches (default ~ 0.08 in). Must be called before graf().

Multi-Language Signatures:

C	`void tickout(int is_out); void tickht(float height);`
Python 3	`graphic.tickout(is_out: int); graphic.tickht(height: float)`
Java 22	`GraphiC.tickout(int is_out); GraphiC.tickht(float height);`

Fortran	`call gpc_tickout(is_out); call gpc_tickht(height)`

void xfgrid(int fon, int nx); void yfgrid(int fon, int ny); LINPLT.C

Level 1

int fon	Line style (1-9) for fine grid lines (uses dashf patterns). Set 0 for no subdivisions (default).
int nx	Desired number of tick subdivisions between major grid lines (e.g. nx=9 creates 10 sub-intervals).

Configures fine grid subdivisions intersecting the X or Y axis.

xfgrid(): Subdivides intervals between main X grid lines. yfgrid(): Subdivides intervals between main Y grid lines.

Multi-Language Signatures:

C	`void xfgrid(int fon, int nx); void yfgrid(int fon, int ny);`
Python 3	`graphic.xfgrid(fon: int, nx: int) -> None; graphic.yfgrid(fon: int, ny: int) -> None`
Java 22	`GraphiC.xfgrid(int fon, int nx); GraphiC.yfgrid(int fon, int ny);`
Fortran	`call gpc_xfgrid(fon, nx); call gpc_yfgrid(fon, ny)`

void xlab(char on_off, char **strlab); void ylab(char on_off, char **strlab); LINPLT.C

Level 1

char / int on_off	1 = enable custom string labels; 0 = disable and use numeric format (default).
char strlab**	Null-terminated array of string pointers corresponding to tick positions.

Supplies custom text labels (e.g. month names "Jan", "Feb", categories) for axis tick marks.

When enabled, GraphiC spaces the supplied string labels uniformly along the axis.

Multi-Language Signatures:

C	`void xlab(char on_off, char **strlab); void ylab(char on_off, char **strlab);`
Python 3	`graphic.xlab(on_off: int, strlab: list[str]) -> None; graphic.ylab(on_off: int, strlab: list[str]) -> None`
Java 22	`GraphiC.xlab(byte on_off, String[] strlab); GraphiC.ylab(byte on_off, String[] strlab);`
Fortran	`call gpc_xlab(on_off, strlab); call gpc_ylab(on_off, strlab)`

void xname(char *xlabel); void yname(char *ylabel); void heading(char *title); — FNTPLT.C

Level 1

char* xlabel	Label string for horizontal X-axis.
char* ylabel	Label string for vertical Y-axis.
char* title	Main heading string centered above plot.

Sets descriptive title strings for axes and the overall plot heading.

Labels are centered along their respective axes. In modern GraphiC, strings support full formatting escape codes for superscripts, subscripts, Greek characters, and font switching. Must be called before graf() or scales().

Multi-Language Signatures:

C	`void xname(char *xlabel); void yname(char *ylabel); void heading(char *title);`
Python 3	`graphic.xname(xlabel: str); graphic.yname(ylabel: str); graphic.heading(title: str)`
Java 22	`GraphiC.xname(String xlabel); GraphiC.yname(String ylabel); GraphiC.heading(String title);`
Fortran	`call gpc_title(title, xlabel, ylabel)`

Simultaneous Plots

Manages independent simultaneous subplots on a single page, context switching, and multi-threaded independent viewports.

void simuplot(int numplots) SIMPLT.C

Level 1

int numplots Total number of simultaneous subplots; `0` finalizes simultaneous mode.

Enables simultaneous plotting on multiple independent axes systems.

Allows concurrent rendering of multiple physical perspectives or simulation states within the same overall calculation loop. Subplots are referenced by zero-based index via context(). Calling simuplot(0) terminates simultaneous plotting mode and consolidates all sub-streams into the final output.

Multi-Language Signatures:

C	`void simuplot(int numplots)`
Python 3	`graphic.simuplot(numplots: int) -> None`
Java 22	`GraphiC.simuplot(int numplots)`
Fortran	`call gpc_simuplot(numplots)`

void context(int plot_number) SIMPLT.C

Level 3

int plot_number The active subplot context index.

Switches the active plotting context to subplot plot_number.

Directs subsequent GraphiC calls (axes, curves, labels, titles) to the coordinate frame, viewport bounds, and scaling factors of the specified subplot. Numbers run consecutively from 0 to (numplots - 1).

Multi-Language Signatures:

C	`void context(int plot_number)`
Python 3	`graphic.context(plot_number: int) -> None`
Java 22	`GraphiC.context(int plot_number)`
Fortran	`call gpc_context(plot_number)`

void plotabsolute(int set_flag) SVC.C

Level 2

int set_flag 1 = absolute positioning, 0 = relative positioning.

Sets coordinate origin mode for compound multi-plot layouts.

When enabled (set_flag = 1), origin shifts performed by physor() or orel() are referenced to the absolute physical page boundary rather than the current subplot boundary.

Multi-Language Signatures:

C	`void plotabsolute(int set_flag)`
Python 3	`graphic.plotabsolute(set_flag: int) -> None`
Java 22	`GraphiC.plotabsolute(int set_flag)`
Fortran	`call gpc_plotabsolute(set_flag)`

2-D Axes Routines

Calibrates linear and logarithmic axes, computes optimum round tick intervals, renders data curves, error bars, and bar charts.

void graf(char *xformat, float xori, float xste, float xma, char *yformat, float yori, float yste, float yma) — LINPLT.C

Level 1

char* xformat	Format string for X-axis numbers (`"%g"` or `""` for no labels).
float xori	Starting X value at left axis boundary.
float xste	Major step spacing between X ticks.
float xma	Ending X value at right axis boundary.
char* yformat	Format string for Y-axis numbers.
float yori	Starting Y value at bottom axis boundary.
float yste	Major step spacing between Y ticks.
float yma	Ending Y value at top axis boundary.

Establishes mathematical user coordinate mapping and draws calibrated 2D axes.

Calculates coordinate transformation matrix converting data values into physical inches and pixel coordinates. Draws tick marks and formatted numerical labels along the X and Y axes according to the provided C printf format specifications.

Multi-Language Signatures:

C	`void graf(char *xformat, float xori, float xste, float xma, char *yformat, float yori, float yste, float yma)`
Python 3	`graphic.graf(xformat: str, xori: float, xste: float, xma: float, yformat: str, yori: float, yste: float, yma: float) -> None`
Java 22	`GraphiC.graf(String xformat, float xori, float xste, float xma, String yformat, float yori, float yste, float yma)`
Fortran	`call gpc_graf(xformat, xori, xste, xma, yformat, yori, yste, yma)`

void scales(int nxdiv, int nydiv, float *x, float *y, int npts) LINPLT.C

Level 1

int nxdiv	Approximate desired number of X divisions (e.g. 5 to 10).
int nydiv	Approximate desired number of Y divisions (e.g. 5 to 10).
float* x	Input array of X data points.
float* y	Input array of Y data points.
int npts	Number of points in arrays.

Automatically inspects data arrays x and y, calculates aesthetically rounded minimum, maximum, and tick step bounds, and invokes graf() automatically.

Multi-Language Signatures:

C	`void scales(int nxdiv, int nydiv, float *x, float *y, int npts)`
Python 3	`graphic.scales(nxdiv: int, nydiv: int, x: list[float], y: list[float], npts: int) -> None`
Java 22	`GraphiC.scales(int nxdiv, int nydiv, float[] x, float[] y, int npts)`
Fortran	`call gpc_scales(nxdiv, nydiv, x, y, npts)`

void xlog(float xorig, float xmax, char *format, float yorig, float ystep, float ymax); void ylog(char *format, float xorig, float xstep, float xmax, float yorig, float ymax); void loglog(float xorig, float xmax, float yorig, float ymax); LOGPLT.C

Level 1

float xorig	Minimum decade value (> 0.0, e.g. 0.01, 1.0).
float xmax	Maximum decade value ($> xorig$, e.g. 1000.0).
float yorig	Minimum Y value.
float ymax	Maximum Y value.

Calibrates logarithmic coordinate axes.

xlog(): Logarithmic X-axis, linear Y-axis (semi-log). ylog(): Linear X-axis, logarithmic Y-axis (semi-log). loglog(): Logarithmic X-axis and Logarithmic Y-axis (log-log).

Decades are labeled with scientific powers of 10 ($10^{-2}, 10^{-1}, 10^0, 10^1$). Intermediate tick marks (2, 3, 4, 5, 6, 7, 8, 9) are placed automatically.

Multi-Language Signatures:

C	`void xlog(float xorig, float xmax, char *format, float yorig, float ystep, float ymax); void ylog(char *format, float xorig, float xstep, float xmax, float yorig, float ymax); void loglog(float xorig, float xmax, float yorig, float ymax);`
Python 3	`graphic.xlog(...); graphic.ylog(...); graphic.loglog(...)`

Java 22	`GraphiC.xlog(...); GraphiC.ylog(...); GraphiC.loglog(...)`
Fortran	`call gpc_xlog(...); call gpc_ylog(...); call gpc_loglog(...)`

void lglgscle(float *x, float *y, int npts) LOGPLT.C

Level 2

float* x	Array of positive non-zero X data points.
float* y	Array of positive non-zero Y data points.
int npts	Number of points in arrays.

Scans arrays x and y, calculates min and max values, and rounds outwards to integer powers of 10 to calibrate a log-log coordinate grid.

Multi-Language Signatures:

C	`void lglgscle(float *x, float *y, int npts)`
Python 3	`graphic.lglgscle(x: list[float], y: list[float], npts: int) -> None`
Java 22	`GraphiC.lglgscle(float[] x, float[] y, int npts)`
Fortran	`call gpc_lglgscle(x, y, npts)`

void curve(float *x, float *y, int npts, int sym) CURVES.C

Level 1

float* x	Array of horizontal coordinates.
float* y	Array of vertical coordinates.
int npts	Total number of points.
int sym	Symbol interval: `0` = line only; `> 0` = line + symbols; `< 0` = symbols only.

Plots a 2D data series connecting data coordinates (x[i], y[i]).

Transforms data coordinates through the active calibration matrix into physical drawing commands. Automatically clips polyline segments at the area2d boundary.

sym = 0 : Draw connected line only. sym > 0 : Draw line and place marker symbol every sym points. sym < 0 : Draw marker symbol ONLY (scatter plot) every |sym| points.

Multi-Language Signatures:

C	`void curve(float *x, float *y, int npts, int sym)`
Python 3	`graphic.curve(x: list[float], y: list[float], npts: int, sym: int) -> None`
Java 22	`GraphiC.curve(float[] x, float[] y, int npts, int sym)`
Fortran	`call gpc_curve(x, y, npts, sym)`

void curvfill(float *x, float *y, int npts, float *x1, float *y1, int npts1, int pcolor, int curv) — SVC.C

Level 3

float* x	X data vector for the first bounding curve.
float* y	Y data vector for the first bounding curve.
int npts	Number of points in the first curve.
float* x1	X data vector for the second bounding curve.
float* y1	Y data vector for the second bounding curve.
int npts1	Number of points in the second curve.
int pcolor	Fill color index or cross-hatch pattern code.
int curv	1 = draw curve boundary outlines in active color; 0 = fill region only without drawing outlines.

Fills the area between curve (x, y) and curve (x1, y1).

If the two curves do not share identical endpoints, GraphiC connects the extremities to form a closed polygon boundary before applying the fill pattern.

Multi-Language Signatures:

C	`void curvfill(float *x, float *y, int npts, float *x1, float *y1, int npts1, int pcolor, int curv)`
Python 3	`graphic.curvfill(x: list[float], y: list[float], npts: int, x1: list[float], y1: list[float], npts1: int, pcolor: int, curv: int) -> None`
Java 22	`GraphiC.curvfill(float[] x, float[] y, int npts, float[] x1, float[] y1, int npts1, int pcolor, int curv)`
Fortran	`call gpc_curvfill(x, y, npts, x1, y1, npts1, pcolor, curv)`

void errorbar(float x, float xmin, float xmax, float y, float ymin, float ymax) — CURVES.C

Level 2

float x	Central X data point.
float xmin	Lower X bound (set equal to x to omit horizontal error bar).
float xmax	Upper X bound.
float y	Central Y data point.
float ymin	Lower Y bound (set equal to y to omit vertical error bar).
float ymax	Upper Y bound.

Draws error bars centered at (x, y) with horizontal range [xmin, xmax] and vertical range [ymin, ymax]. End tick bars are drawn automatically.

Multi-Language Signatures:

C	`void errorbar(float x, float xmin, float xmax, float y, float ymin, float ymax)`
Python 3	`graphic.errorbar(x: float, xmin: float, xmax: float, y: float, ymin: float, ymax: float) -> None`
Java 22	`GraphiC.errorbar(float x, float xmin, float xmax, float y, float ymin, float ymax)`
Fortran	`call gpc_errorbar(x, xmin, xmax, y, ymin, ymax)`

void bar(int nxdiv, float *x, float *y, int npts, float bwid, int mode, int *car, int *tpe); void histogram(float *x, float *y, int npts, int type, int *colors); BARPLT.C

Level 1

int nxdiv	Number of bar groups along the X axis.
float* x	Bar center coordinates.
float* y	Bar height values.
int npts	Total number of data values.
float bwid	Bar physical width in inches (e.g. 0.35 in).
int mode	Stacking mode (0 = side-by-side, 1 = stacked).
int* car	Array of color indices for each bar segment.
int* tpe	Array of hatch fill patterns (0-15) for each bar segment.

Renders statistical column or bar charts with pattern hatching and color fills.

bar(): Renders grouped or segmented bars. Supports custom widths, side-by-side comparisons, and distinct hatch fills per category. histogram(): Divides continuous data into frequency bins and plots rectangular steps.

Multi-Language Signatures:

C	`void bar(int nxdiv, float *x, float *y, int npts, float bwid, int mode, int *car, int *tpe); void histogram(float *x, float *y, int npts, int type, int *colors);`
Python 3	`graphic.bar(nxdiv, x, y, npts, bwid, mode, car, tpe); graphic.histogram(x, y, npts, type, colors)`
Java 22	`GraphiC.bar(int nxdiv, float[] x, float[] y, int npts, float bwid, int mode, int[] car, int[] tpe); GraphiC.histogram(...)`
Fortran	`call gpc_bar(nxdiv, x, y, npts, bwid, mode, car, tpe)`

void barchart(int nbars, float *xarray, float *yarray, int npts, float width, int *carray, int *typearray) — BARPLT.C

Level 3

int nbars	Number of bar positions along the X-axis.
float* xarray	Array of bar center positions.
float* yarray	Array of bar heights.
int npts	Total dimension of x and y arrays.
float width	Bar width expressed as a fraction of the X interval (1.0 = adjacent touching bars).
int* carray	Array of color or hatch pattern indices for bars.
int* typearray	Array defining bar arrangement:

Renders categorized bar charts with custom bar widths and individual colors.

0 = Standard individual bars. 1 = Stacked bars for repeated X positions. 2 = Grouped side-by-side bars for repeated X positions.

Multi-Language Signatures:

C	`void barchart(int nbars, float *xarray, float *yarray, int npts, float width, int *carray, int *typearray)`
Python 3	`graphic.barchart(nbars: int, xarray: list[float], yarray: list[float], npts: int, width: float, carray: list[int], typearray: list[int]) -> None`
Java 22	`GraphiC.barchart(int nbars, float[] xarray, float[] yarray, int npts, float width, int[] carray, int[] typearray)`
Fortran	`call gpc_barchart(nbars, xarray, yarray, npts, width, carray, typearray)`

void histogram(float *x, float *y, int npts, int type, int *colors) BARPLT.C

Level 3

float* x	Array of bin boundary values.
float* y	Array of bin frequency counts.
int npts	Number of bins.
int type	Histogram style (0 = stepped line profile, 1 = filled vertical columns).
int* colors	Array of color indices for bins.

Plots a statistical histogram given bin boundaries and frequency counts.

Multi-Language Signatures:

C	`void histogram(float *x, float *y, int npts, int type, int *colors)`
Python 3	`graphic.histogram(x: list[float], y: list[float], npts: int, type: int, colors: list[int]) -> None`
Java 22	`GraphiC.histogram(float[] x, float[] y, int npts, int type, int[] colors)`
Fortran	`call gpc_histogram(x, y, npts, type, colors)`

void scatplot(char dot) SVC.C

Level 3

char / int dot	1 = enable dot-only scatter mode; 0 = normal line drawing (default).

Configures scatter plot marker mode for curve().

When dot=1, passing sym=0 to curve() draws an individual pixel dot at each data point without interconnecting line segments. Passing sym=-1 draws selected glyph symbols at every point without connecting lines.

Multi-Language Signatures:

C	`void scatplot(char dot)`
Python 3	`graphic.scatplot(dot: int) -> None`
Java 22	`GraphiC.scatplot(byte dot)`
Fortran	`call gpc_scatplot(dot)`

Smith Charts

Generates transmission line impedance Smith charts, constant-VSWR circles, normalized resistance/reactance arcs, and electrical wavelength translations.

void smdraw(Uchar lcolor) — SMITH.C

Level 1

Uchar lcolor Grid line palette color index.

Draws the complete Smith impedance/admittance chart coordinate system.

Renders constant-resistance circles ($R/Z_0 = 0, 0.2, 0.5, 1.0, 2.0, 5.0$), constant-reactance arcs ($\pm j X/Z_0$), outer unit circumference ($|\Gamma| = 1.0$), and reflection coefficient angle scales in physical vector lines.

Multi-Language Signatures:

C	`void smdraw(Uchar lcolor)`
Python 3	`graphic.smdraw(lcolor: int) -> None`
Java 22	`GraphiC.smdraw(byte lcolor)`
Fortran	`call gpc_smdraw(lcolor)`

void plotrx(float r, float x); void plotrho(float rho, float theta); void smove(float dell); — SMITH.C

Level 1

float r Normalized resistance $R / Z_0 \ge 0$.

float x Normalized reactance X / Z_0.

float rho

float theta Reflection coefficient phase angle in degrees.

Renders impedance data onto the Smith chart.

plotrx(r, x): Plots a normalized impedance point $Z = r + j x$. plotrho(rho, theta): Plots a reflection coefficient point $\Gamma = | ho| e^{j heta}$. smove(dell): Draws a line at constant $|\Gamma|$ through electrical length dell (radians).

ho| \le 1.0$).

Multi-Language Signatures:

C	`void plotrx(float r, float x); void plotrho(float rho, float theta); void smove(float dell);`
Python 3	`graphic.plotrx(r, x); graphic.plotrho(rho, theta); graphic.smove(dell)`
Java 22	`GraphiC.plotrx(float r, float x); GraphiC.plotrho(float rho, float theta); GraphiC.smove(float dell)`

Fortran	`call gpc_plotrx(r, x); call gpc_plotrho(rho, theta); call gpc_smove(dell)`

void smove(float dell) SMITH.C

Level 3

float dell Distance in wavelengths along the line.

Steps a distance dell along a uniform transmission line toward the generator or load.

Multi-Language Signatures:

C	`void smove(float dell)`
Python 3	`graphic.smove(dell: float) -> None`
Java 22	`GraphiC.smove(float dell)`
Fortran	`call gpc_smove(dell)`

void smithcir(float u, float v, float radius) SMITH.C

Level 3

float u Real coordinate of circle center.

float v Imaginary coordinate of circle center.

float radius Radius in reflection coefficient units.

Draws a circle centered at (u, v) with specified radius on the active Smith chart.

Multi-Language Signatures:

C	`void smithcir(float u, float v, float radius)`
Python 3	`graphic.smithcir(u: float, v: float, radius: float) -> None`
Java 22	`GraphiC.smithcir(float u, float v, float radius)`
Fortran	`call gpc_smithcir(u, v, radius)`

Polar Plots

Draws circular polar grids, radial decibel/linear scales, polar data curves, and angular sectors.

void polgrid(float rmax, float rmin, int mode, char *larray[], int dim, int ntheta, int col); void polcurve(float *r, float *theta, unsigned int dim, int sym); POLPLT.C

Level 1

int ntheta	Number of angular division rays (e.g. 12 for 30-deg rays, 24 for 15-deg rays).
int dim	Number of points.

Generates polar coordinate systems and angular plots.

polgrid(...): Draws concentric radial range circles (from rmin to rmax) and radial rays dividing 360 degrees into ntheta angular spokes.

Multi-Language Signatures:

C	`void polgrid(float rmax, float rmin, int mode, char *larray[], int dim, int ntheta, int col); void polcurve(float *r, float *theta, unsigned int dim, int sym);`
Python 3	`graphic.polgrid(...); graphic.polcurve(...)`
Java 22	`GraphiC.polgrid(...); GraphiC.polcurve(...)`
Fortran	`call gpc_polgrid(...); call gpc_polcurve(...)`

void polcurve(float *r, float *theta, int npts, int sym) POLPLT.C

Level 3

float* r	Array of radial magnitudes.
float* theta	Array of angles in radians.
int npts	Number of points.
int sym	Symbol selector (0 = line only, >0 = symbol every sym points, <0 = symbol only).

Connects polar points (r[i], theta[i]) where theta is in degrees. wedge(...): Fills a circular wedge bounded by [rmin, rmax] and [thmin, thmax] with pattern.

Plots data points given radial vector r and angular vector theta (in radians).

Multi-Language Signatures:

C	`void polcurve(float *r, float *theta, int npts, int sym)`
Python 3	`graphic.polcurve(r: list[float], theta: list[float], npts: int, sym: int) -> None`
Java 22	`GraphiC.polcurve(float[] r, float[] theta, int npts, int sym)`

Fortran	`call gpc_polcurve(r, theta, npts, sym)`

void wedge(float rmin, float rmax, float thmin, float thmax, int pattern, int border) — POLPLT.C

Level 3

float rmin	Inner radius.
float rmax	Outer radius.
float thmin	Starting angle in degrees.
float thmax	Ending angle in degrees.
int pattern	Fill color index or cross-hatch pattern code.
int border	1 = draw outline border; 0 = fill only.

Fills an annular sector bounded by [rmin, rmax] and angles [thmin, thmax] (in degrees).

Multi-Language Signatures:

C	`void wedge(float rmin, float rmax, float thmin, float thmax, int pattern, int border)`
Python 3	`graphic.wedge(rmin: float, rmax: float, thmin: float, thmax: float, pattern: int, border: int) -> None`
Java 22	`GraphiC.wedge(float rmin, float rmax, float thmin, float thmax, int pattern, int border)`
Fortran	`call gpc_wedge(rmin, rmax, thmin, thmax, pattern, border)`

Contour Plots

Calculates 2D isoclines across regularly or irregularly spaced data grids, elevation color coding, contour stacking, and ternary phase diagrams.

void conmat(float *zmat, float *x, int nx, float *y, int ny, float orig, float step, float maxq, int *lnstyle, int labint) — CNTPLT.C

Level 1

float* zmat	Matrix of elevation values.
float* x	Coordinates of grid columns.
int nx	Column dimension.
float* y	Coordinates of grid rows.
int ny	Row dimension.
float orig	Starting contour height.
float step	Step spacing between contours.
float maxq	Ending contour height.
int* lnstyle	Line style assigned to each level.
int labint	Label frequency along isoclines.

Generates and draws contour elevation curves across a regular 2D rectangular grid.

Scans elevation matrix cells, locates isocline crossings via linear interpolation, threads smooth continuous contour lines across adjacent cells, and formats elevation value numbers inline along contour paths according to labint.

0 = no numeric labels, 1 = label every line, 2 = label every 2nd line, etc.

Multi-Language Signatures:

C	`void conmat(float *zmat, float *x, int nx, float *y, int ny, float orig, float step, float maxq, int *lnstyle, int labint)`
Python 3	`graphic.conmat(zmat, x, nx, y, ny, orig, step, maxq, lnstyle, labint)`
Java 22	`GraphiC.conmat(float[] zmat, float[] x, int nx, float[] y, int ny, float orig, float step, float maxq, int[] lnstyle, int labint)`
Fortran	`call gpc_conmat(zmat, x, nx, y, ny, orig, step, maxq, lnstyle, labint)`

void lconmat(float *zmat, float *x, int nx, float *y, int ny, float *level, int *lnstyle, int nlevel, int labint, char *formt) — CNTPLT.C

Level 2

float* level	Array of explicit contour elevation heights.

int nlevel	Number of custom levels.
char* formt	Custom printf format string for inline elevation numbers (e.g. "%.1f").

Traces contour isoclines at custom arbitrary elevation thresholds.

Unlike conmat() which steps at uniform intervals, lconmat() accepts an explicit array of arbitrary elevation values, allowing clustered lines near steep gradients.

Multi-Language Signatures:

C	`void lconmat(float *zmat, float *x, int nx, float *y, int ny, float *level, int *lnstyle, int nlevel, int labint, char *formt)`
Python 3	`graphic.lconmat(zmat, x, nx, y, ny, level, lnstyle, nlevel, labint, formt)`
Java 22	`GraphiC.lconmat(float[] zmat, float[] x, int nx, float[] y, int ny, float[] level, int[] lnstyle, int nlevel, int labint, String formt)`
Fortran	`call gpc_lconmat(zmat, x, nx, y, ny, level, lnstyle, nlevel, labint, formt)`

void contcolor(int *colors, int border, ...); void ColorKeyL(float minv, float maxv, ...); — CNTPLT.C

Level 2

int* colors	Array of palette color indices matching contour levels.

Configures color-mapped contour rendering and companion legend key.

contcolor(colors, border): Assigns an array of color indices to successive contour levels. ColorKeyL(...): Draws a graduated color scale key rectangle annotated with numeric tick bounds.

Multi-Language Signatures:

C	`void contcolor(int *colors, int border, ...); void ColorKeyL(float minv, float maxv, ...);`
Python 3	`graphic.contcolor(colors, border); graphic.ColorKeyL(minv, maxv, ...)`
Java 22	`GraphiC.contcolor(int[] colors, int border); GraphiC.ColorKeyL(float minv, float maxv, ...)`
Fortran	`call gpc_contcolor(colors, border); call gpc_ColorKeyL(minv, maxv, ...)`

void lcontour(GPC_FUNQ fun, float *x, int nx, float *y, int ny, float *level, int nlevel, int *lnstyle, int labint, char *format) — CNTPLT.C

Level 3

GPC_FUNQ fun	Function pointer returning float elevation z given (float x, float y).
float* x	Array of grid X values (length nx).
int nx	Number of X grid points (MUST be odd).
float* y	Array of grid Y values (length ny).

int ny	Number of Y grid points (MUST be odd).
float* level	Array of target contour elevation values in ascending order.
int nlevel	Number of contour levels.
int* lnstyle	Array of 3-digit coded integers specifying line style and color:
int labint	Numerical label interval (e.g. 1 = label every line, 2 = alternate lines).
char* format	C-style format string for elevation labels (e.g. "%-3.1f").

Evaluates function fun across a regular grid of nx by ny points and generates smooth interpolated contour lines at the specified elevation levels.

Grid dimensions nx and ny MUST be odd integers for the interpolation algorithm.

The first 2 digits define line style (1-9); the 3rd digit defines color index (1-15). For example, 104 = line style 1 in color 4.

Multi-Language Signatures:

C	`void lcontour(GPC_FUNQ fun, float *x, int nx, float *y, int ny, float *level, int nlevel, int *lnstyle, int labint, char *format)`
Python 3	`graphic.lcontour(fun: Callable[[float, float], float], x: list[float], nx: int, y: list[float], ny: int, level: list[float], nlevel: int, lnstyle: list[int], labint: int, format: str) -> None`
Java 22	`GraphiC.lcontour(MemorySegment fun, float[] x, int nx, float[] y, int ny, float[] level, int nlevel, int[] lnstyle, int labint, String format)`
Fortran	`call gpc_lcontour(fun, x, nx, y, ny, level, nlevel, lnstyle, labint, format)`

void stack(char stacked) — CNTPLT.C

Level 3

char / int stacked	1 = enable 3D perspective stacking; 0 = standard flat 2D contours (default).

Enables or disables 3D elevation stacking for 2D contour maps.

When stacked=1, subsequent contour maps or workboxes are offset in 3D space according to plane3d calibration, creating multi-tier topographical slices.

Multi-Language Signatures:

C	`void stack(char stacked)`
Python 3	`graphic.stack(stacked: int) -> None`
Java 22	`GraphiC.stack(byte stacked)`
Fortran	`call gpc_stack(stacked)`

Triangle Plots

Triangle plots provide for the graphic display of an attribute of a three-component mixture. Each component is associated with a vertex. The fraction of that component is plotted on a line parallel to the opposite side. The position of the line is proportionally closer to the vertex depending on the relative amount of the component.

void trifun(GPC_FUNQ fun, int dim, float zmin, float zstep, float zmax, int *lnstyle, int border) — TRIANGLE.C

Level 3

GPC_FUNQ fun	Function returning attribute z given triangular coordinates (float a, float b).
int dim	Number of grid points along each triangular edge.
float zmin	Minimum contour level.
float zstep	Contour level spacing.
float zmax	Maximum contour level.
int* lnstyle	Array of 3-digit coded integers for line styles and colors.
int border	1 = draw triangular boundary frame; 0 = contours only.

Evaluates mathematical function fun across a triangular coordinate simplex and renders ternary contour lines.

Multi-Language Signatures:

C	`void trifun(GPC_FUNQ fun, int dim, float zmin, float zstep, float zmax, int *lnstyle, int border)`
Python 3	`graphic.trifun(fun: Callable[[float, float], float], dim: int, zmin: float, zstep: float, zmax: float, lnstyle: list[int], border: int) -> None`
Java 22	`GraphiC.trifun(MemorySegment fun, int dim, float zmin, float zstep, float zmax, int[] lnstyle, int border)`
Fortran	`call gpc_trifun(fun, dim, zmin, zstep, zmax, lnstyle, border)`

void trimat(...); void aname(char *alabel); void bname(char *blabel); void cname(char *clabel); — TRIANGLE.C

Level 2

float* z	Elevation data array over triangular domain.

Renders ternary phase diagrams where components satisfy A + B + C = 100%.

trimat(): Generates contour isoclines over a triangular phase space. aname(), bname(), cname(): Sets vertex labels for the three constituent components.

Multi-Language Signatures:

C	`void trimat(...); void aname(char *alabel); void bname(char *blabel); void cname(char *clabel);`
Python 3	`graphic.trimat(...); graphic.aname(al); graphic.bname(bl); graphic.cname(cl)`
Java 22	`GraphiC.trimat(...); GraphiC.aname(String al); GraphiC.bname(String bl); GraphiC.cname(String cl)`
Fortran	`call gpc_trimat(...); call gpc_aname(al); call gpc_bname(bl); call gpc_cname(cl)`

Three-Dimensional Plots

Renders 3D perspective surface meshes with hidden-line horizon removal, 3D coordinate boxes, space trajectories, and cross-sectional planes.

void volm3d(float x3axis, float y3axis, float z3axis) — D3PLT.C

Level 1

float x3axis	Physical width of 3D box along X in inches.
float y3axis	Physical depth of 3D box along Y in inches.
float z3axis	Physical height of 3D box along Z in inches.

Sets the physical size of the 3D workbox enclosing the 3D surface plot in inches.

Must be called before graf3d() or vuabs().

Multi-Language Signatures:

C	`void volm3d(float x3axis, float y3axis, float z3axis)`
Python 3	`graphic.volm3d(x3axis: float, y3axis: float, z3axis: float) -> None`
Java 22	`GraphiC.volm3d(float x3axis, float y3axis, float z3axis)`
Fortran	`call gpc_volm3d(x3axis, y3axis, z3axis)`

void vuabs(float xvu, float yvu, float zvu); void vuangl(float phi, float theta, float radius); void view(float a, float b, float c); — D3PLT.C

Level 1

float phi	Azimuth rotation angle in degrees (e.g. 30.0).
float theta	Elevation angle in degrees (e.g. 25.0).
float radius	Viewing distance multiplier.

Establishes the 3D camera eye viewpoint.

vuabs(xvu, yvu, zvu): Positions observer in scaled 3D coordinate space. vuangl(phi, theta, radius): Positions observer via spherical angles: phi = azimuth angle in X-Y plane (degrees), theta = elevation angle above Z plane (degrees), radius = eye distance factor. view(a, b, c): Positions viewpoint in user data space coordinates.

Multi-Language Signatures:

C	`void vuabs(float xvu, float yvu, float zvu); void vuangl(float phi, float theta, float radius); void view(float a, float b, float c);`
Python 3	`graphic.vuabs(...); graphic.vuangl(...); graphic.view(...)`
Java 22	`GraphiC.vuabs(...); GraphiC.vuangl(...); GraphiC.view(...)`
Fortran	`call gpc_vuangl(phi, theta, radius)`

void graf3d(float xorig, float xstp, float xmax, float yorig, float ystp, float ymax, float zorig, float zstp, float zmax) D3PLT.C

Level 1

Draws the 3D coordinate box and sets linear data bounds for X, Y, and Z.

Renders back and front perspective pillars, baseline frames, and tick subdivisions.

Multi-Language Signatures:

C	`void graf3d(float xorig, float xstp, float xmax, float yorig, float ystp, float ymax, float zorig, float zstp, float zmax)`
Python 3	`graphic.graf3d(...)`
Java 22	`GraphiC.graf3d(...)`
Fortran	`call gpc_graf3d(...)`

void surmat(float *zmat, int xdim, int ydim); void surmat2(float **zmat, int xdim, int ydim); D3PLT.C

Level 1

float* zmat	Elevation matrix.
int xdim	Number of grid points along X.
int ydim	Number of grid points along Y.

Evaluates and draws a 3D perspective surface mesh.

Implements GraphiC's analytical floating horizon algorithm: Traces grid lines from front to back, updating upper and lower horizon profile arrays. Vector segments occluded by foreground peaks are clipped in analytical coordinate space, producing clean vector output without depth-buffering artifacts.

surmat(): 1D row-major array zmat[xdim * ydim]. surmat2(): 2D pointer-to-pointer array zmat[ydim][xdim]. surfun(): Evaluates an analytical C function f(x, y) dynamically.

Multi-Language Signatures:

C	`void surmat(float *zmat, int xdim, int ydim); void surmat2(float **zmat, int xdim, int ydim);`
Python 3	`graphic.surmat(zmat, xdim, ydim); graphic.surmat2(zmat, xdim, ydim)`
Java 22	`GraphiC.surmat(float[] zmat, int xdim, int ydim); GraphiC.surmat2(float[][] zmat, int xdim, int ydim)`
Fortran	`call gpc_surmat(zmat, xdim, ydim)`

void surfun(GPC_FUNQ zfun, int ixpts, float xdel, int iypts, float ydel) D3PLT.C

Level 3

GPC_FUNQ zfun	User-supplied mathematical function returning z = zfun(x, y).
int ixpts	Number of interpolation subdivisions along X.
float xdel	Step size between X grid lines in user units.
int iypts	Number of interpolation subdivisions along Y.
float ydel	Step size between Y grid lines in user units.

Evaluates function zfun across an interpolated grid and renders a 3D surface mesh with analytical hidden-line horizon removal.

Multi-Language Signatures:

C	`void surfun(GPC_FUNQ zfun, int ixpts, float xdel, int iypts, float ydel)`
Python 3	`graphic.surfun(zfun: Callable[[float, float], float], ixpts: int, xdel: float, iypts: int, ydel: float) -> None`
Java 22	`GraphiC.surfun(MemorySegment zfun, int ixpts, float xdel, int iypts, float ydel)`
Fortran	`call gpc_surfun(zfun, ixpts, xdel, iypts, ydel)`

void survis(char *str); void nohide(int visible); void tcolor(Uchar value); void bcolor(Uchar value); D3PLT.C

Level 2

char* str	"TOP", "BOTTOM", or "BOTH".
int visible	1 = no hidden lines (transparent), 0 = standard hidden lines.
Uchar value	Palette color index.

Configures surface facet visibility and contrasting mesh colors.

survis(str): Selects visible sides: "TOP", "BOTTOM", or "BOTH". nohide(1): Disables hidden-line removal to render a transparent see-through wireframe. tcolor(col): Sets color index for top of surface (e.g. BLUE). bcolor(col): Sets color index for bottom of surface (e.g. RED).

Multi-Language Signatures:

C	`void survis(char *str); void nohide(int visible); void tcolor(Uchar value); void bcolor(Uchar value);`
Python 3	`graphic.survis(str); graphic.nohide(visible); graphic.tcolor(val); graphic.bcolor(val)`
Java 22	`GraphiC.survis(String str); GraphiC.nohide(int visible); GraphiC.tcolor(byte val); GraphiC.bcolor(byte val)`
Fortran	`call gpc_survis(str); call gpc_nohide(visible); call gpc_tcolor(val); call gpc_bcolor(val)`

void box3d(void) D3PLT.C

Level 3

Draws the 3D workbox framework established by graf3d() and view()/vuabs().

See Also: `graf3d, vuabs, sclfix`

Multi-Language Signatures:

C	`void box3d(void)`
Python 3	`graphic.box3d() -> None`
Java 22	`GraphiC.box3d()`
Fortran	`call gpc_box3d()`

void curv3d(float *x, float *y, float *z, int npts) D3PLT.C

Level 3

float* x	Array of X coordinates in user units.
float* y	Array of Y coordinates in user units.
float* z	Array of Z coordinates in user units.
int npts	Number of points in the curve.

Renders a 3D space curve given coordinate vectors x, y, z.

Multi-Language Signatures:

C	`void curv3d(float *x, float *y, float *z, int npts)`
Python 3	`graphic.curv3d(x: list[float], y: list[float], z: list[float], npts: int) -> None`
Java 22	`GraphiC.curv3d(float[] x, float[] y, float[] z, int npts)`
Fortran	`call gpc_curv3d(x, y, z, npts)`

void d3color(int *array, int nlevels, int mesh) D3PLT.C

Level 2

int* array	Array of color palette indices corresponding to elevation intervals.
int nlevels	Number of color bands from zmin to zmax.
int mesh	1 = render surface mesh lines over color fill; 0 = smooth filled facets only.

Configures elevation color coding for 3D surfaces.

Multi-Language Signatures:

C	`void d3color(int *array, int nlevels, int mesh)`
Python 3	`graphic.d3color(array: list[int], nlevels: int, mesh: int) -> None`
Java 22	`GraphiC.d3color(int[] array, int nlevels, int mesh)`
Fortran	`call gpc_d3color(array, nlevels, mesh)`

void d3line(float xf, float yf, float zf, float xt, float yt, float zt) D3PLT.C

Level 3

Draws a 3D line from (xf, yf, zf) to (xt, yt, zt) in perspective projection.

Multi-Language Signatures:

C	`void d3line(float xf, float yf, float zf, float xt, float yt, float zt)`
Python 3	`graphic.d3line(xf: float, yf: float, zf: float, xt: float, yt: float, zt: float) -> None`

Java 22	`GraphiC.d3line(float xf, float yf, float zf, float xt, float yt, float zt)`
Fortran	`call gpc_d3line(xf, yf, zf, xt, yt, zt)`

void d3head(char *title, int place); void d3pltfnt(float x, float y, float z, char *string, float height, int angle); D3PLT.C

Level 3

Typography in 3D coordinate space:

d3head(title, place): Places a caption string relative to the 3D workbox. d3pltfnt(x, y, z, string, height, angle): Plots text at user coordinate (x, y, z).

Multi-Language Signatures:

C	`void d3head(char *title, int place); void d3pltfnt(float x, float y, float z, char *string, float height, int angle);`
Python 3	`graphic.d3head(title: str, place: int) -> None; graphic.d3pltfnt(x: float, y: float, z: float, string: str, height: float, angle: int) -> None`
Java 22	`GraphiC.d3head(String title, int place); GraphiC.d3pltfnt(float x, float y, float z, String string, float height, int angle);`
Fortran	`call gpc_d3head(title, place); call gpc_d3pltfnt(x, y, z, string, height, angle)`

void sclfix(int isfixed) D3PLT.C

Level 1

int isfixed 1 = freeze 3D scale; 0 = auto-scale to maximize viewport (default).

Controls auto-scaling of the 3D workbox.

When isfixed=1, GraphiC maintains a fixed physical workbox scale across multiple frames even as viewing distance or viewing angles change.

Multi-Language Signatures:

C	`void sclfix(int isfixed)`
Python 3	`graphic.sclfix(isfixed: int) -> None`
Java 22	`GraphiC.sclfix(int isfixed)`
Fortran	`call gpc_sclfix(isfixed)`

void x3name(char *xname); void y3name(char *yname); void z3name(char *zname); — D3PLT.C

Level 1

char* name Axis title string.

Labels the X, Y, and Z axes of the 3D workbox.

The first character may be a stroke font selector code. Must be called before graf3d().

Multi-Language Signatures:

C	`void x3name(char *xname); void y3name(char *yname); void z3name(char *zname);`
Python 3	`graphic.x3name(xname: str) -> None; graphic.y3name(yname: str) -> None; graphic.z3name(zname: str) -> None`
Java 22	`GraphiC.x3name(String xname); GraphiC.y3name(String yname); GraphiC.z3name(String zname);`
Fortran	`call gpc_x3name(xname); call gpc_y3name(yname); call gpc_z3name(zname)`

void d3bars, plane3d, WaterfallPlot() — D3PLT.C

Level 2

Specialized 3D visualization routines.

d3bars(bwid): Configures surface to render perspective 3D column bars of width bwid.
plane3d(wplane, pos, hide): Draws a planar cross-section inside the 3D box at position pos.
WaterfallPlot(...): Renders a series of successive 2D spectral curves staggered along the Y axis.

Pie Charts & Statistical Plots

Generates proportional circular pie charts, normal probability error functions, and statistical distribution metrics.

void pieplot(int dim, float *seg, int *color, float *offset, char *caption[], float size, int border, int txtclr) — PIEPLT.C

Level 1

int dim	Number of pie sectors.
float* seg	Array of sector quantitative values.
int* color	Array of palette color indices for each sector.
float* offset	Radial displacement distance in inches for exploding slices (0.0 for normal).
char caption**	Array of string labels for each sector.
float size	Radius of the pie in physical inches (e.g. 2.0 in).
int border	Border stroke color index.
int txtclr	Sector text caption color index.

Renders a circular pie chart with optional sector explosion, hatching, and callouts.

Automatically computes percentage wedges from raw input values seg[i], draws radial dividers, applies distinct colors and vector fill patterns, applies radial displacement offsets (offset[i]) to explode highlighted slices, and centers text labels.

Multi-Language Signatures:

C	`void pieplot(int dim, float *seg, int *color, float *offset, char *caption[], float size, int border, int txtclr)`
Python 3	`graphic.pieplot(dim, seg, color, offset, caption, size, border, txtclr)`
Java 22	`GraphiC.pieplot(int dim, float[] seg, int[] color, float[] offset, String[] caption, float size, int border, int txtclr)`
Fortran	`call gpc_pieplot(dim, seg, color, offset, caption, size, border, txtclr)`

float mean(float *x, int xdim); float median(float *x, int xdim); float StdDev(float *x, int xdim); void BoxPlot(float x, float *y, int ydim, int outlier, int symnum, float width); — STATFUNS.C

Level 2

Statistical analysis and visualization suite.

mean(x, dim): Calculates arithmetic mean μ. median(x, dim): Calculates median (50th percentile). variance(x, dim): Computes sample variance s^2. StdDev(x, dim): Computes sample standard deviation s. BoxPlot(x, y, ydim, outlier, symnum, width): Draws a Tukey box plot at position x showing median line, interquartile range (IQR) box, 1.5 IQR whiskers, and plotted outlier dots.

Multi-Language Signatures:

C	`float mean(float *x, int xdim); float median(float *x, int xdim); float StdDev(float *x, int xdim); void BoxPlot(float x, float *y, int ydim, int outlier, int symnum, float width);`
Python 3	`graphic.mean(x, dim) -> float; graphic.BoxPlot(x, y, ydim, outlier, symnum, width)`
Java 22	`GraphiC.mean(float[] x, int xdim) -> float; GraphiC.BoxPlot(...)`
Fortran	`m = gpc_mean(x, dim); call gpc_BoxPlot(...)`

void erfscales(int nydiv, float *y, int npts); void erfcurve(float *x, float *y, int npts, int sym); void erfgrid(float ymin, float ystep, float ymax, char *yformat); — ERFPLT.C

Level 2

Error function and normal probability plotting.

erfscales(nydiv, y, npts): Automatically sets up Y axis calibrated according to the inverse error function $ ext{erf}^{-1}(y)$. erfcurve(x, y, npts, sym): Transforms Y coordinates into probability space and plots curve. erfgrid(...): Draws probability grid lines at standard percentile intervals (1%, 5%, 50%, 95%, 99%).

Multi-Language Signatures:

C	`void erfscales(int nydiv, float *y, int npts); void erfcurve(float *x, float *y, int npts, int sym); void erfgrid(float ymin, float ystep, float ymax, char *yformat);`
Python 3	`graphic.erfscales(nydiv, y, npts); graphic.erfcurve(x, y, npts, sym); graphic.erfgrid(ymin, ystep, ymax, yformat)`
Java 22	`GraphiC.erfscales(...); GraphiC.erfcurve(...); GraphiC.erfgrid(...)`
Fortran	`call gpc_erfscales(...); call gpc_erfcurve(...); call gpc_erfgrid(...)`

Line-Drawing Routines

Low-level vector graphics primitives for arbitrary lines, circles, ellipses, radial vectors, and customizable arrowheads.

void uvector(float xfrom, float yfrom, float xto, float yto, float ltow, float size, char *type); void vector(...); — ARROW.C

Level 1

float ltow	Arrowhead length-to-width aspect ratio (typically 2.0 to 3.0).
float size	Arrowhead length in inches (typically 0.1 to 0.25 in).
char* type	Arrowhead style string ("F", "O", "H").

Draws directed vector arrows with customizable head geometry.

uvector(): Coordinates specified in mathematical plot data units. vector(): Coordinates specified in physical inches on the page.

Renders shaft and arrowhead. Type specifies arrowhead style: 'F' = Filled solid arrowhead. 'O' = Open stroked arrowhead (V-shaped). 'H' = Hollow outline arrowhead.

Multi-Language Signatures:

C	`void uvector(float xfrom, float yfrom, float xto, float yto, float ltow, float size, char *type); void vector(...);`
Python 3	`graphic.uvector(xfrom, yfrom, xto, yto, ltow, size, type)`
Java 22	`GraphiC.uvector(float xfrom, float yfrom, float xto, float yto, float ltow, float size, String type)`
Fortran	`call gpc_uvector(xfrom, yfrom, xto, yto, ltow, size, type)`

void ellipse(float xc, float yc, float xr, float yr, float rotation, float th1, float th2, char degrees, int units); void fillellipse(int onoff, int pattern, int border);

CIRCLES.C

Level 2

float rotation	Tilt orientation angle of the ellipse major axis.
char degrees	1 = angles in degrees, 0 = angles in radians.
int units	0 = plot data units, 1 = physical inches.

Draws an ellipse or elliptical arc.

Allows arbitrary center location (xc, yc), semi-major and semi-minor radii (xr, yr), overall orientation tilt angle (rotation), and angular arc span [th1, th2]. fillellipse() sets fill pattern and border color for subsequent ellipse calls.

Multi-Language Signatures:

C	`void ellipse(float xc, float yc, float xr, float yr, float rotation, float th1, float th2, char degrees, int units); void fillellipse(int onoff, int pattern, int border);`
Python 3	`graphic.ellipse(xc, yc, xr, yr, rotation, th1, th2, degrees, units); graphic.fillellipse(onoff, pattern, border)`
Java 22	`GraphiC.ellipse(...); GraphiC.fillellipse(...)`
Fortran	`call gpc_ellipse(...); call gpc_fillellipse(...)`

Line and Symbol Styles

Renders line styles, dot-dash patterns, line thickening, marker symbols, and custom symbol scales.

void dashf(int type) LINEATTR.C

Level 1

int type Pattern index (0 to 8).

Sets stroke pattern for polylines and curves.

type: 0 = Solid line 1 = Short dash 2 = Long dash 3 = Dotted line 4 = Dash-dot 5 = Dash-dot-dot

Multi-Language Signatures:

C	`void dashf(int type)`
Python 3	`graphic.dashf(type: int) -> None`
Java 22	`GraphiC.dashf(int type)`
Fortran	`call gpc_dashf(type)`

void sympick(int symbol); void symbol(float x, float y, int symnum); SYMBOLS.C

Level 1

int symbol Marker symbol index (1 to 20).

float x X data coordinate.

float y Y data coordinate.

int symnum Marker symbol index.

Selects and renders centered vector marker symbols.

sympick(symbol): Selects active marker glyph: 1 = Circle, 2 = Triangle, 3 = Square, 4 = Diamond, 5 = Star, 6 = Cross (+), 7 = Saltire (X), etc. symbol(x, y, symnum): Draws symbol symnum centered directly at data coordinate (x, y).

Multi-Language Signatures:

C	`void sympick(int symbol); void symbol(float x, float y, int symnum);`
Python 3	`graphic.sympick(symbol: int); graphic.symbol(x: float, y: float, symnum: int)`
Java 22	`GraphiC.sympick(int symbol); GraphiC.symbol(float x, float y, int symnum);`
Fortran	`call gpc_sympick(symbol); call gpc_symbol(x, y, symnum)`

void tcurve(int width); void tline(int width); SVC.C

Level 2

int width Line stroke thickness (≥ 1).

Configures line stroke width.

tcurve(width): Sets curve line width in device units (1 = standard, 2 = bold, 3 = heavy). tline(width): Sets general line width for borders, boxes, and axes.

Multi-Language Signatures:

C	`void tcurve(int width); void tline(int width);`
Python 3	`graphic.tcurve(width: int); graphic.tline(width: int)`
Java 22	`GraphiC.tcurve(int width); GraphiC.tline(int width);`
Fortran	`call gpc_tcurve(width); call gpc_tline(width)`

Color & Palettes

High-precision conversions between RGB, HLS, CMYK, and indexed palette systems.

void color(int col); int RGBcolor(int r, int g, int b); LINEATTR.C

Level 1

int col Palette color index (0-255).

Configures the active drawing color for lines, axes, symbols, and text.

color(col): Selects standard color index (0=BLACK, 1=BLUE, 2=GREEN, 3=CYAN, 4=RED, 5=MAGENTA, 6=BROWN, 7=LIGHTGRAY, 15=WHITE). RGBcolor(r, g, b): Constructs 24-bit RGB color (components 0-255) and binds it.

Multi-Language Signatures:

C	`void color(int col); int RGBcolor(int r, int g, int b);`
Python 3	`graphic.color(col: int); graphic.RGBcolor(r: int, g: int, b: int) -> int`
Java 22	`GraphiC.color(int col); GraphiC.RGBcolor(int r, int g, int b) -> int`
Fortran	`call gpc_color(col)`

void RGBtoHLS(int r, int g, int b, int *h, int *l, int *s); void HLStoRGB(int h, int l, int s, int *r, int *g, int *b); void RGBtoCMYK(int r, int g, int b, int *c, int *m, int *y, int *k); int RGBtoIBM(int rgb[3]); COLOR.C

Level 3

Mathematical color space conversion routines.

RGBtoHLS(): Transforms RGB coordinates [0-255] into perceptual HLS space: Hue: 0-360 degrees, Lightness: 0-100%, Saturation: 0-100%.

HLStoRGB(): Transforms HLS parameters back to RGB primary channels. RGBtoCMYK(): Computes Cyan, Magenta, Yellow, and Key (black) printing ink percentages [0-100].

RGBtoIBM(): Maps 24-bit RGB values to the closest matching standard indexed palette color.

Multi-Language Signatures:

C	`void RGBtoHLS(int r, int g, int b, int *h, int *l, int *s); void HLStoRGB(int h, int l, int s, int *r, int *g, int *b); void RGBtoCMYK(int r, int g, int b, int *c, int *m, int *y, int *k); int RGBtoIBM(int rgb[3]);`
Python 3	`graphic.RGBtoHLS(r, g, b) -> (h, l, s); graphic.HLStoRGB(h, l, s) -> (r, g, b)`
Java 22	`GraphiC.RGBtoHLS(...); GraphiC.HLStoRGB(...)`
Fortran	`call gpc_RGBtoHLS(...); call gpc_HLStoRGB(...)`

Text Routines

TrueType (.ttf), PostScript Type 1 (.pfb), and Hershey vector stroke (.fnt) typography, multi-font inline switching, solid glyph filling, pairwise kerning, and mathematical script shifts.

void font(int nfont, ...); void pltfnt(float x, float y, char *string, float height, int angle); void prtfnt(float xinch, float yinch, char *string, float height, int angle);

TTFNTLD.C

Level 1

int nfont	Number of font pairs to load (1-10), or -1 to reload previous font set.
char* filename	Path to font file: .TTF (TrueType), .PFB (PostScript Type 1), or .FNT (Hershey stroke).
int esc_sym	Trigger character code (e.g. '\\', '|', '/') to switch to this font in mid-string.

Multi-engine typography and vector text rendering subsystem.

font(nfont, file1, esc1, ...): Registers and loads up to 10 fonts into memory simultaneously. File extensions automatically select the rendering engine:
- .TTF: Microsoft/Apple TrueType font with Quadratic B-spline outline evaluation, pairwise kerning, native UTF-8 sequence decoding, and even-odd counter hollowing.
- .PFB: Adobe PostScript Type 1 font with automated eexec decryption and Cubic Bézier CharStrings.
- .FNT: Digitized Hershey vector stroke font (simplex, complex, duplex, triplex, script, italics, Greek).

Call font(-1) to reload the active font set, or freememq() to release font memory buffers.

Companion Typography Controls:
fillfont(fill): 0 = hollow vector outline; 1 = solid polygon glyph fill (default for TrueType/PostScript).
fontfill(fill, pat, bdr): Fills characters with hatch pattern pat and draws outline in color bdr.
tifont(wid) / tfont(wid): Configures stroke pen thickness in physical inches (float) or raster pixels (int).
widen(factor) / skew(angle): Horizontal aspect ratio scaling and italic/oblique slanting angle.
SetTTKerning(onoff, str): Enables/disables TrueType pairwise kerning tables (kern SFNT table).
SetTTFull(onoff, str): 1 = full 256-character set (Windows ANSI/Latin-1); 0 = standard 128 ASCII.
SetTTMaxError(err, str): Configures Bézier curve subdivision tolerance for smoothness vs speed.
circtxt(x, y, r, string, height, angle): Renders text curved along a circular arc of radius r.
pltfnt(x, y, string, height, angle): Renders text at user data coordinates (x, y).
prtfnt(xinch, yinch, string, height, angle): Renders text at physical inches (xinch, yinch).

Multi-Language Signatures:

C	`void font(int nfont, ...); void pltfnt(float x, float y, char *string, float height, int angle); void prtfnt(float xinch, float yinch, char *string, float height, int angle);`
Python 3	`graphic.font(nfont); graphic.pltfnt(x, y, string, height, angle); graphic.prtfnt(xinch, yinch, string, height, angle)`
Java 22	`GraphiC.font(int nfont); GraphiC.pltfnt(float x, float y, String string, float height, int angle); GraphiC.prtfnt(...)`
Fortran	`call gpc_font(nfont); call gpc_pltfnt(x, y, string, height, angle)`

void ctline(char *string, float height); void ltline(char *string, float height); void rtline(char *string, float height); float strwidth(char *string, float height); FNTPLT.C

Level 2

char* string Input text string.

float height Character height in inches.

Text string alignment and metric measurement.

ctline(string, height): Draws a line of text horizontally centered at the top of the plot area.
ltline(string, height): Draws a line of text left-aligned with the plot box.
rtline(string, height): Draws a line of text right-aligned with the plot box.
strwidth(string, height): Calculates physical length of string in inches at height.
@return strwidth returns float width in inches.

Multi-Language Signatures:

C	`void ctline(char *string, float height); void ltline(char *string, float height); void rtline(char *string, float height); float strwidth(char *string, float height);`
Python 3	`graphic.ctline(string, height); graphic.strwidth(string, height) -> float`
Java 22	`GraphiC.ctline(String string, float height); GraphiC.strwidth(String string, float height) -> float`
Fortran	`call gpc_ctline(string, height); w = gpc_strwidth(string, height)`

void fntslant, fntthick, supsym, subsym() FNTPLT.C

Level 2

Fine typographical controls.

skew(slant): Applies forward italic slant angle to vector strokes.
widen(fact): Expands or condenses character horizontal aspect ratio.
supsym(ch): Defines the escape character that triggers superscript mode (default '^').
subsym(ch): Defines the escape character that triggers subscript mode (default '_').

Interactive Crosshairs & Digitization

Interactive full- screen cursor crosshairs with live digital coordinate readout and point digitization.

void curson(int srow, int scol, float **fx, float **fy, int *find) CURSORW.C

Level 3

int srow, scol	Position of digital readout on screen ($0, 0$ is top-left).
float fx, fy**	Pointers to coordinate arrays (allocated with initial size ≥ 64).
int* find	Output pointer receiving total points marked.

Enables interactive full-screen crosshair cursor and live digital position readout.

Displays a digital readout on the screen at text row srow, column scol in an 8x10 font. Crosshairs are manipulated via arrow keys or mouse pointer. The readout defaults to user data coordinates. Pressing <Alt-i> switches to physical inches; <Alt-u> returns to user data coordinates.

Marked points (<Enter>) are dynamically recorded into arrays fx and fy.

Interactive Key Controls:

<Enter> : Mark point at current crosshair location.
<Ins> : Draw line segment from previous point to cursor location.
<Del> : Erase last drawn segment.
<Alt-1>..<9> : Select line pattern for drawing.
<q> or <Esc> : Exit crosshairs and return to program execution.

Axis routines (graf, scales) must be called prior to curson().

Multi-Language Signatures:

C	`void curson(int srow, int scol, float **fx, float **fy, int *find)`
Python 3	`graphic.curson(srow: int, scol: int) -> tuple[list[float], list[float]]`
Java 22	`GraphiC.curson(int srow, int scol, float[][] fx, float[][] fy, int[] find)`
Fortran	`call gpc_curson(srow, scol, fx, fy, find)`

void curvefollow(int row, int col, float *x, float *y, int npts) CURSORW.C

Level 3

int npts Number of points on curve.

Constrains crosshairs to follow an existing discrete data curve (x[i], y[i]).

Arrow keys step forward and backward along the curve index. Digital readout displays exact sampled point values.

Multi-Language Signatures:

C	`void curvefollow(int row, int col, float *x, float *y, int npts)`
Python 3	`graphic.curvefollow(row, col, x, y, npts)`
Java 22	`GraphiC.curvefollow(int row, int col, float[] x, float[] y, int npts)`
Fortran	`call gpc_curvefollow(row, col, x, y, npts)`

Spline Interpolation & Numerical Methods

Cubic spline interpolation, curvature-constrained smoothing, non-linear regression fitting, and matrix solvers.

void spline(float *x, float *y, int n, int n1, float s1, float s2, float *xn, float *yn) — SMOOTH.C

Level 2

int n Number of input points.

int n1 Number of output interpolated points to generate.

Fits a continuous cubic spline curve through input points (x[i], y[i]).

Generates an interpolated smooth curve of n1 densely sampled points (xn, yn). Boundary conditions are controlled by end slope constraints s1 and s2.

Multi-Language Signatures:

C	`void spline(float *x, float *y, int n, int n1, float s1, float s2, float *xn, float *yn)`
Python 3	`graphic.spline(x, y, n, n1, s1, s2, xn, yn)`
Java 22	`GraphiC.spline(float[] x, float[] y, int n, int n1, float s1, float s2, float[] xn, float[] yn)`
Fortran	`call gpc_spline(x, y, n, n1, s1, s2, xn, yn)`

void rfit(float *x, float *y, int npts, int nits, float sfact, int nout, float *xout, float *yout) — RFIT.C

Level 2

int npts Number of input points.

int nits Number of reweighting iterations (e.g. 3-5).

float sfact Smoothing factor parameter.

int nout Number of fitted points to evaluate.

Performs iterative robust curve fitting with outlier attenuation.

Uses iteratively reweighted least squares to minimize the influence of experimental outliers and noisy measurement spikes.

Multi-Language Signatures:

C	`void rfit(float *x, float *y, int npts, int nits, float sfact, int nout, float *xout, float *yout)`
Python 3	`graphic.rfit(x, y, npts, nits, sfact, nout, xout, yout)`
Java 22	`GraphiC.rfit(...)`
Fortran	`call gpc_rfit(...)`

void matsolve(double **a, int n, double *b); void matinverse(double **a, double **y, int n); MATSOLVE.C

Level 3

double a**	2D matrix of coefficients [n][n].
int n	Dimension of system.
double* b	Right-hand side vector (returns solution x).
double y**	Output matrix receiving inverse A^-1.

High-performance linear equation solver and matrix inversion using Gaussian elimination with partial pivoting.

matsolve(a, n, b): Solves A * x = b. Matrix a is factored in-place and solution vector is returned in b.
matinverse(a, y, n): Inverts square matrix a into matrix y.

Multi-Language Signatures:

C	`void matsolve(double **a, int n, double *b); void matinverse(double **a, double **y, int n);`
Python 3	`graphic.matsolve(a, n, b); graphic.matinverse(a, y, n)`
Java 22	`GraphiC.matsolve(double[][] a, int n, double[] b); GraphiC.matinverse(...)`
Fortran	`call gpc_matsolve(a, n, b); call gpc_matinverse(a, y, n)`

Polygons & Panel Filling

Draws filled, hatched, and stippled polygonal regions and panels with 16 vector fill patterns.

void panel(float *x, float *y, int nlines, int pcolor, int border); void panelinch(...); SVC.C

Level 1

float* x	Array of X polygon vertices.
float* y	Array of Y polygon vertices.
int nlines	Number of vertices in polygon.
int pcolor	Fill pattern index (0-15) or solid palette color.
int border	Border line color (or -1 for no border outline).

Renders filled 2D planar polygons.

panel(): Vertex coordinates given in user plot units. panelinch(): Vertex coordinates given in physical inches. curvfill(): Fills the area between two arbitrary data curves.

In the modern SVG driver, panel geometries are emitted as compound paths with fill-rule="evenodd", ensuring proper hollowing of inner holes and contours.

Multi-Language Signatures:

C	`void panel(float *x, float *y, int nlines, int pcolor, int border); void panelinch(...);`
Python 3	`graphic.panel(x, y, nlines, pcolor, border); graphic.panelinch(...)`
Java 22	`GraphiC.panel(float[] x, float[] y, int nlines, int pcolor, int border)`
Fortran	`call gpc_panel(x, y, nlines, pcolor, border)`

Memory, File I/O & Utility Routines

Coordinate conversions (user units to pixels/inches), memory auditing, file management, and terminal control.

void usertopix(float x_in, float y_in, float z_in, int *x_out, int *y_out); void pixtouser(int x_in, int y_in, float *x_out, float *y_out); void usertoinch(float x_in, float y_in, float *x_out, float *y_out); float pixtoin(int pixels); int intopix(float inches); SVC.C

Level 3

int pixels Input pixel count.

float inches Input length in inches (or cm).

Coordinate space translation pipeline.

usertopix(x_in, y_in, z_in, *x_out, *y_out): Transforms mathematical data coordinate (x, y, z) into internal device pixel raster coordinates (0 to 8191 upright space). In 16-bit DOS GraphiC, device space spanned 4096 x 3072 units; modern GraphiC operates in 8192 x 6144 high-resolution space (MAXRESX = 8192, MAXRESY = 6144). For polar plots, x_in is radial distance r and y_in is angle theta (degrees); returned coordinates are Cartesian (x, y) device units. If plot rotation is active (rotate_flgq & 1), returned x increases downwards and y increases to the right on screen, calibrated so pixel coordinates assume an upright plot.

pixtouser(x_in, y_in, *x_out, *y_out): Inverts device pixel coordinates back into analytical data units, honoring linear, log10, ln, or polar mappings.

usertoinch(x_in, y_in, *x_out, *y_out): Converts mathematical data coordinate into physical canvas inches (or cm if metricunits(1) is active) measured from plot origin.

pixtoin(pixels): Converts device pixel quantity into physical inches (or cm). intopix(inches): Converts physical inches (or cm) into device pixel quantity.

Multi-Language Signatures:

C	`void usertopix(float x_in, float y_in, float z_in, int *x_out, int *y_out); void pixtouser(int x_in, int y_in, float *x_out, float *y_out); void usertoinch(float x_in, float y_in, float *x_out, float *y_out); float pixtoin(int pixels); int intopix(float inches);`
Python 3	`graphic.usertopix(x, y, z) -> (px, py); graphic.pixtouser(px, py) -> (x, y); graphic.usertoinch(x, y) -> (ix, iy)`
Java 22	`GraphiC.usertopix(float x, float y, float z, int[] out); GraphiC.pixtouser(int px, int py, float[] out);`
Fortran	`call gpc_usertopix(x_in, y_in, z_in, x_out, y_out); call gpc_pixtouser(x_in, y_in, x_out, y_out);`

void minmax(float *minq, float *maxq, float *xy, int npts); ROUNDOFF.C

Level 3

float* minq	Pointer receiving the computed minimum value.
float* maxq	Pointer receiving the computed maximum value.
float* xy	Pointer to input array of floating-point values.
int npts	Number of points in array.

Computes minimum and maximum extrema across a data array.

Used internally by calibration routines such as graf(), loggraf(), and r3d_contour() to determine optimal axis limits, tick intervals, and contour elevation slices. Non-real IEEE-754 floats (NaN and Inf) are automatically detected and bypassed via IsNotRealFloat(). If an array contains zero valid real numbers, minq is assigned 0.0, maxq is assigned 1.0, and Warning 47 is dispatched.

Multi-Language Signatures:

C	`void minmax(float *minq, float *maxq, float *xy, int npts);`
Python 3	`min_val, max_val = graphic.minmax(data_array)`
Java 22	`GraphiC.minmax(float[] minOut, float[] maxOut, float[] xy, int npts);`
Fortran	`call gpc_minmax(minval, maxval, xy, npts)`

void settolerance(float tolerance); R3PLT.C

Level 2

float tolerance	Tolerance factor (default is dynamically scaled).

Configures the interpolation tolerance factor for random point plotting.

When plotting contours or surfaces from scattered, ungridded (x, y, z) observations via r3d_surface() or r3d_contour(), clustering or collinear points can trigger the error: "Too few data points. Try a coarser grid." Increasing tolerance expands the neighborhood acceptance threshold for triangular facet formation, eliminating the error. As tolerance increases, interpolating precision is smoothed across sparse regions.

Multi-Language Signatures:

C	`void settolerance(float tolerance);`
Python 3	`graphic.settolerance(tolerance)`
Java 22	`GraphiC.settolerance(float tolerance);`
Fortran	`call gpc_settolerance(tolerance)`

void blank(unsigned char attr); BAUX_PORTABLE.C

Level 3

unsigned char attr Display attribute byte (color/intensity).

Clears the text terminal screen and resets console attributes.

In legacy DOS environments, blank(attr) wrote attribute bytes directly to video RAM (0xB800 / 0xB000). In modern POSIX / 64-bit environments, blank() emits standard ANSI escape sequences (\033[2J\033[H) to reset modern terminal emulators cleanly.

Multi-Language Signatures:

C	`void blank(unsigned char attr);`
Python 3	`graphic.blank(attr)`
Java 22	`GraphiC.blank(byte attr);`
Fortran	`call gpc_blank(attr)`

FILE *gfopen(char *filename, char *mode, size_t bigbuff); int gfclose(FILE *stream); BAUX_PORTABLE.C

Level 3

char* filename Relative or absolute filename to open.

char* mode Standard C fopen file mode ("r", "rb", "w+b", etc.).

size_t bigbuff If > 0, allocates a 12 KB I/O buffer via setvbuf().

Intelligent file open routine with directory search fallback and buffer allocation.

gfopen() locates and opens GraphiC support files (Hershey vector fonts, TrueType fonts, color maps, message tables, and TKF files) using an automated multi-tier search hierarchy: 1. Current working directory of the calling process. 2. Path specified by the GPC or GPC_DIR environment variable. 3. Dynamic binary location discovery: Uses POSIX dladdr() or macOS _NSGetExecutablePath() to resolve the directory containing libgraphic.dylib / libgraphic.so, searching the adjacent ../share/graphic/ and ../fonts/ directories. 4. Standard system installations: /usr/local/share/graphic/, /opt/homebrew/share/graphic/.

If bigbuff is non-zero, gfopen() attaches a high-throughput 12 KB memory buffer (TKBUFFSIZE = 12288 bytes) via setvbuf() to minimize operating system kernel context switches during rapid sequential streaming.

@return [FILE*] Opened standard C file pointer, or NULL on failure.

Multi-Language Signatures:

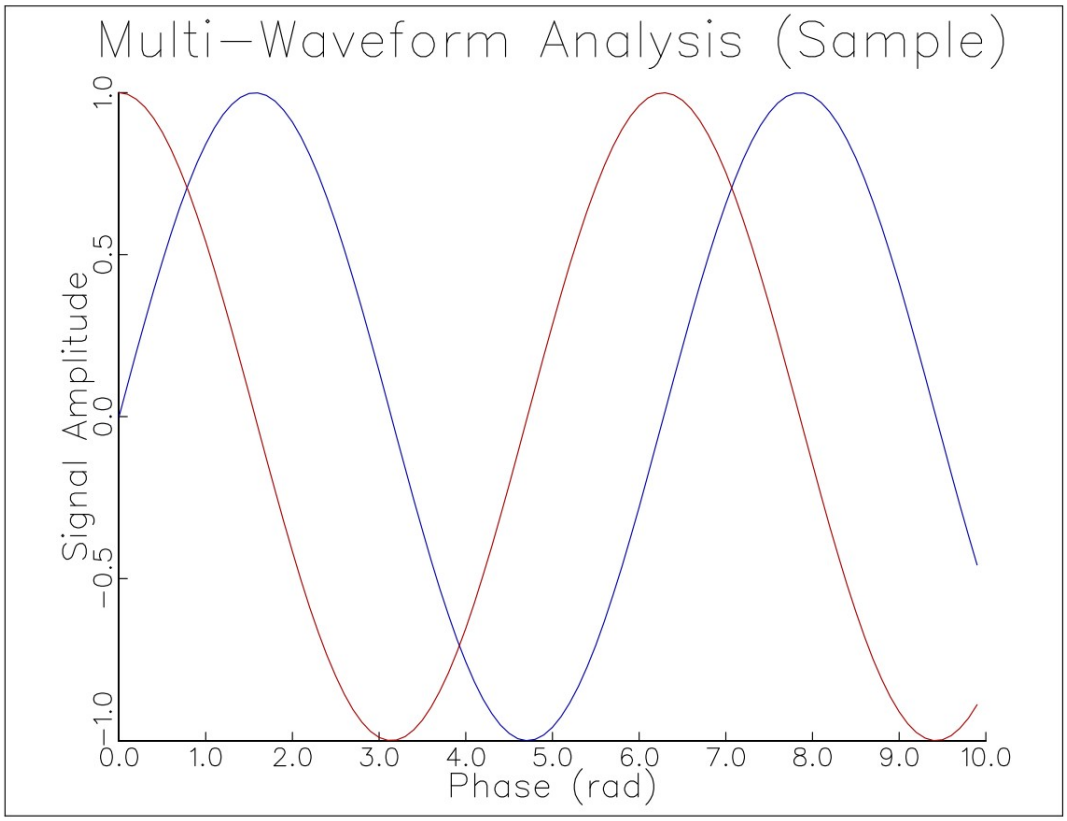
C	`FILE *gfopen(char *filename, char *mode, size_t bigbuff); int gfclose(FILE *stream);`
Python 3	`Managed automatically within native library core.`
Java 22	`Managed automatically within native library core.`
Fortran	`Managed automatically within native library core.`

9. Visual Gallery, Computational Recipes & Industrial Applications

This chapter presents the complete visual output, source code recipes, and real-world industrial applications of the GraphiC library across C, Python 3, Java 22, and Fortran.

9.1 2D Linear Calibration & Curve Plots (`sample`)

Linear calibrated axes with custom precision formats (`%3.1f`), grid tick marks, multi-curve plotting, and marker symbols. Following the authentic GraphiC / DISSPLA execution pipeline: initialize (`bgnplot`), set page units (`page`), begin frame (`startplot`), establish viewport area (`area2d`), format axes (`graf`), plot curves (`curve`), and terminate (`endplot`, `stopplot`).



2D Linear Multi-Curve Plot

C Implementation

```
#include "graphic.h"

int main() {
    bgnplot(0, 'S', "c/examples/d2.svg");
    page(8.0, 6.0);
    startplot(WHITE);
    area2d(5.5, 4.0);
    physor(1.5, 1.2);
    box();
    heading("Harmonic Oscillations");
    xname("Time (s)");
```

```
    yname("Voltage (V)");
    graf("%3.1f", 0.0, 1.0, 10.0, "%3.1f", -1.0, 0.5, 1.0);
    color(BLUE);
    curve(x1, y1, npts, 0);
    color(RED);
    curve(x2, y2, npts, 1);
    endplot();
    stopplot();
    return 0;
}
```

Fortran Implementation

```
program d2test
  use graphic
  implicit none
  integer, parameter :: NPTS = 50
  real :: x1(NPTS), y1(NPTS), x2(NPTS), y2(NPTS)
  ! ... populate data arrays ...

  call gpc_bgnplot(DRIVER_SVG, "fortran/examples/d2test.svg")
  call gpc_page(8.0, 6.0)
  call gpc_startplot(COLOR_WHITE)
  call gpc_area2d(5.5, 4.0)
  call gpc_physor(1.5, 1.2)
  call gpc_box()
  call gpc_heading("Harmonic Oscillations")
  call gpc_xname("Time (s)")
  call gpc_yname("Voltage (V)")
  call gpc_graf("%3.1f", 0.0, 1.0, 10.0, "%3.1f", -1.0, 0.5, 1.0)
  call gpc_color(COLOR_BLUE)
  call gpc_curve(x1, y1, NPTS, 0)
  call gpc_color(COLOR_RED)
  call gpc_curve(x2, y2, NPTS, 1)
  call gpc_endplot()
  call gpc_stopplot()
end program d2test
```

Python Implementation

```
import graphic as gpc

gpc.bgnplot("S", "python/examples/d2test.svg")
gpc.page(8.0, 6.0)
gpc.startplot(gpc.WHITE)
gpc.area2d(5.5, 4.0)
gpc.physor(1.5, 1.2)
gpc.box()
gpc.heading("Harmonic Oscillations")
gpc.xname("Time (s)")
gpc.yname("Voltage (V)")
gpc.graf("%3.1f", 0.0, 1.0, 10.0, "%3.1f", -1.0, 0.5, 1.0)
```

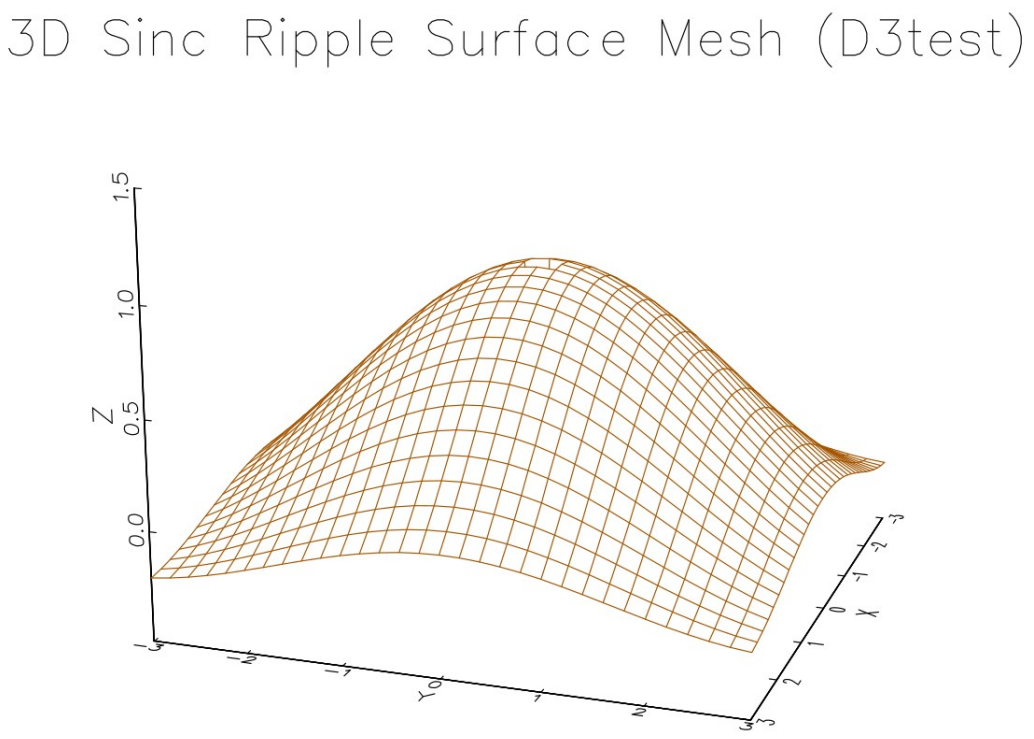
```
gpc.color(gpc.BLUE)
gpc.curve(x1, y1, len(x1), 0)
gpc.color(gpc.RED)
gpc.curve(x2, y2, len(x2), 1)
gpc.endplot()
gpc.stopplot()
```

Java Implementation

```
import com.sciend.graphic.GraphiC;
import com.sciend.graphic.Color;

public class D2Test {
    public static void main(String[] args) {
        GraphiC.bgnplot("S", "java/examples/d2test.svg");
        GraphiC.page(8.0f, 6.0f);
        GraphiC.startplot(Color.WHITE);
        GraphiC.area2d(5.5f, 4.0f);
        GraphiC.physor(1.5f, 1.2f);
        GraphiC.box();
        GraphiC.heading("Harmonic Oscillations");
        GraphiC.xname("Time (s)");
        GraphiC.yname("Voltage (V)");
        GraphiC.graf("%3.1f", 0.0f, 1.0f, 10.0f, "%3.1f", -1.0f, 0.5f, 1.0f);
        GraphiC.color(Color.BLUE);
        GraphiC.curve(x1, y1, npts, 0);
        GraphiC.color(Color.RED);
        GraphiC.curve(x2, y2, npts, 1);
        GraphiC.endplot();
        GraphiC.stopplot();
    }
}
```

9.2 3D Perspective Surface Mesh with Hidden-Line Removal (`d3test`)

Renders three-dimensional mathematical surfaces using GraphiC's analytical horizon-scanning hidden-line elimination algorithm (`surmat`).



3D Perspective Surface

C Implementation

```
#include "graphic.h"

int main() {
    bgnplot(0, 'S', "c/examples/d3.svg");
    page(8.0, 6.0);
    startplot(WHITE);
    volm3d(5.0, 5.0, 3.5);
    vuabs(10.0, -15.0, 12.0);
    graf3d(-5.0, 2.5, 5.0, -5.0, 2.5, 5.0, -1.0, 0.5, 1.0);
    survis("top");
    surmat((float*)zmatrix, 40, 40);
    endplot();
    stopplot();
    return 0;
}
```

Fortran Implementation

```
program d3test
  use graphic
  implicit none
  integer, parameter :: NX = 40, NY = 40
  real :: zmat(NX, NY)
  ! ... compute surface elevation array ...

  call gpc_bgnplot(DRIVER_SVG, "fortran/examples/d3test.svg")
  call gpc_page(8.0, 6.0)
  call gpc_startplot(COLOR_WHITE)
  call gpc_volm3d(5.0, 5.0, 3.5)
  call gpc_vuabs(10.0, -15.0, 12.0)
  call gpc_graf3d(-5.0, 2.5, 5.0, -5.0, 2.5, 5.0, -1.0, 0.5, 1.0)
  call gpc_surmat(zmat, NX, NY)
  call gpc_endplot()
  call gpc_stopplot()
end program d3test
```

Python Implementation

```
import graphic as gpc

gpc.bgnplot("S", "python/examples/d3test.svg")
gpc.page(8.0, 6.0)
gpc.startplot(gpc.WHITE)
gpc.volm3d(5.0, 5.0, 3.5)
gpc.vuabs(10.0, -15.0, 12.0)
gpc.graf3d(-5.0, 2.5, 5.0, -5.0, 2.5, 5.0, -1.0, 0.5, 1.0)
gpc.survis("top")
gpc.surmat(z_matrix, 40, 40)
gpc.endplot()
gpc.stopplot()
```

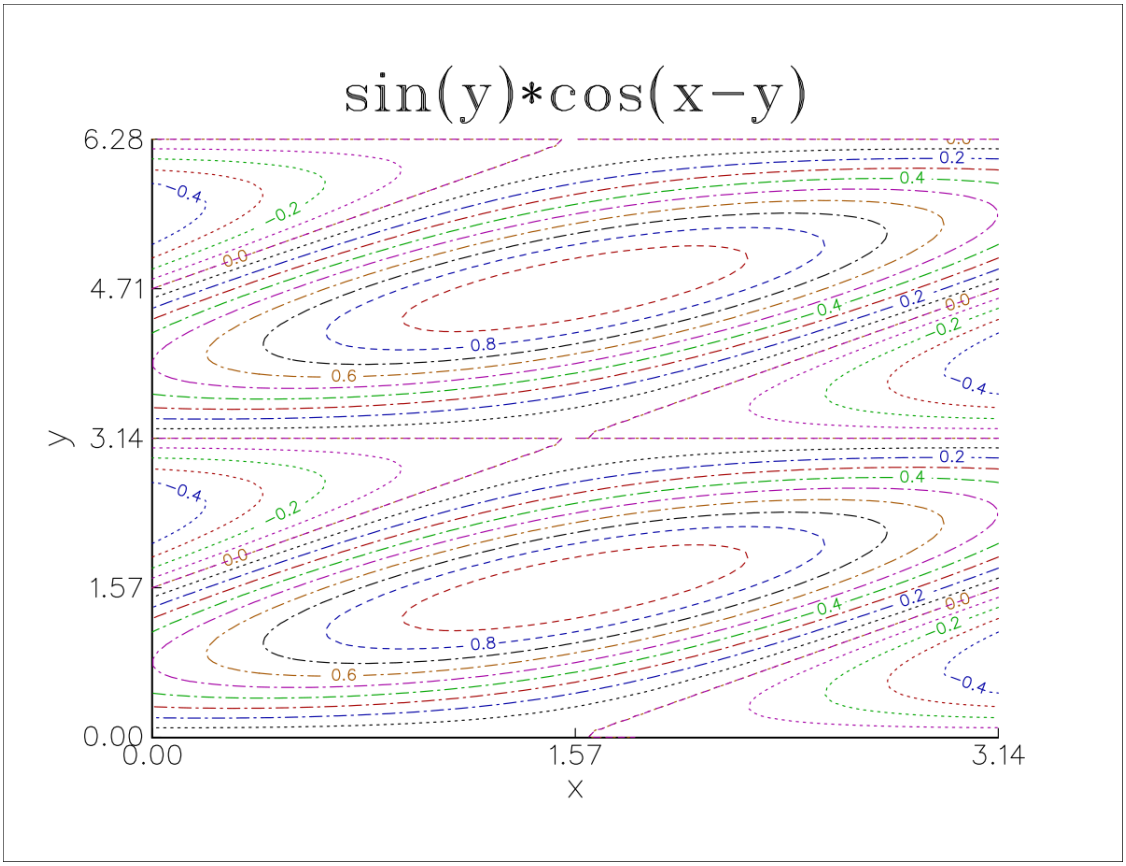
Java Implementation

```
import com.sciend.graphic.GraphiC;
import com.sciend.graphic.Color;

public class D3Test {
    public static void main(String[] args) {
        GraphiC.bgnplot("S", "java/examples/d3test.svg");
        GraphiC.page(8.0f, 6.0f);
        GraphiC.startplot(Color.WHITE);
        GraphiC.volm3d(5.0f, 5.0f, 3.5f);
        GraphiC.vuabs(10.0f, -15.0f, 12.0f);
        GraphiC.graf3d(-5.0f, 2.5f, 5.0f, -5.0f, 2.5f, 5.0f, -1.0f, 0.5f,
1.0f);
        GraphiC.survis("top");
        GraphiC.surmat(zFlat, 40, 40);
        GraphiC.endplot();
        GraphiC.stopplot();
```

```
    }
}
```

9.3 2D Contour Topography with Elevation Labels (`cnttest`)

Calculates iso-potential or elevation contour lines through a regular grid matrix and places inline elevation labels using the core `conmat` routine.



2D Contour Topography

C Implementation

```
#include "graphic.h"

int main() {
    bgnplot(0, 'S', "c/examples/cnttest.svg");
    page(8.0, 6.0);
    startplot(WHITE);
    area2d(5.0, 4.0);
    physor(1.5, 1.0);
    box();
    heading("2D Contour Topography");
    xname("X Coordinate");
    yname("Y Coordinate");
    graf("%3.1f", -3.0, 1.0, 3.0, "%3.1f", -3.0, 1.0, 3.0);
    color(BLACK);
    conmat((float*)zmatrix, xcoords, 30, ycoords, 30, -0.8, 0.2, 0.8, NULL,
```

```
1);
    endplot();
    stopplot();
    return 0;
}
```

Fortran Implementation

```
program cnttest
  use graphic
  implicit none
  integer, parameter :: NX = 30, NY = 30
  real :: zmat(NX, NY), xcoords(NX), ycoords(NY)
  ! ... generate grid and matrix ...

  call gpc_bgnplot(DRIVER_SVG, "fortran/examples/cnttest.svg")
  call gpc_page(8.0, 6.0)
  call gpc_startplot(COLOR_WHITE)
  call gpc_area2d(5.0, 4.0)
  call gpc_physor(1.5, 1.0)
  call gpc_box()
  call gpc_heading("2D Contour Topography")
  call gpc_xname("X Coordinate")
  call gpc_yname("Y Coordinate")
  call gpc_graf("%3.1f", -3.0, 1.0, 3.0, "%3.1f", -3.0, 1.0, 3.0)
  call gpc_color(COLOR_BLACK)
  call gpc_conmat(zmat, xcoords, NX, ycoords, NY, -0.8, 0.2, 0.8, 1)
  call gpc_endplot()
  call gpc_stopplot()
end program cnttest
```

Python Implementation

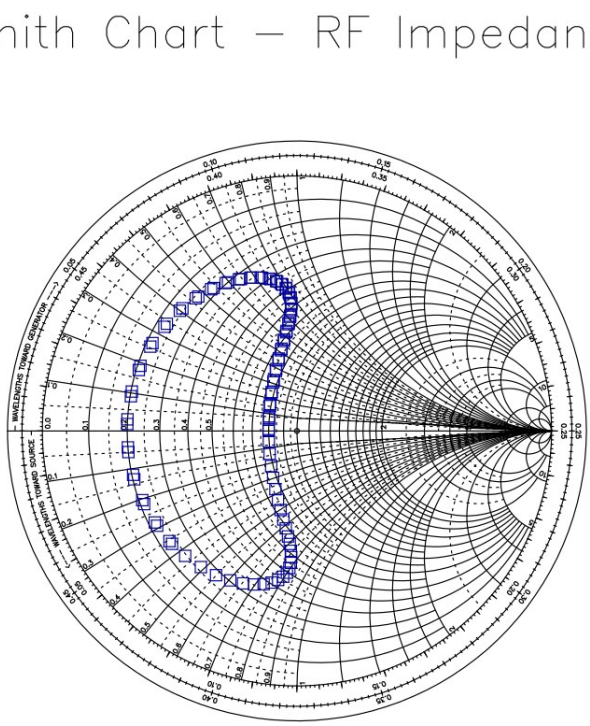
```
import graphic as gpc

gpc.bgnplot("S", "python/examples/cnttest.svg")
gpc.page(8.0, 6.0)
gpc.startplot(gpc.WHITE)
gpc.area2d(5.0, 4.0)
gpc.physor(1.5, 1.0)
gpc.box()
gpc.heading("2D Contour Topography")
gpc.xname("X Coordinate")
gpc.yname("Y Coordinate")
gpc.graf("%3.1f", -3.0, 1.0, 3.0, "%3.1f", -3.0, 1.0, 3.0)
gpc.color(gpc.BLACK)
gpc.conmat(z_matrix, x_coords, 30, y_coords, 30, -0.8, 0.2, 0.8, None, 1)
gpc.endplot()
gpc.stopplot()
```

Java Implementation

```
import com.sciend.graphic.GraphiC;
import com.sciend.graphic.Color;

public class CntTest {
    public static void main(String[] args) {
        GraphiC.bgnplot("S", "java/examples/cnttest.svg");
        GraphiC.page(8.0f, 6.0f);
        GraphiC.startplot(Color.WHITE);
        GraphiC.area2d(5.0f, 4.0f);
        GraphiC.physor(1.5f, 1.0f);
        GraphiC.box();
        GraphiC.heading("2D Contour Topography");
        GraphiC.xname("X Coordinate");
        GraphiC.yname("Y Coordinate");
        GraphiC.graf("%3.1f", -3.0f, 1.0f, 3.0f, "%3.1f", -3.0f, 1.0f, 3.0f);
        GraphiC.color(Color.BLACK);
        GraphiC.conmat(zFlat, xCoords, 30, yCoords, 30, -0.8f, 0.2f, 0.8f,
1);
        GraphiC.endplot();
        GraphiC.stopplot();
    }
}
```

9.4 RF Microwave Impedance Smith Chart (`smtest`)

Plots reflection coefficients, transmission line impedances, and admittance circles used internationally in radar and RF microwave engineering using `smdraw` and `plotrx`.



Smith Chart

C Implementation

```
#include "graphic.h"

int main() {
    bgnplot(0, 'S', "c/examples/smtest.svg");
    page(7.0, 7.0);
    startplot(WHITE);
    smdraw(BLACK);
    plotrx(0.5, 1.0);   // Normalized R = 0.5, X = +1.0
    plotrx(1.0, -0.5);  // Normalized R = 1.0, X = -0.5
    endplot();
    stopplot();
    return 0;
}
```

Fortran Implementation

```
program smtest
  use graphic
  implicit none

  call gpc_bgnplot(DRIVER_SVG, "fortran/examples/smtest.svg")
  call gpc_page(7.0, 7.0)
  call gpc_startplot(COLOR_WHITE)
  call gpc_smdraw(COLOR_BLACK)
  call gpc_plotrx(0.5, 1.0)
  call gpc_plotrx(1.0, -0.5)
  call gpc_endplot()
  call gpc_stopplot()
end program smtest
```

Python Implementation

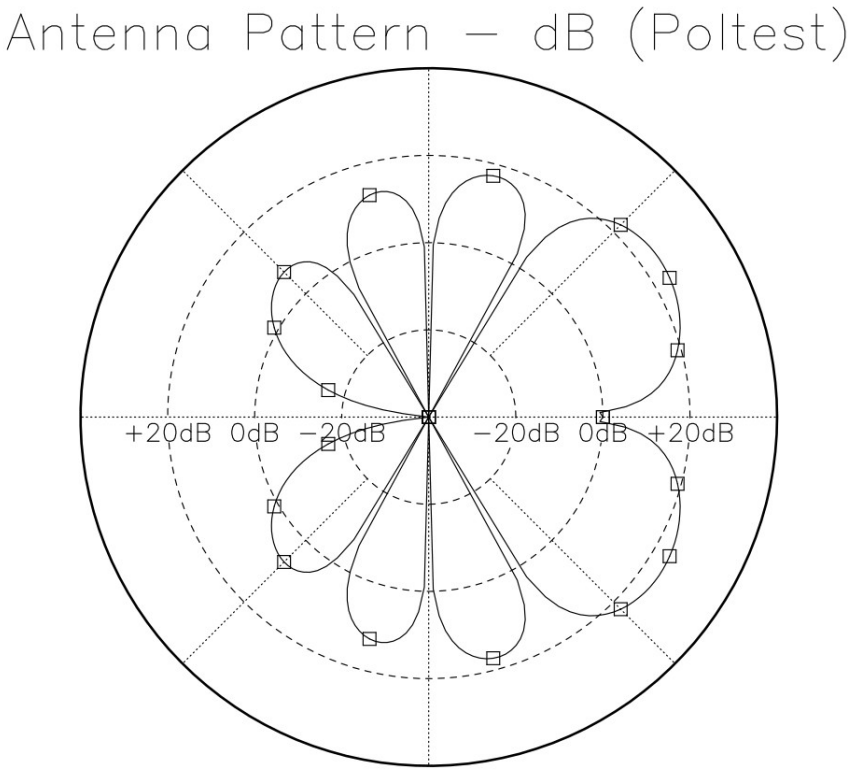
```
import graphic as gpc

gpc.bgnplot("S", "python/examples/smtest.svg")
gpc.page(7.0, 7.0)
gpc.startplot(gpc.WHITE)
gpc.smdraw(gpc.BLACK)
gpc.plotrx(0.5, 1.0)
gpc.plotrx(1.0, -0.5)
gpc.endplot()
gpc.stopplot()
```

Java Implementation

```
import com.sciend.graphic.GraphiC;
import com.sciend.graphic.Color;

public class SmTest {
    public static void main(String[] args) {
        GraphiC.bgnplot("S", "java/examples/smtest.svg");
        GraphiC.page(7.0f, 7.0f);
        GraphiC.startplot(Color.WHITE);
        GraphiC.smdraw(Color.BLACK);
        GraphiC.plotrx(0.5f, 1.0f);
        GraphiC.plotrx(1.0f, -0.5f);
        GraphiC.endplot();
        GraphiC.stopplot();
    }
}
```

9.5 Polar Antenna Radiation Patterns (`poltest`)

Plots circular directional gain profiles for electromagnetic radiation, acoustic transducers, and hydrophones using `polgrid` and `polcurve`.



Polar Antenna Plot

C Implementation

```
#include "graphic.h"

int main() {
    bgnplot(0, 'S', "c/examples/poltest.svg");
    page(7.0, 7.0);
    startplot(WHITE);
    area2d(5.0, 5.0);
    physor(1.0, 1.0);
    polgrid(0.0, 360.0, 30.0, 0.0, 1.0, 0.2, "%3.1f", BLACK);
    color(BLUE);
    polcurve(angles, radii, npts, 0);
    endplot();
    stopplot();
    return 0;
}
```

Fortran Implementation

```
program poltest
  use graphic
  implicit none
  integer, parameter :: NPTS = 360
  real :: angles(NPTS), radii(NPTS)
  ! ... compute antenna lobe radiation ...

  call gpc_bgnplot(DRIVER_SVG, "fortran/examples/poltest.svg")
  call gpc_page(7.0, 7.0)
  call gpc_startplot(COLOR_WHITE)
  call gpc_area2d(5.0, 5.0)
  call gpc_physor(1.0, 1.0)
  call gpc_polgrid(1.0, 0.0, COLOR_BLACK)
  call gpc_color(COLOR_BLUE)
  call gpc_polcurve(angles, radii, NPTS, 0)
  call gpc_endplot()
  call gpc_stopplot()
end program poltest
```

Python Implementation

```
import graphic as gpc

gpc.bgnplot("S", "python/examples/poltest.svg")
gpc.page(7.0, 7.0)
gpc.startplot(gpc.WHITE)
gpc.area2d(5.0, 5.0)
gpc.physor(1.0, 1.0)
gpc.polgrid(0.0, 360.0, 30.0, 0.0, 1.0, 0.2, "%3.1f", gpc.BLACK)
gpc.color(gpc.BLUE)
gpc.polcurve(angles, radii, len(angles), 0)
gpc.endplot()
gpc.stopplot()
```

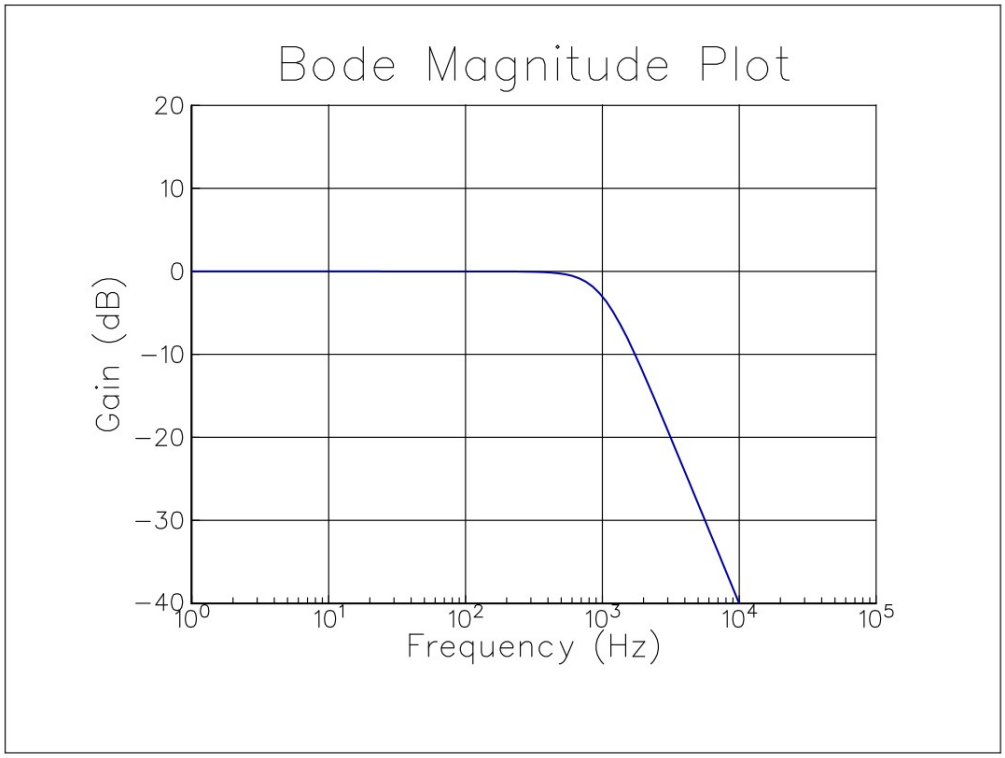
Java Implementation

```
import com.sciend.graphic.GraphiC;
import com.sciend.graphic.Color;

public class PolTest {
    public static void main(String[] args) {
        GraphiC.bgnplot("S", "java/examples/poltest.svg");
        GraphiC.page(7.0f, 7.0f);
        GraphiC.startplot(Color.WHITE);
        GraphiC.area2d(5.0f, 5.0f);
        GraphiC.physor(1.0f, 1.0f);
        GraphiC.polgrid(0.0f, 360.0f, 30.0f, 0.0f, 1.0f, 0.2f, "%3.1f",
Color.BLACK);
        GraphiC.color(Color.BLUE);
        GraphiC.polcurve(angles, radii, npts, 0);
        GraphiC.endplot();
```

```
        GraphiC.stopplot();
    }
}
```

9.6 Logarithmic & Semi-Log Plots (`logtest`)

Handles dynamic frequency sweeps, audio Bode plots, and wide-range spectroscopic measurements using `xlog` and `ylog`.



Logarithmic Frequency Plot

C Implementation

```
#include "graphic.h"

int main() {
    bgnplot(0, 'S', "c/examples/logtest.svg");
    page(8.0, 6.0);
    startplot(WHITE);
    area2d(5.5, 4.0);
    physor(1.5, 1.2);
    box();
    heading("Bode Magnitude Plot");
    xname("Frequency (Hz)");
    yname("Gain (dB)");
    xlog(1.0, 1.0e5, "%1.0e", -40.0, 10.0, 20.0);
    color(BLUE);
    curve(freq, gain, npts, 0);
```

```
    endplot();
    stopplot();
    return 0;
}
```

Fortran Implementation

```
program logtest
  use graphic
  implicit none
  integer, parameter :: NPTS = 100
  real :: freq(NPTS), gain(NPTS)
  ! ... compute frequency response ...

  call gpc_bgnplot(DRIVER_SVG, "fortran/examples/logtest.svg")
  call gpc_page(8.0, 6.0)
  call gpc_startplot(COLOR_WHITE)
  call gpc_area2d(5.5, 4.0)
  call gpc_physor(1.5, 1.2)
  call gpc_box()
  call gpc_heading("Bode Magnitude Plot")
  call gpc_xname("Frequency (Hz)")
  call gpc_yname("Gain (dB)")
  call gpc_xlog(1.0, 1.0e5, "%1.0e", -40.0, 10.0, 20.0)
  call gpc_color(COLOR_BLUE)
  call gpc_curve(freq, gain, NPTS, 0)
  call gpc_endplot()
  call gpc_stopplot()
end program logtest
```

Python Implementation

```
import graphic as gpc

gpc.bgnplot("S", "python/examples/logtest.svg")
gpc.page(8.0, 6.0)
gpc.startplot(gpc.WHITE)
gpc.area2d(5.5, 4.0)
gpc.physor(1.5, 1.2)
gpc.box()
gpc.heading("Bode Magnitude Plot")
gpc.xname("Frequency (Hz)")
gpc.yname("Gain (dB)")
gpc.xlog(1.0, 1.0e5, "%1.0e", -40.0, 10.0, 20.0)
gpc.color(gpc.BLUE)
gpc.curve(freq, gain, len(freq), 0)
gpc.endplot()
gpc.stopplot()
```

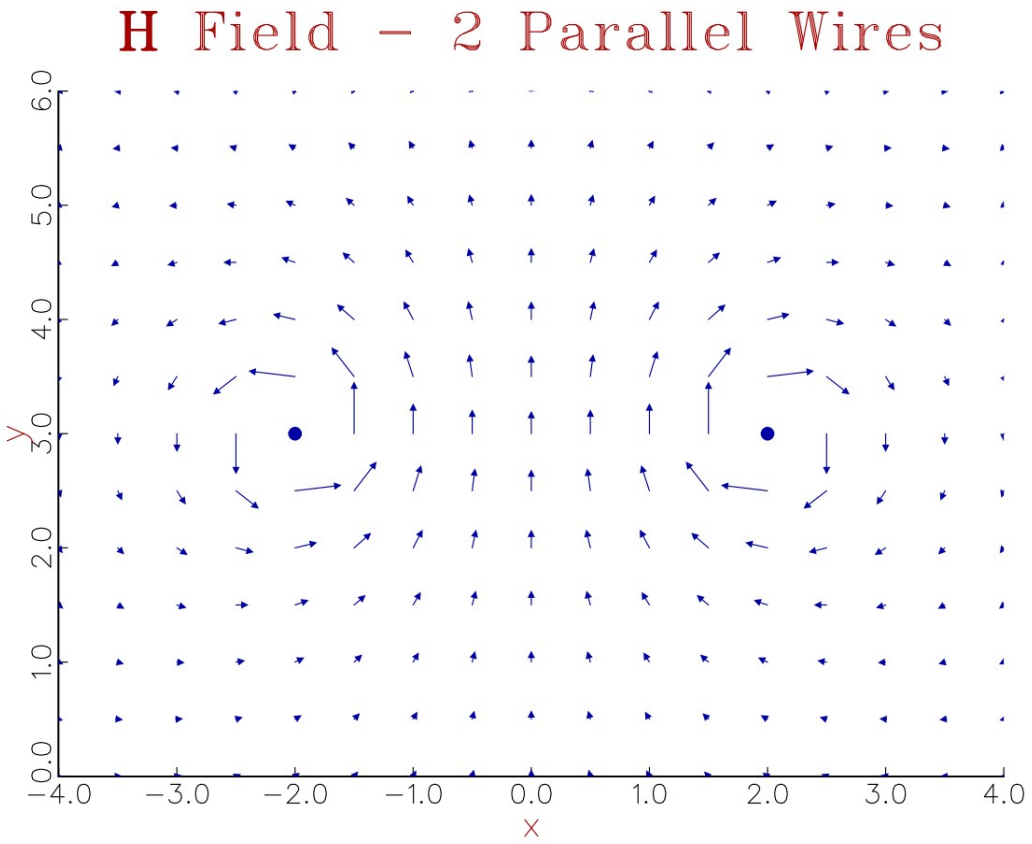
Java Implementation

```
import com.sciend.graphic.GraphiC;
import com.sciend.graphic.Color;
```

```
public class LogTest {
    public static void main(String[] args) {
        GraphiC.bgnplot("S", "java/examples/logtest.svg");
        GraphiC.page(8.0f, 6.0f);
        GraphiC.startplot(Color.WHITE);
        GraphiC.area2d(5.5f, 4.0f);
        GraphiC.physor(1.5f, 1.2f);
        GraphiC.box();
        GraphiC.heading("Bode Magnitude Plot");
        GraphiC.xname("Frequency (Hz)");
        GraphiC.yname("Gain (dB)");
        GraphiC.xlog(1.0f, 1.0e5f, "%1.0e", -40.0f, 10.0f, 20.0f);
        GraphiC.color(Color.BLUE);
        GraphiC.curve(freq, gain, npts, 0);
        GraphiC.endplot();
        GraphiC.stopplot();
    }
}
```

9.7 2D Velocity Vector Fields (`vfield`)

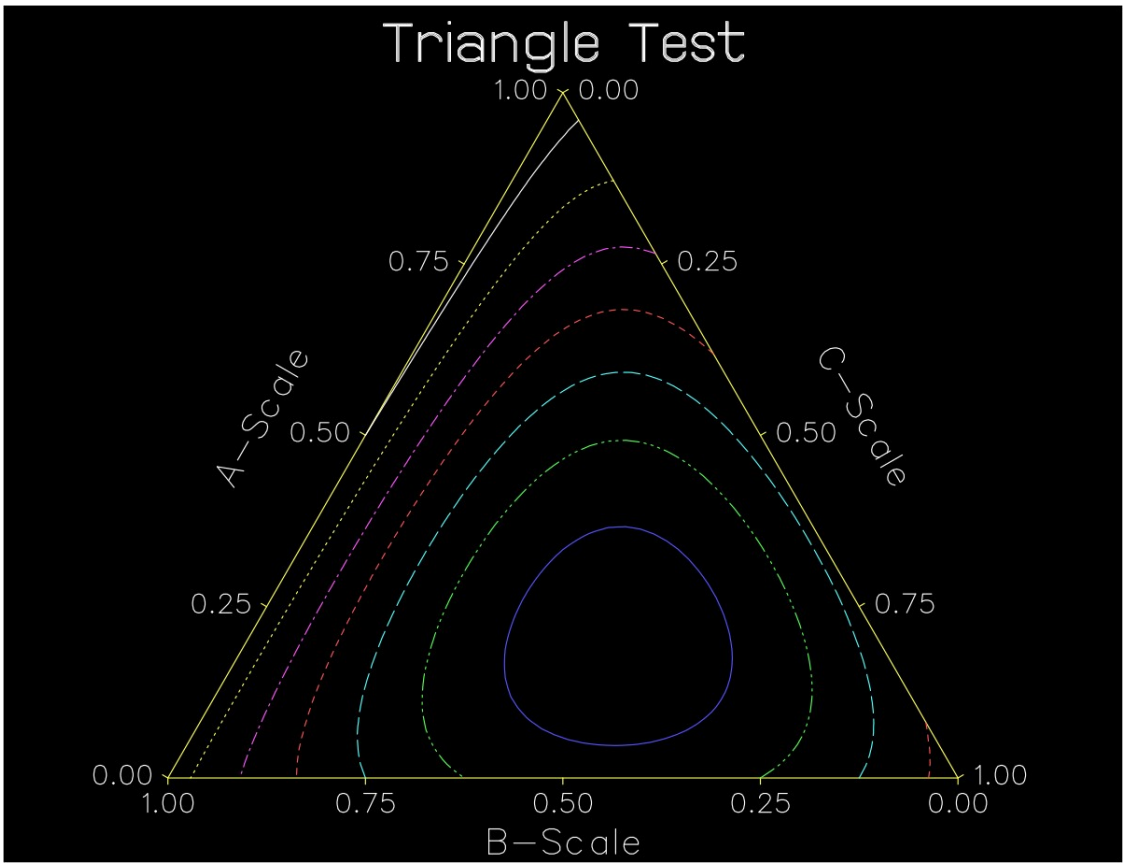
Visualizes fluid dynamics, wind velocities, and magnetic fields using directed vector arrows.



2D Velocity Field

9.8 Ternary Phase Equilibrium Diagrams (`tritest`)

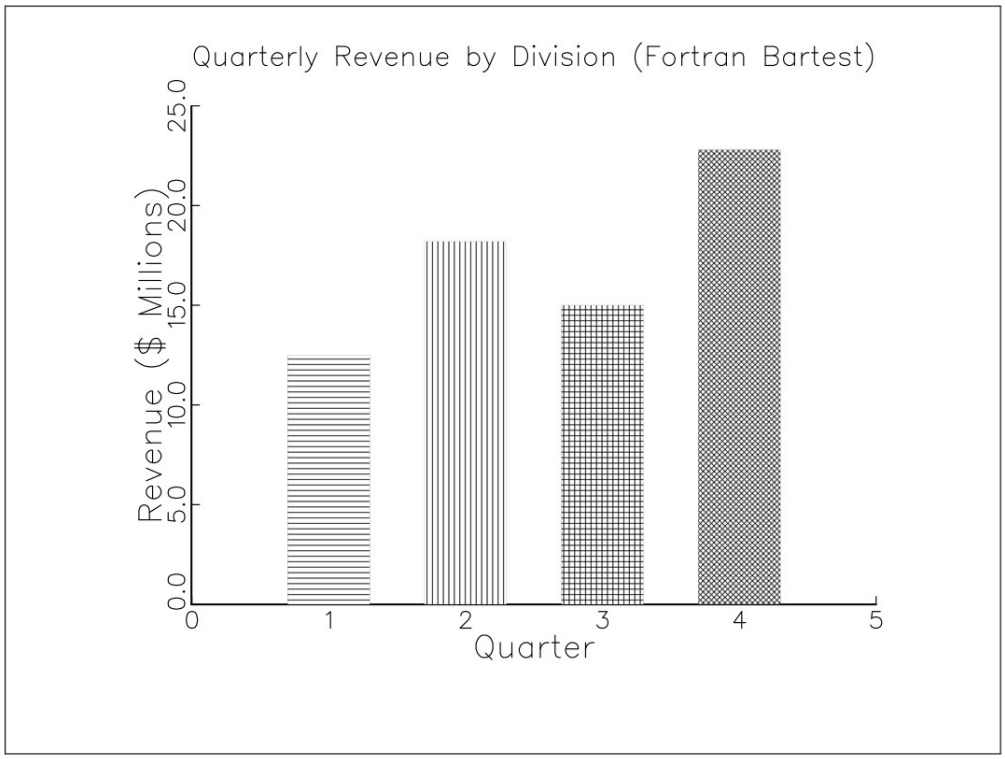
Specialized triangular coordinate framework for 3-component chemical mixtures, alloy metallurgy, and geology.



Ternary Phase Diagram

9.9 Grouped Bar Charts with Vector Hatching (`bartest`)

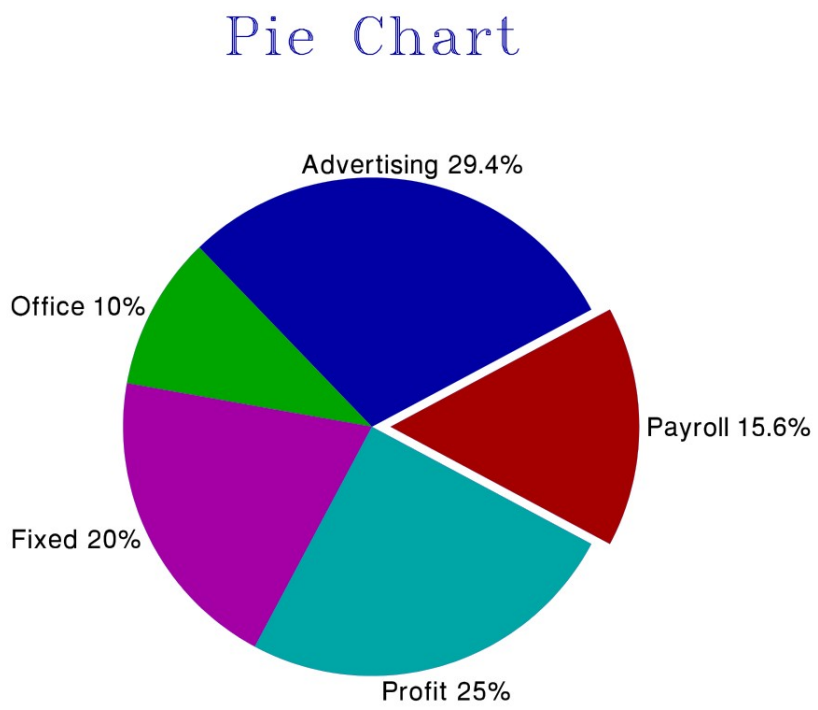
Scientific bar plots with customizable widths, groupings, and vector crosshatching.



Patterned Bar Chart

9.10 Exploded Pie Charts (`pietest`)

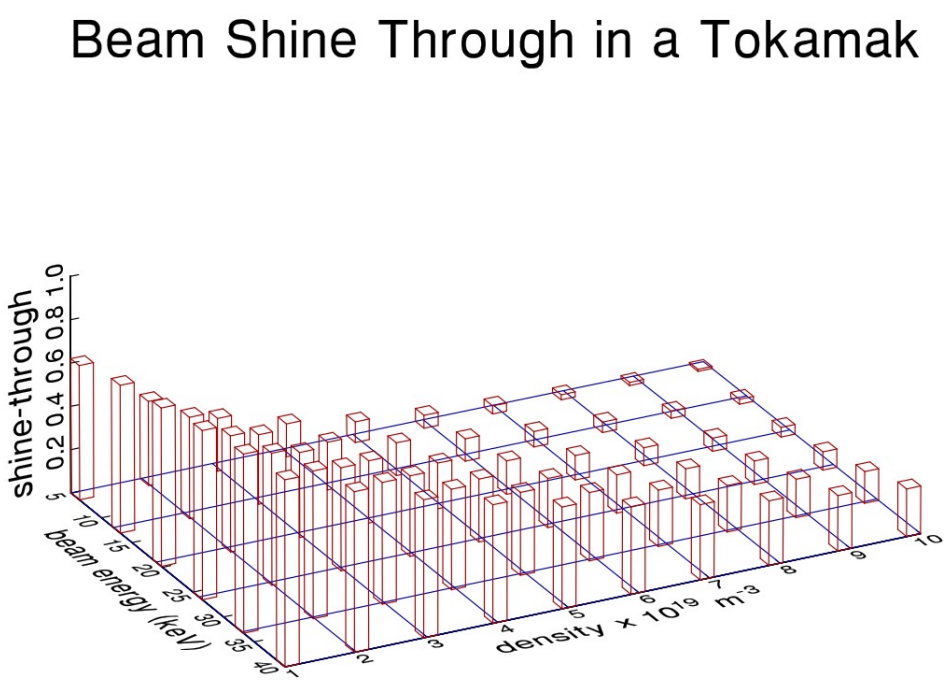
Proportional slice charts with independent offset distances for emphasis.



Exploded Pie Chart

9.11 3D Perspective Column Bars (`bar3d`)

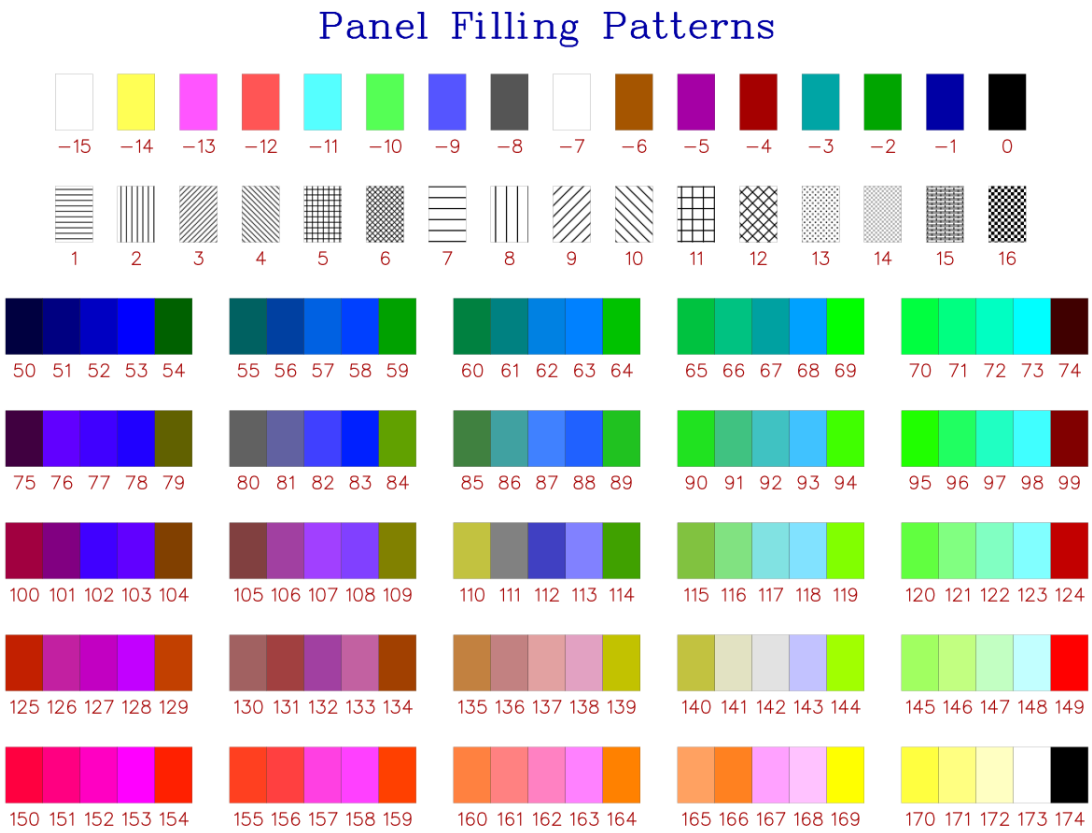
Three-dimensional perspective histogram bars with true spatial occlusion.



3D Perspective Bars

9.12 Vector Hatching & Stippling Patterns (`pattern`)

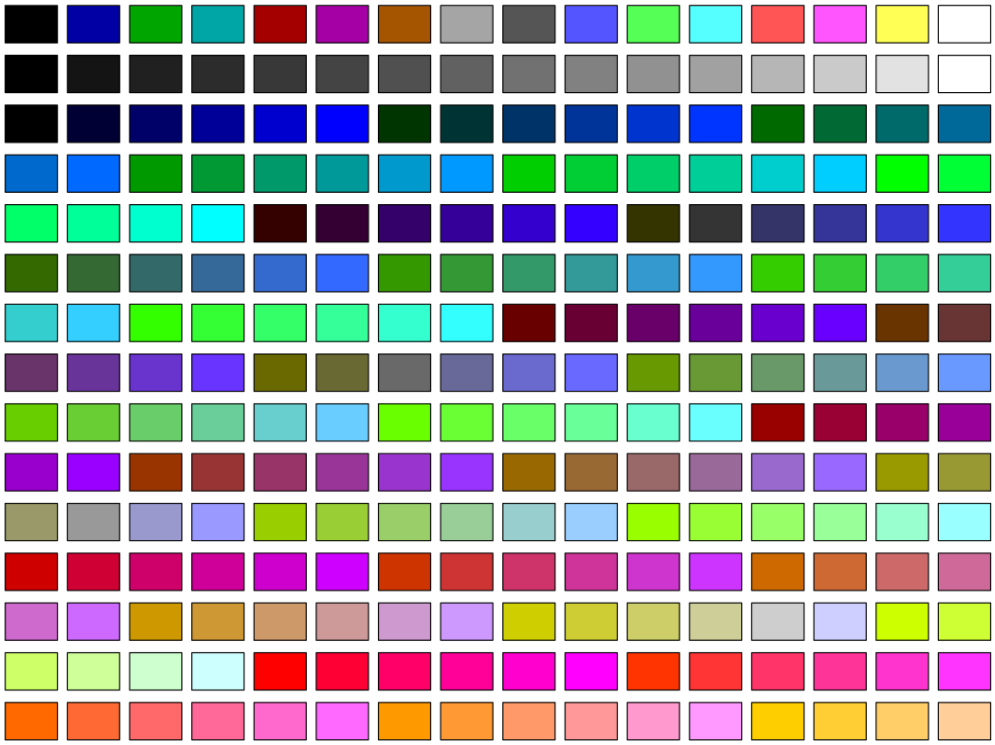
Vector fills for black-and-white publication printing, including diagonal crosshatch, brickwork, and stipple patterns.



16 Vector Patterns

9.13 256-Color Palette Grid (parn256)

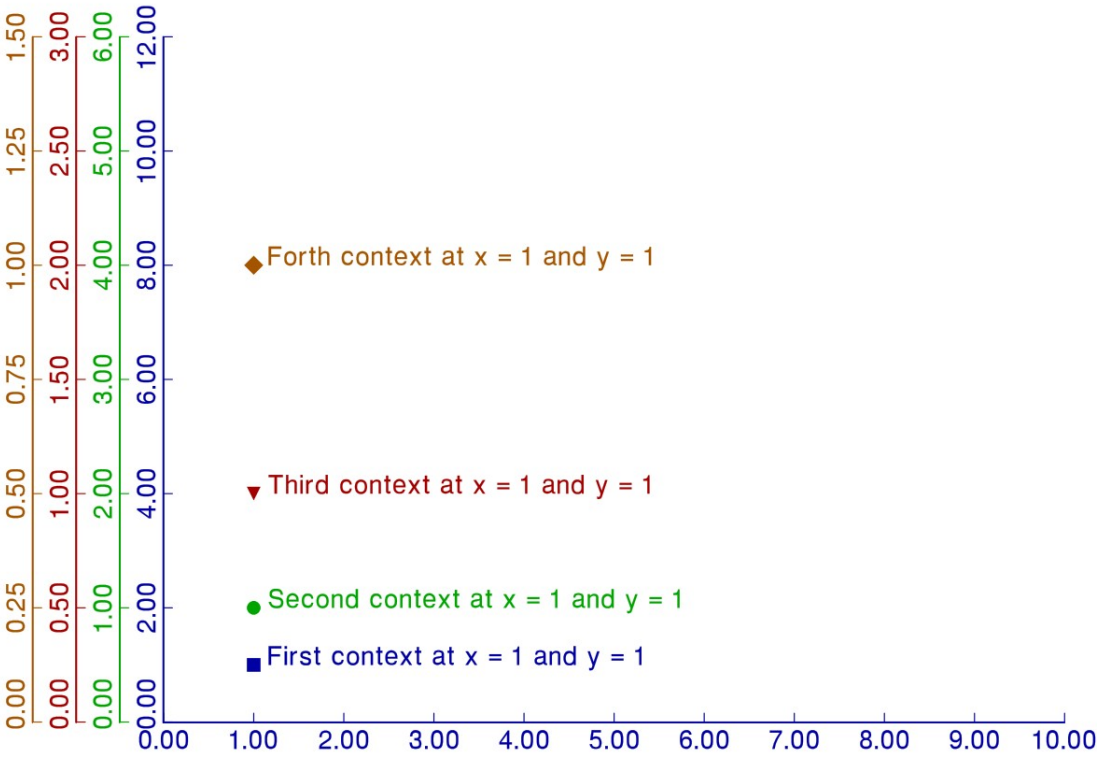
The full 256-color palette featuring a 16-step grayscale ramp and 216-color RGB color cube.



256 Color Palette

9.14 Multiple Independent Y-Axes (`multi_y`)

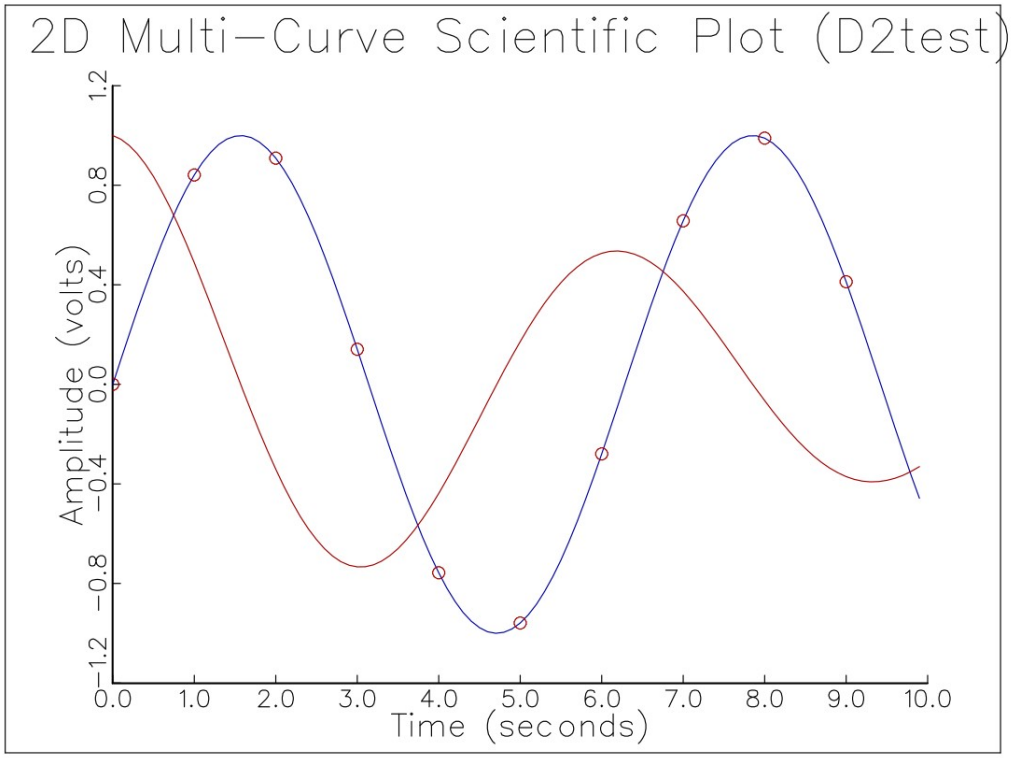
Simultaneous plotting of multiple physical quantities (e.g., pressure, temperature, flow rate) on distinct vertical scales.



Multiple Y Axes

9.15 Multi-Panel Composite Subplots (`d2test`)

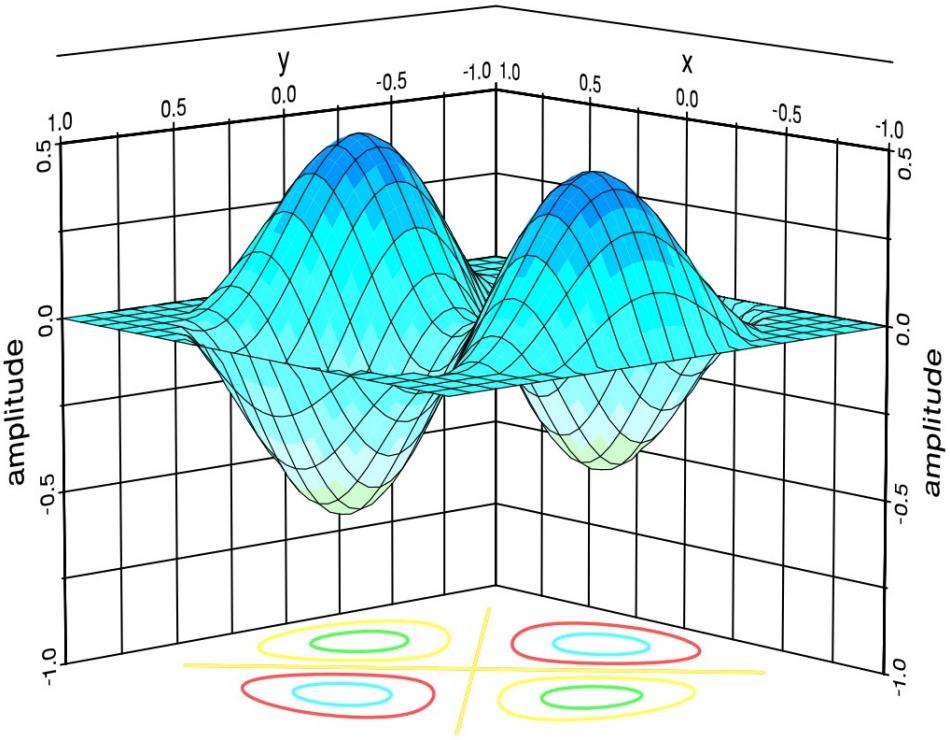
Multi-viewport layouts displaying distinct 2D, 3D, and polar plots in a single publication figure.



Multi-Panel Subplots

9.16 Cylindrical Bessel Functions (bess)

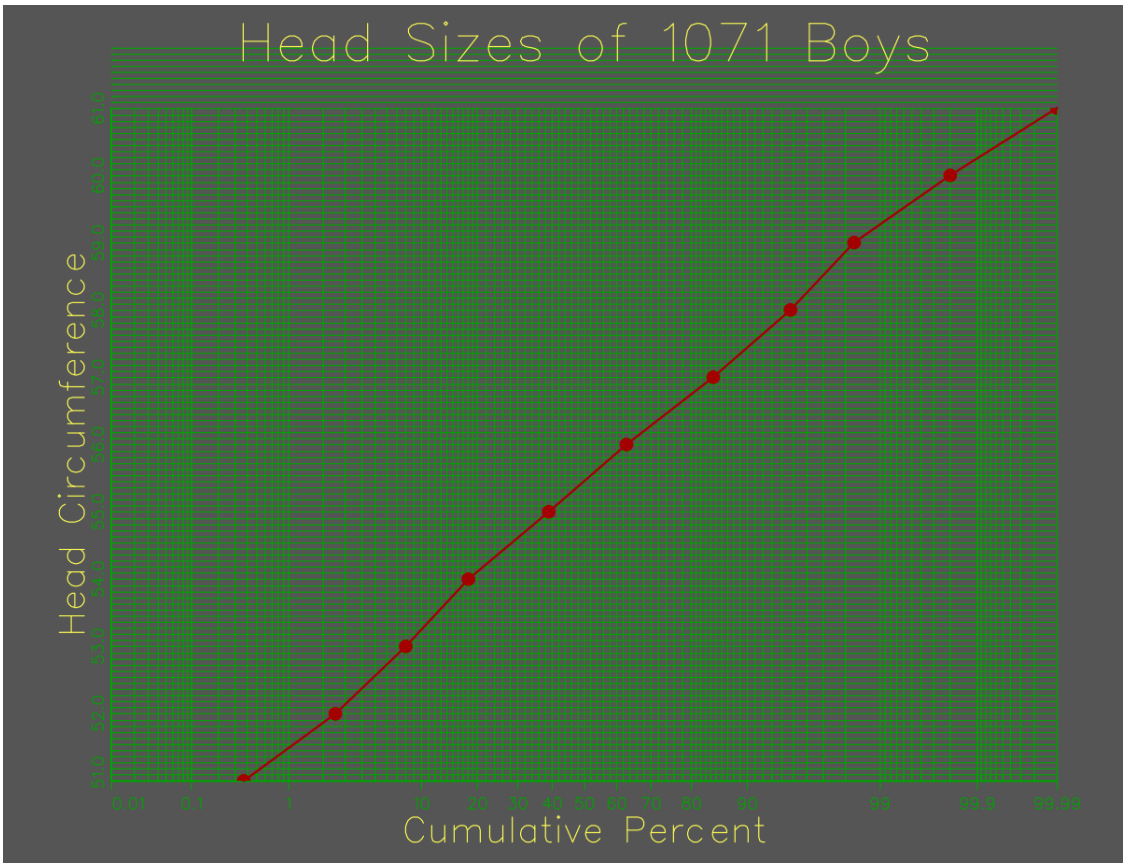
High-precision mathematical function visualization evaluating and plotting Bessel functions of the first kind (J_0, J_1, J_2, J_3) with customized line styles and scientific formatting.



Bessel Functions

9.17 Error Functions and Normal Distributions (`erftest`)

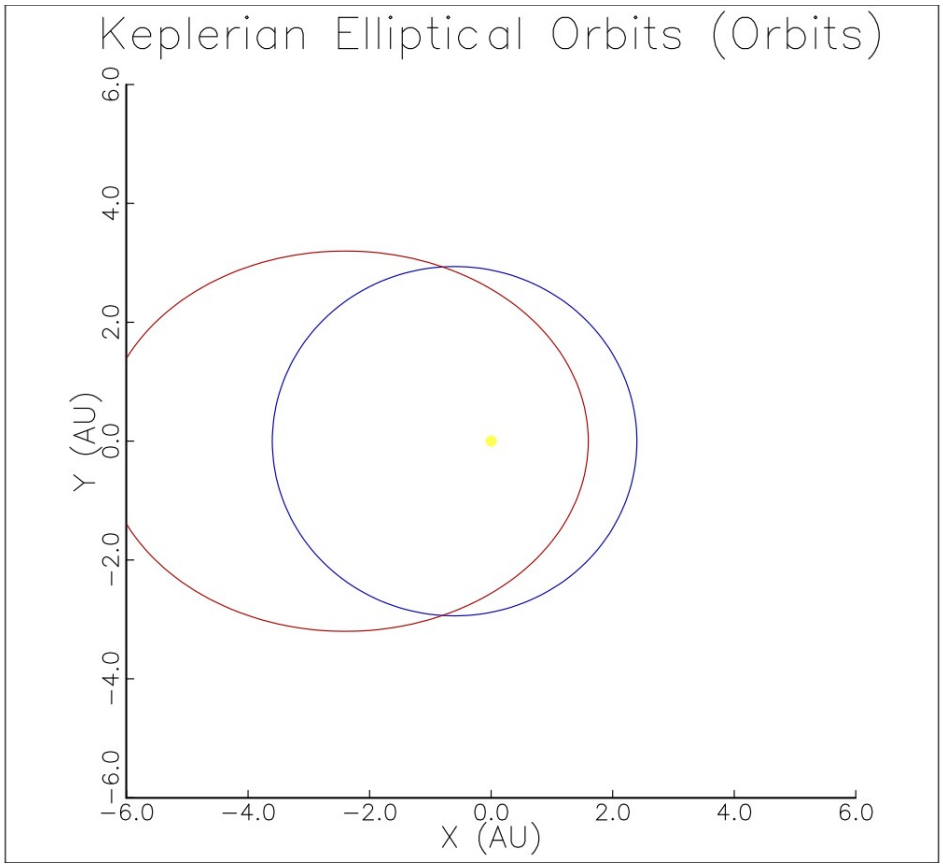
Visualizing the Gaussian error function (erf(x)) and complementary error function (erfc(x)) on specialized probability calibration scales.



Error Functions

9.18 Orbital Trajectories (`orbits`)

Numerical simulation of multi-body gravitational orbital mechanics plotting path in constants-of-motion space.

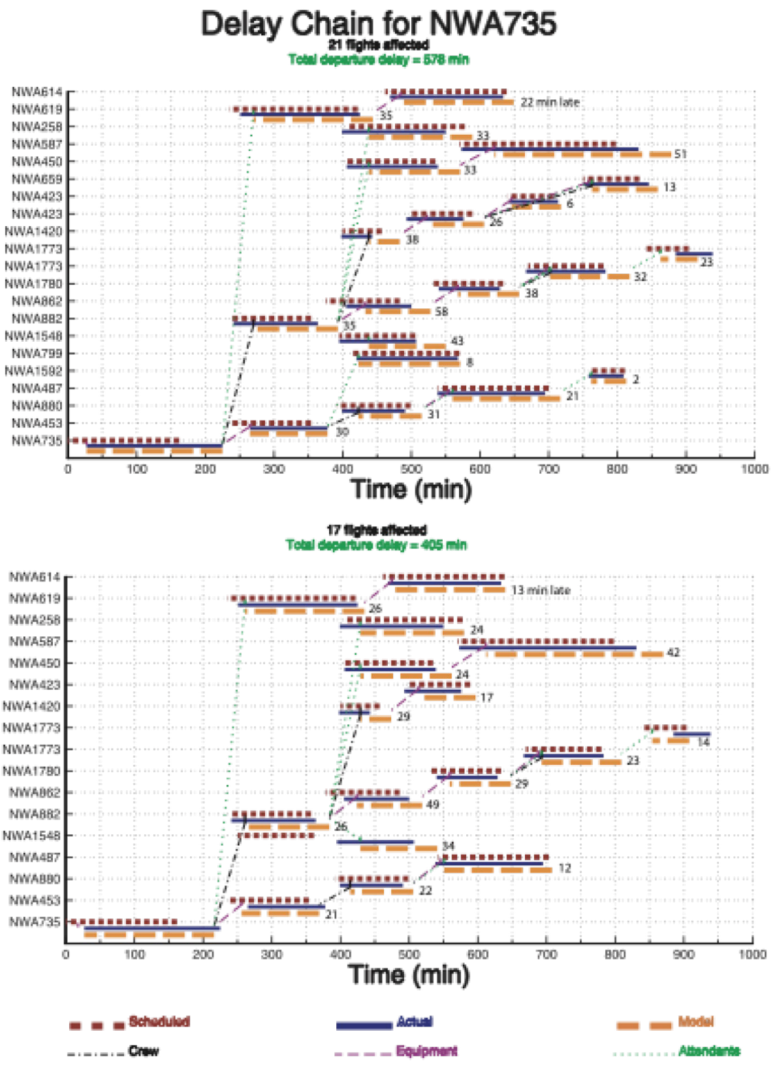


9.19 Industrial Applications: Airline Delay Propagation & Network Scheduling (NWA)

In real-world mission-critical operations, GraphiC has been deployed to model and visualize large-scale aerospace and transport systems. Below are seminal operational research figures produced with GraphiC for Northwest Airlines (NWA), and the Federal Aviation Administration (FAA), analyzing air traffic flow management, departure delay propagation down flight connection chains, and hub arrival banking at Minneapolis–Saint Paul International Airport (MSP).

Flight Delay Cascade Propagation

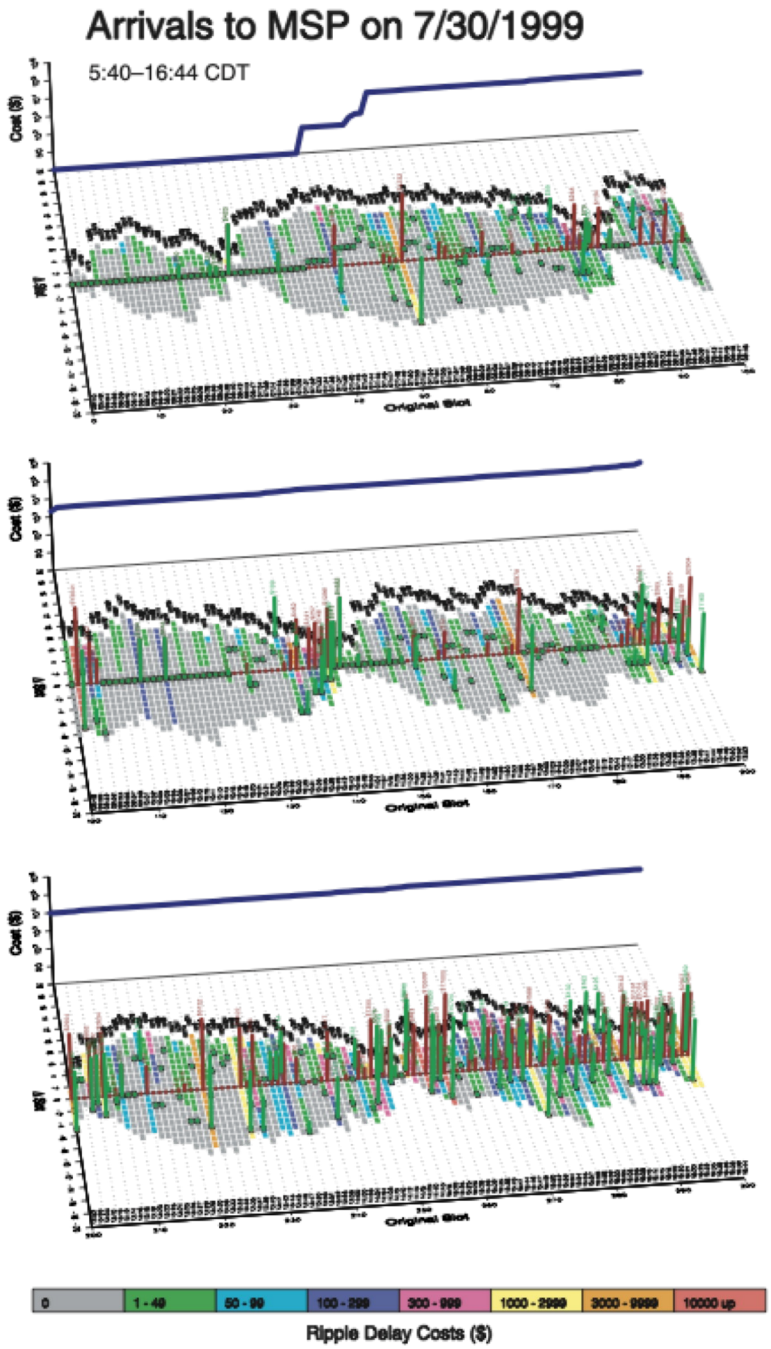
A multi-tier flight connection timeline tracking delay cascade across 17 downstream aircraft rotations. The chart distinguishes scheduled times, actual touchdown/takeoff times, modal block durations, equipment turnaround intervals, and upstream delay propagation.



Flight Delay Cascade Propagation (NWA 735 Chain)

Hub Arrival Bank Operations at Minneapolis–Saint Paul (MSP)

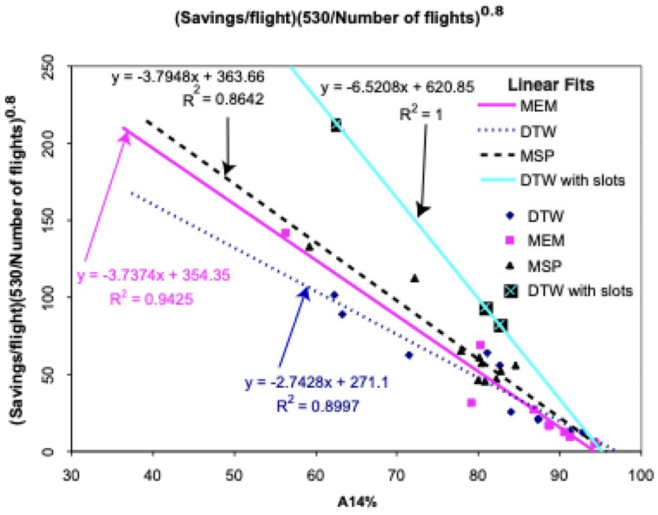
A high-density temporal schedule analysis showing scheduled arrival slots versus actual runway touchdown distributions during heavy traffic banks at MSP International Airport on July 30, 1999.



MSP Hub Arrival Bank Operations

Schedule Resequencing & Fleet Cost Optimization

Statistical modeling and non-linear power-law regression ($y = -3.7848\,x + 363.66$) demonstrating multi-million-dollar fleet cost reductions through dynamic flight resequencing and gate pushback optimization.



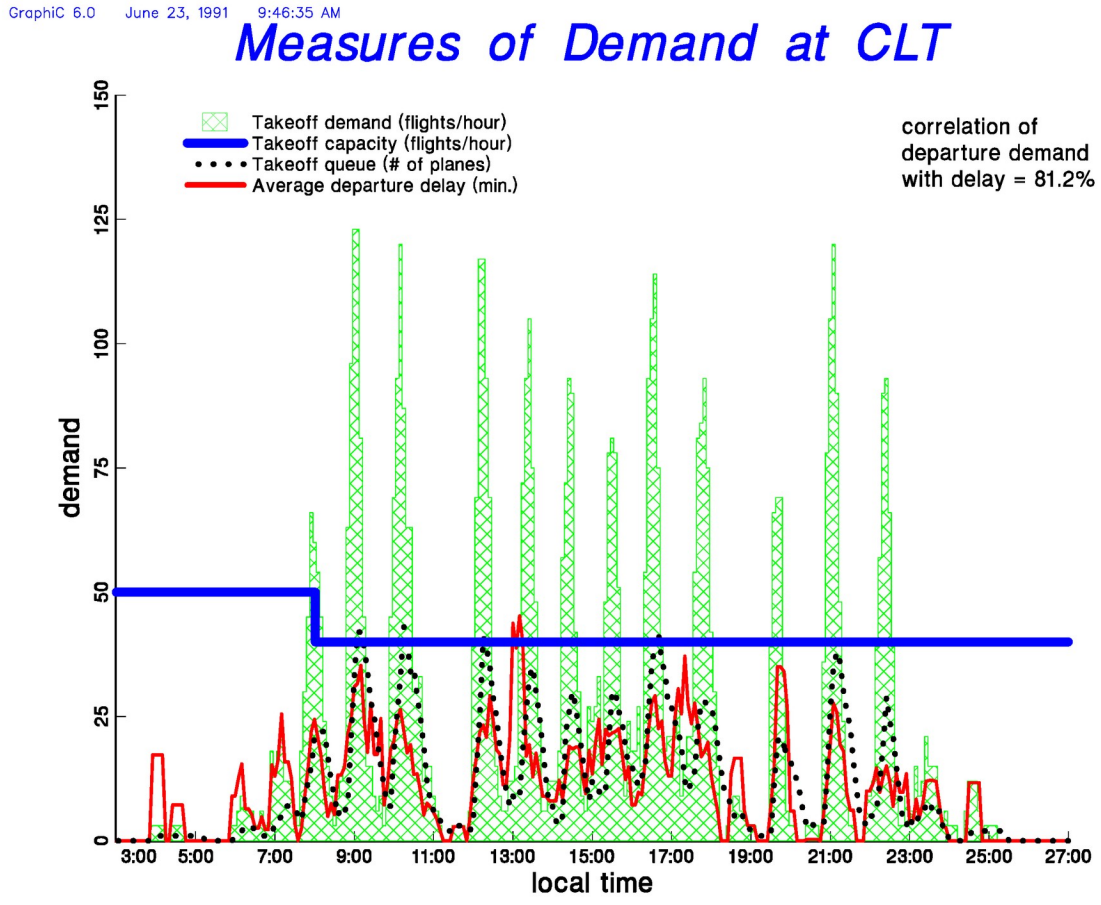
Fleet Resequencing Cost Savings Model

9.20 FAA Airport Runway Demand & Capacity Queuing Dynamics (CLT)

At major hub airports, flight scheduling in tightly clustered arrival and departure banks frequently exceeds the physical acceptance rates of runways, inducing long taxiway queues, gate holds, and ripple delays across the National Airspace System. Developed for the FAA and the Collaborative Decision Making (CDM) program, these visualizations model the exact queue buildup, runway capacity step limits, and demand-to-delay correlation at Charlotte Douglas International Airport (CLT).

Charlotte Douglas International Airport (CLT) Takeoff Demand, Capacity & Queuing Dynamics

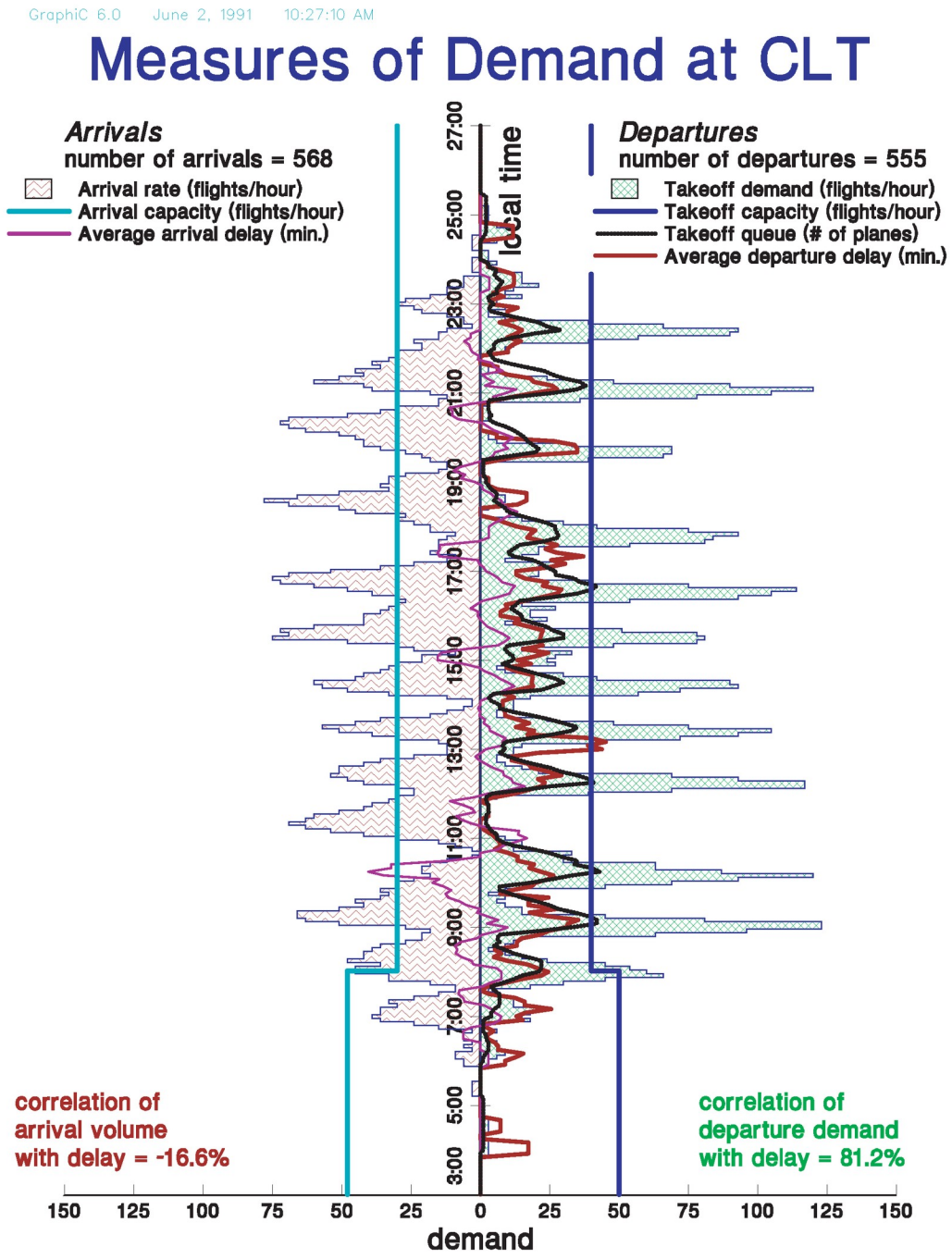
A temporal operational queuing analysis (`DEMAND.C`) tracking 24 hours of flight activity at CLT. The plot models the physical runway takeoff capacity step-function (heavy blue rule, 40–50 flights/hour), actual departure demand surges (green cross-hatched area), resulting takeoff queue length (heavy dotted black line), and average ground departure delay (solid red curve). The statistical cross-correlation between departure demand surges and departure delay is **81.2%**, demonstrating that ground delays are overwhelmingly driven by scheduled banking peaks exceeding runway capacity.



FAA CLT Departure Demand, Runway Capacity, and Delay Dynamics

Bidirectional Runway Demand vs. Capacity (Arrivals & Departures Butterfly Plot)

A dual-sided symmetrical time-distribution chart (`CLTDEMAND.eps`) comparing incoming arrivals against outgoing departures across the 24-hour operational day (03:00 to 27:00 local time). The central vertical axis displays local time, with arrival metrics mirrored on the left and departure metrics on the right: - **Left Wing (Arrivals)**: 568 total arrivals; arrival rate distribution (wavy hatch fill), declared arrival capacity limit (cyan boundary line), and average arrival delay (magenta curve). Correlation of arrival volume with delay is **-16.6%**. - **Right Wing (Departures)**: 555 total departures; takeoff demand (cross-hatch fill), declared takeoff capacity (blue boundary line), active takeoff queue (black stippled curve), and average departure delay (red curve). Correlation of departure demand with delay is **81.2%**.



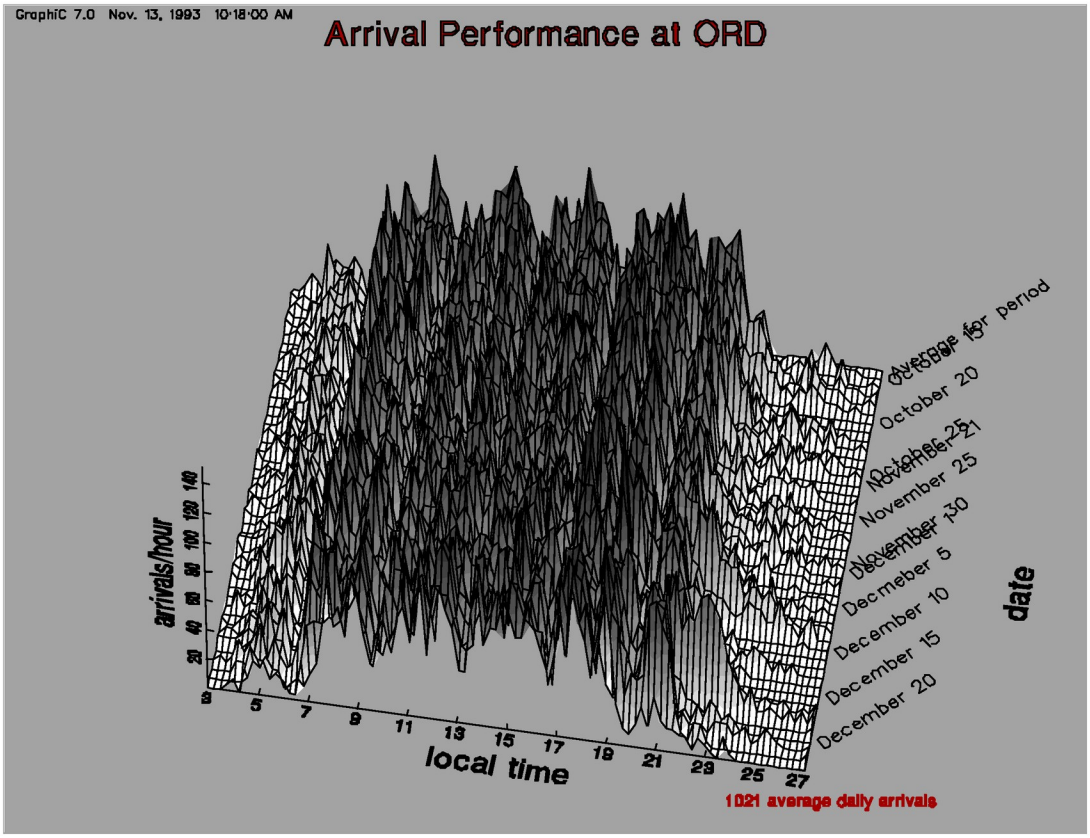
FAA CLT Airport Bidirectional Runway Demand vs. Capacity

9.21 FAA National Airspace System Flow & Airport Performance Gallery (ORD, EWR, ZOB)

GraphiC has provided the primary visualization engine for national aerospace infrastructure modeling across Federal Aviation Administration research centers, the Volpe National Transportation Systems Center, and Oak Ridge National Laboratory (ORNL).

Chicago O'Hare International Airport (ORD) Arrival Surface Topography

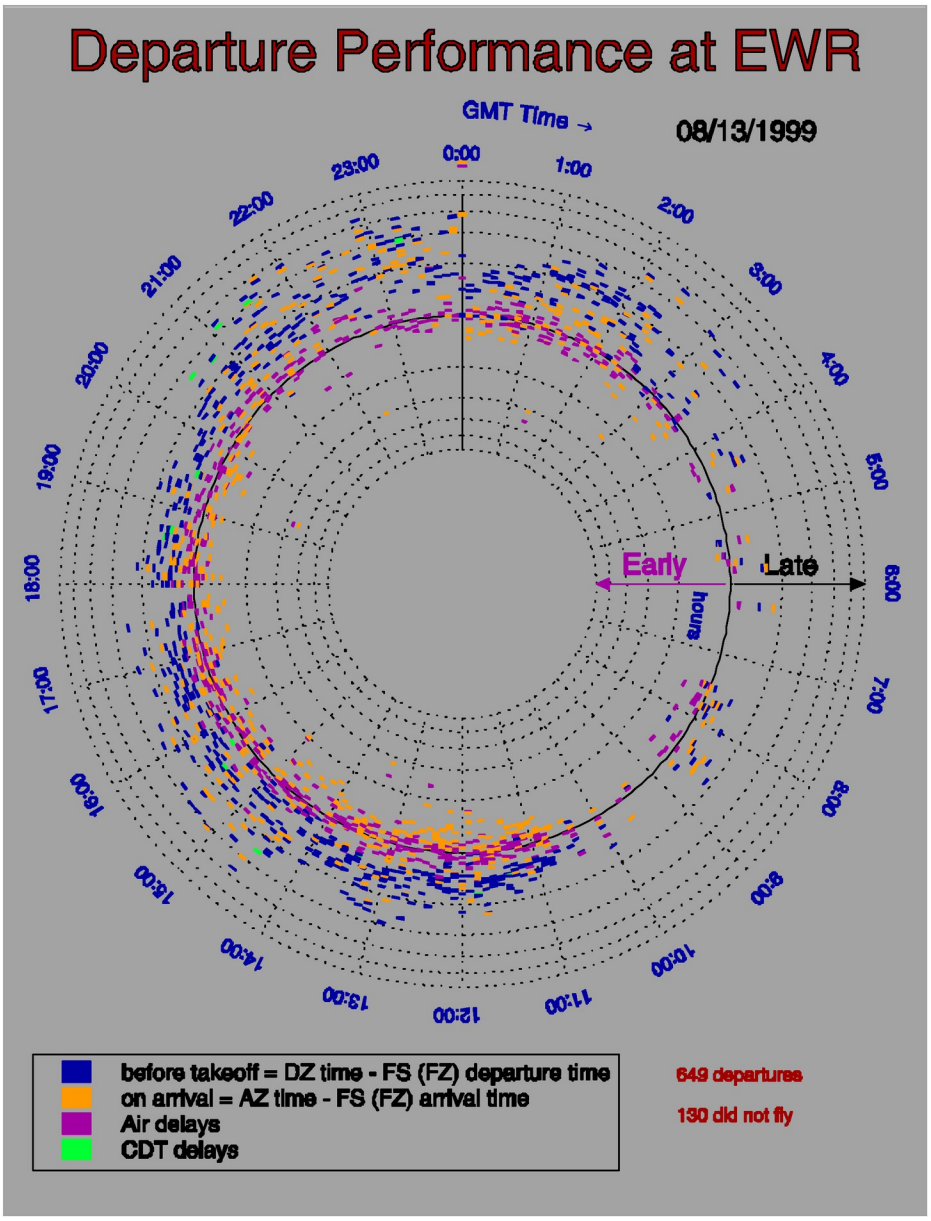
A high-resolution 3D perspective surface mesh (`curv3d` / `surfun` with hidden-line removal and surface shading) mapping two months of continuous daily arrival operations (October 20 through December 20) at Chicago O'Hare. The plot plots local time-of-day (03:00 to 27:00) against calendar date and arrival volume (arrivals/hour), highlighting daily cyclical morning and evening banking waves and severe weather arrival disruptions against a baseline of 1,021 average daily arrivals.



Chicago O'Hare (ORD) Daily Arrival Performance Topography

Newark Liberty International Airport (EWR) 24-Hour Polar Clock Delay Distribution

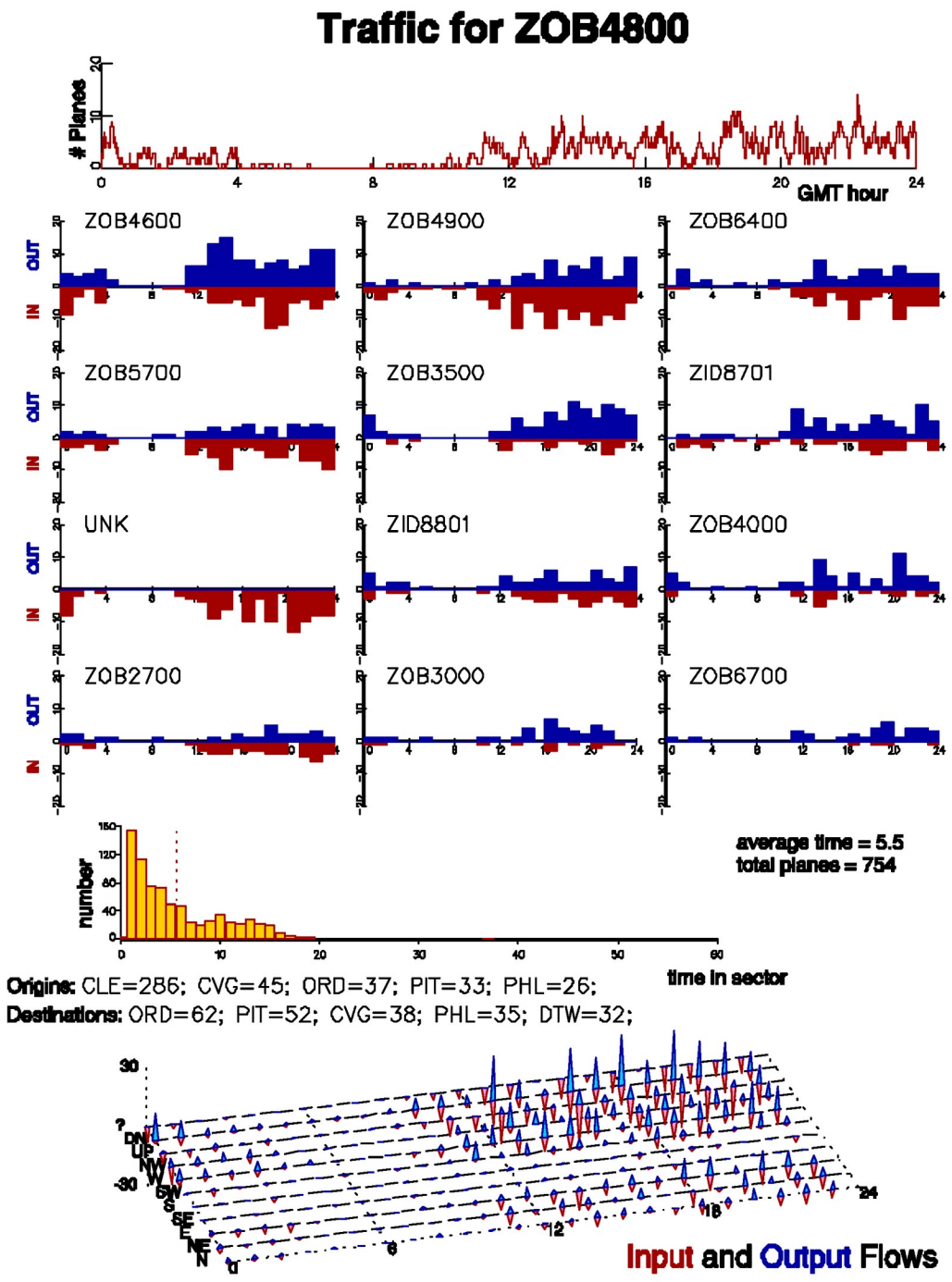
A 24-hour circular polar coordinates chart representing flight schedule compliance and delay types throughout the operational day (GMT). The circular axis represents time of day (0:00 to 23:00 GMT), while radial distance from the circular baseline denotes delay magnitude in hours (concentric circles for Early vs. Late): - **Blue Markers**: Pre-takeoff taxi and gate holds ($AZ-$FS departure time). - **Orange Markers**: Arrival delays (AZ−FS arrival time). - **Purple & Green Markers**: Airborne en route holding and Collaborative Decision Making (CDT) ground delay program holds. - **Operational Summary**: 649 active departures analyzed; 130 flight cancellations identified.



Newark Liberty (EWR) 24-Hour Polar Clock Delay Analysis

Cleveland Air Route Traffic Control Center (ZOB4800) Airspace Sector Activity Dashboard

A multi-panel operational dashboard analyzing en route high-altitude sector loading and boundary hand-offs for Air Route Traffic Control Center sector ZOB4800: - **Top Panel**: 24-hour instantaneous sector aircraft occupancy count. - **Center** 4×3 **Array**: 12 paired bidirectional transfer histograms showing aircraft transitions IN (red bars) and OUT (blue bars) between ZOB4800 and all contiguous boundary sectors (ZOB4600, ZOB4900, ZOB6400, ZOB5700, ZOB3500, ZID8701, UNK, ZID8801, ZOB4000, ZOB2700, ZOB3000, ZOB6700). - **Lower Panel**: Sector dwell-time distribution (average time in sector = 5.5 min across 754 aircraft) with top origin and destination airport tallies (CLE, ORD, CVG, PIT, PHL, DTW). - **Bottom 3D Compass**: 3D directional vector flow projection showing directional traffic volume across the 8 horizontal compass bearings (N, NE, E, SE, S, SW, W, NW) and vertical climb/descent transitions (UP, DN).



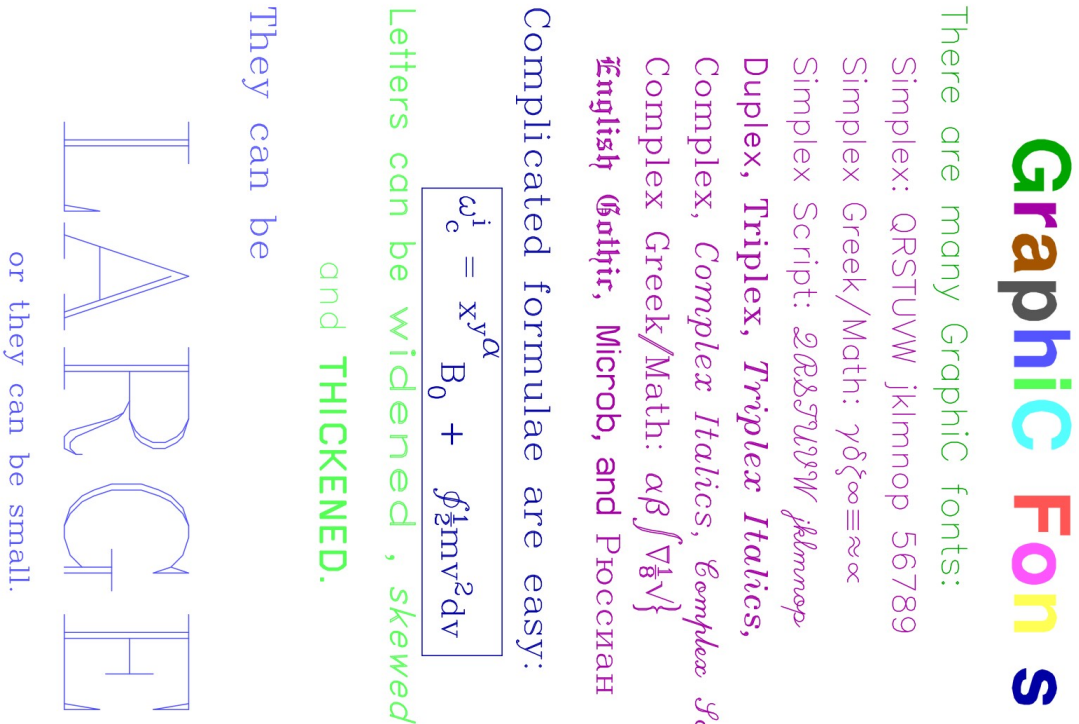
Cleveland ARTCC (ZOB4800) Airspace Sector Flow & Boundary Hand-off Dashboard

9.22 Typography & Vector Stroke Font Showcase (`fnttest`)

The GraphiC vector stroke typography engine renders precision Hershey and mathematical fonts identically across any resolution, scale, or output driver. This comprehensive demonstration (documented on Pages E-102 through E-104 of the original manual) showcases: 1. **Multi-Font Coexistence**: Concurrent active fonts (`simplex.fnt`, `simgrma.fnt`, `simscr.fnt`, `duplex.fnt`, `triplex.fnt`, `tripital.fnt`, `complex.fnt`, `compital.fnt`, `comscr.fnt`, `compgrma.fnt`, `engoth.fnt`, `microb.fnt`, `russian.fnt`). 2. **Mathematical Typesetting**: Nested subscripts and superscripts (`\[`, `\]`), Greek and math symbols, and integral signs:

$$\omega_i^c = x\left[y^a B \right]_0 + \oint \frac{1}{2} m v^2 dv$$

3. **Geometric Character Transformations**: - **Widening (`widen`)**: Dynamically scales horizontal aspect ratio without vertical distortion. - **Skewing (`skew`)**: Applies arbitrary mathematical slant angles. - **Thickening (`tfont,#...#`)**: Generates bold stroked contours. - **Arbitrary Sizing**: Renders headers up to 1.0 inch height (`LARGE`) down to micro-subscripts (`.15`).



GraphiC Typography & Vector Stroke Font Showcase

Complete C Implementation (FNTTEST.C)

```
/*****************************************************************************
    (c) 1984-1993 by Scientific Endeavors Corporation.
    This program is a demonstration of the font plotting features.
    Documented on Pages E-102, E-103, and E-104 of the GraphiC User's Manual.
*****************************************************************************/
#include <graphic.h>

void GPC_MAIN(void)
{
    bgnplot(0, 'S', "fntdemo.svg");
    startplot(WHITE);
    metricunits(0);                         /* Ensure scaling is in inch
units */
    rotate(1);                              /* Plot rotated 90 degrees */
    fillfont(1);                            /* Enable filled fonts */
    page(6.884f, 9.0f);                     /* Size of page and plot area */

    /* The first string is Swiss Bold with each character a different color
*/
    font(1, "swissbld.fnt", '\310');
    lmargin(0.1f);                          /* Left margin is .1 inch */
    tmargin(0.0f);                          /* No top margin */
    linesp(1.1f);                           /* Line space is 1.1 * char
height */
    ctline("\310|2|G|5|r|6|a|8|p|9|h|10|i|11|C |12|F|13|o|14|n|15|t|1|s",
.45f);

    /* An example of the Simplex fonts */
    font(3, "simplex.fnt", '\310', "simgrma.fnt", '\311', "simscr.fnt", '\
312');
    color(GREEN);
    linesp(2.9f);
    ltline("\310There are many GraphiC fonts:", .2f);
    color(MAGENTA);
    linesp(2.2f);
    lmargin(.5f);
    ltline("\310Simplex: QRSTUVW jklmnop 56789", .18f);
    ltline("\310Simplex Greek/Math: \311gdz8@#$~", .18f);
    ltline("\310Simplex Script: \312QRSTUVW jklmnop", .18f);

    /* Duplex and Triplex fonts */
    font(3, "duplex.fnt", '\310', "triplex.fnt", '\311', "tripital.fnt", '\
312');
    ltline("\310Duplex, \311Triplex, \312Triplex Italics,", .18f);

    /* Complex fonts */
    font(4, "complex.fnt", '\310', "compital.fnt", '\311',
            "comscr.fnt", '\312', "compgrma.fnt", '\313');
```

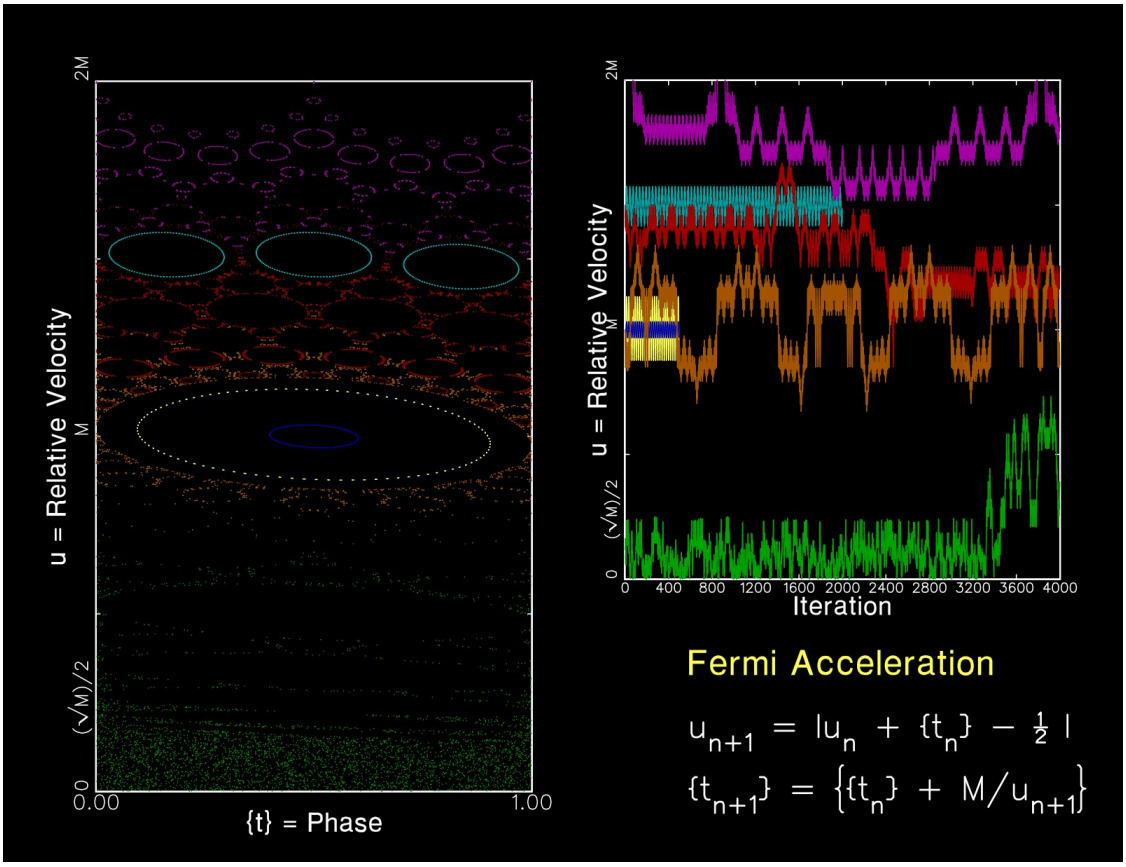
```
    ltline("\310Complex, \311Complex Italics, \312Complex Script,", .18f);
    ltline("\310Complex Greek/Math: \313ab,0E&}", .18f);

    /* English Gothic, Microb, and Russian fonts */
    font(3, "engoth.fnt", '\310', "microb.fnt", '\311', "russian.fnt", '\
312');
    ltline("\310English Gothic, \311Microb, and \312Russian", .18f);

    /* Mathematical formulae using Greek/Math fonts */
    font(2, "complex.fnt", '\310', "compgrma.fnt", '\311');
    color(BLUE);
    lmargin(0.1f);
    linesp(2.9f);
    ltline("\310Complicated formulae are easy:", .2f);
    fbox(1, .05f);                          /* Box around formula string */
    ctline("\311w\310[i]]c[ = x[y[\311a]]\310B]0[ + \311 2A\310mv[2]dv",
.18f);
    fbox(0, .1f);

    /* Widened, skewed, thick, and sizeable characters */
    font(4, "simplex.fnt", '\310', "duplex.fnt", '\311',
            "complex.fnt", '\312', "microb.fnt", '\313');
    color(LGT_GREEN);
    linesp(2.8f);
    widen(1.5f);                            /* Specify aspect ratio for
widened text */
    skew(.3f);                              /* Specify slant of skewed text
*/
    tfont(4);
    ltline("\311Letters can be ^widened^ , `skewed`", .2f);
    charspc(4);
    ctline("\310and \313#THICKENED#.", .2f);
    charspc(0);
    color(LGT_BLUE);
    ltline("\312They can be", .2f);
    linesp(1.5f);
    ctline("\312LARGE", 1.0f);
    linesp(2.9f);
    ctline("\312or they can be small.", .15f);

    endplot();
    stopplot();
}
```

9.23 Simultaneous Plots: Fermi Acceleration (`fermi`)

The simultaneous plotting facility in GraphiC (simuplot, context) allows multiple concurrent graphic coordinate windows to evolve and update dynamically on the same page. Documented on Pages E-108 through E-112 of the original GraphiC User's Manual, this classic scientific demonstration models the nonlinear dynamics of Fermi acceleration —simultaneously charting the Poincaré phase space section alongside the particle's velocity iteration history.

Physical Model & Hamiltonian Chaos

Enrico Fermi originally proposed (1949) that cosmic rays could attain ultra-relativistic energies through repeated collisions with massive, moving interstellar magnetic cloud structures. Joseph Ford popularized a simplified one-dimensional Hamiltonian realization in "A Picture Book of Stochasticity" (in Topics in Nonlinear Dynamics: A Tribute to Sir Edward Bullard, edited by Siebe Jorna, American Institute of Physics, 1978, p. 140).



Simultaneous Plots: Fermi Acceleration Phase Space and Velocity Trajectories

Complete C Implementation (FERMI.C)

```
/*=========================================================================
 * GraphiC Classic C Suite: Fermi
 *
 * How to Compile & Run (from GraphiC repository root):
 *   make examples
 *   ./bin/Fermi
 *
 * Direct Command-Line Compilation:
 *   clang -O2 -IInclude -DGPC_PORTABLE=1 c/examples/Fermi.c lib/libgraphic.a -lm -o bin/Fermi
 *   ./bin/Fermi
 *=========================================================================*/
```

/* This program gives an example of doing simultaneous plots with GraphiC.

Custom y-axis labels are used.

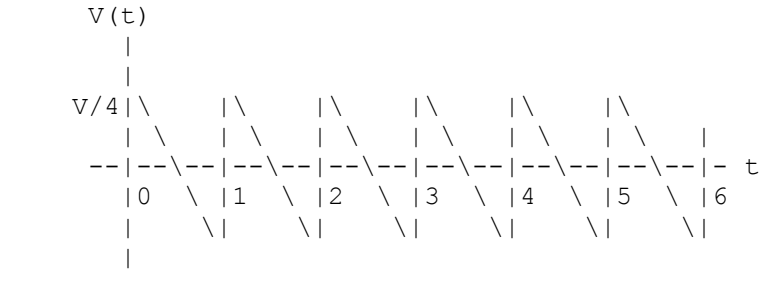
The example is a simple model of Fermi acceleration. Fermi claimed that
cosmic rays could be accelerated by a similar method to very high velocities.
However, as you can see from the results, the method does not work well
because the stochastic acceleration regions are divided by nonstochastic
regions at M, 2M, etc. The brown points start on this separatrix. They can
sneak through the region at M, but it is unlikely that the trajectory will
also sneak through the separatrix at 2M.

A good reference for this topic is Joseph Ford's "A Picture Book of
Stochasticity" on page 140 in the book Topics in Nonlinear Dynamics:
A Tribute to Sir Edward Bullard, edited by Siebe Jorna, American Institute of Physics, 1978.

A massless ball bounces in one dimension between two massive walls. One wall
oscillates with a velocity given by

$$V(t) = (V/4)[1 - 2\{t\}]$$

where {t} is the fractional part of t (denoted by ft in the code).



The velocity of the ball just before the nth collision with the oscillating

wall is

$$U_{n+1} = |u + \{t_n\} - 1/2|$$

and if the space between the walls (d) is much greater than the maximum amplitude of oscillation (V/16 = 2a),

$$\{t_{n+1}\} = \{\{t_n\} + M/U_{n+1}\}$$

where u = the velocity of the particle/v, and M = d/16a.

This program simultaneously plots the velocity of the particle and its phase space trajectory for several starting points.

*/

```
#include <graphic.h>
#define NCURVE 7

float M = 10.0f;
int nmax = 4000;
double ustart[NCURVE];
double tstart[NCURVE];
int stepmax[NCURVE];

/*****************************************************************/
void gpc_init(void);
void GPC_MAIN(void)
{
        int i, ncolor = 14;
        int ch;
        int n = 0;                          /* Iteration number */
        float u[2];                 /* Relative velocity before collision */
        float ft[2];                        /* Fractional parts of t */
        float nstep[2];                /* Steps (need float for the plot) */

        bgnplot(1, 'g', "fermi.tkf");          /* Generate fermi.svg & fermi.tkf */
        plotabsolute(1);              /* Prevent GraphiC from resizing axes */
        metricunits(0);             /* Ensure scaling is done in inch units */
        startplot(BLACK);
        font(3,"simplex.fnt",'\310',"swiss.fnt",'\311',"simgrma.fnt",'\312');
        fillfont(1);
        gpc_init();
        ustart[0] = ustart[1] = M;       /* Pick interesting starting points */
        stepmax[0] = stepmax[1] = 500;
        tstart[0] = .1f;
```

```
tstart[1] = .4f;
ustart[2] = .4f*sqrt((double)M);
tstart[2] = .5f;
stepmax[2] = 4000;
ustart[3] = 1.45f*M;
tstart[3] = .5f;
stepmax[3] = 2000;
ustart[4] = 1.5f*M;
tstart[4] = 0.0f;
stepmax[4] = 4000;
ustart[5] = 2.0f*M;
tstart[5] = 0.5f;
stepmax[5] = 4000;
ustart[6] = M;
tstart[6] = 0.0f;
stepmax[6] = 4000;
for (i = 0; i < NCURVE; i++) {
        u[0] = (float)ustart[i];
        ft[0] = (float)tstart[i];
        nstep[0] = 0.0f;
        context(0);
        symbol(ft[0], u[0], 16);
        for (n = 1; n < stepmax[i]; n++) {
                nstep[1] = (float)n;
                u[1] = u[0] + ft[0] - .5f;
                if (u[1] < 0.0f)
                        u[1] = -u[1];
                ft[1] = ft[0] + (M/u[1]);
                while (ft[1] > 1.0f)
                        ft[1] -= 1.0f;
                context(0);                     /* First context */
                symbol(ft[1], u[1], 16);
                context(1);                     /* Second context */
                curve(nstep, u, 2, 0);
                u[0] = u[1];
                ft[0] = ft[1];
                nstep[0] = nstep[1];
                if ((ch = cstsq()) == 'q')
                        goto end;
        }
        ncolor++;
        if (ncolor==15)
                ncolor = 1;
        if (ncolor==7)
                ncolor = 9;
        context(0);         /* Change colors for the different contexts */
```

```
                color(ncolor);
                context(1);
                color(ncolor);
        }
end:
        simuplot(0);   /* Must end the simultaneous plots to make the TKF file */
        endplot();
        stopplot();
}

/*************************************************************************/
/* Set up the two contexts for the simultaneous plots */
void gpc_init(void)
{
        simuplot(2);                            /* Two contexts */
        context(0);/* Establish page and plot parameters for the first context */
        color(BRT_WHITE);
        frame(1, 1);
        tcurve(2);
        page(4.0f, 6.884f);
        pgshift(0.5f, 0.0f);
        symht(.05f);                    /* This call must come after page */
        area2d(3.5f, 5.7f);             /* Set the area of the Poincar, plot */
        xname("\311{t} = Phase\310");
        yname("\311u = Relative Velocity\310");
        graf("%-1.2f", 0.0f, 1.0f, 1.0f, " ", 0.0f, 0.5f*M, 2.0f*M);
        pltfnt(-.02f, 0.0f, "\310""0", .1f, 90);   /* Custom labels for y axis */
        pltfnt(-.02f, M, "\310M", .1f, 90);
        pltfnt(-.02f, 2.f*M, "\310""2M", .1f, 90);
        pltfnt(-.02f, (float)(sqrt((double)M))*.5f, "\310(\312""1\310M)/2", .1f, 90);

/* LABEL THE PLOT */
        color(BLUE);            /* Outline the title in blue (for printer) */
        prtfnt(5.0f, 1.5f, "\311|14|Fermi Acceleration", 0.2f, 0);
        color(BRT_WHITE);
        tfont(3);
/* The `#' toggles the thick fonts */
        prtfnt(5.0f, 1.0f, "\310#u]n+1[ = @|u]n[ + {t]n[} - \312A\310 @|", 0.18f, 0);
        prtfnt(5.0f, 0.5f, "\310{t]n+1[} = \312{\310{t]n[} + M/u]n+1[\312}#\310", 0.18f,
0);
        tfont(0);
        tcurve(0);
        color(YELLOW);                          /* Set context color */

/* START VELOCITY PLOT */
        context(1);/* Establish page and plot parameters for the second context */
```

```
        color(BRT_WHITE);
        symht(.01f);                    /* This call must come after page */
        page(4.5f, 4.0f);
        pgshift(4.5f,2.3f);
        axnamht(.3f);
        xname("\311Iteration");
        yname("\311u = Relative Velocity");
        area2d(3.5f, 4.0f);
                /* The `-' is to left align the axis labels */
        graf("%-1.0f", 0.0f, 0.1f*nmax, (float)nmax,
                " ", 0.0f, .5f*M, 2.f*M);
                        /* Custom labels for y-axis */
        pltfnt(-.02f*nmax, -0.0f, "\310""0", .08f, 90);
        pltfnt(-.02f*nmax, M, "\310M", .08f, 90);
        pltfnt(-.02f*nmax, 2.f*M, "\310""2M", .08f, 90);
        pltfnt(-.02f*nmax, (float)(sqrt((double)M))*.5f, "\310(\312""1\310M)/2", .08f, 90);
        color(YELLOW);
}
```

Appendix: GraphiC Vector Font Character Tables

Overview and Purpose

The GraphiC library includes 23 stroked vector fonts based on the Hershey scientific typographic database, expanded with Cyrillic (Russian), English Gothic, mathematical symbols, astronomical signs, and cartographic icons.

The utility program `FONTTABL.C` (documented on Pages E-105 through E-107 of the original GraphiC User's Manual) generates a complete, multi-column character reference table for any `.fnt` file. Each character is displayed alongside its decimal ASCII/stroke index, width metrics, and escape code delimiters.

The Complete `FONTTABL.C` Program

```
/*************************************************************************
    FONTTABL.C
    (c) 1984-1993 by Scientific Endeavors Corporation.
    This code makes a font reference table for any GraphiC font
    and was used to generate the authoritative font tables in Appendix B.
    USAGE: fonttabl <fontname>
*************************************************************************/
#include "graphic.h"

int isspecial(int ich)
{
    char ch = (char)ich;
    if (ch == '@' || ch == '[' || ch == ']' || ch == '^' ||
        ch == '`' || ch == '#' || ch == '|')
        return 1;
    return 0;
}

void GPC_MAIN(int argc, char *argv[])
{
    char fontname[80], tkfname[80], title[20];
    char ascii[7], fontchar[5];
    FILE *fptr = NULL;
    int i, j, ch;

    if (argc > 1)
        strcpy(fontname, argv[1]);
    else
        strcpy(fontname, "simplex.fnt");

    strcpy(tkfname, fontname);
    for (i = 0; fontname[i] != '\0'; i++) {
        if (fontname[i] == '.') break;
```

```
        fontname[i] = (char)tolower((int)fontname[i]);
    }
    if (fontname[i] == '\0')
        strcpy(&(fontname[i]), ".fnt");
    strcpy(&(tkfname[i]), ".tkf");

    fptr = gfopen(fontname, "rb", 0);
    if (fptr == NULL) {
        GPC_PUTS("FONTTABL: Could not open font file\n");
        return;
    }
    gfclose(fptr);

    bgnplot(0, 'S', tkfname);
    startplot(BRT_WHITE);
    metricunits(0);                         /* Ensure scaling is in inches */
    rotate(1);                              /* Rotate 90 degrees */
    font(3, "simplex.fnt", '\310', fontname, '\311', "news.fnt", '\312');
    fillfont(1);

    color(BLACK);
    page(6.884f, 0.5f);                     /* Display font title header */
    pgshift(0.0f, 8.5f);
    tmargin(0.05f);
    sprintf(title, "\312%s", fontname);
    linesp(1.0f);
    ctline(title, 0.3f);

    linesp(2.1f);                           /* Table grid spacing */
    widen(0.8f);                            /* Compact character aspect */
    ch = 33;                                /* Start with character code 33
*/
    for (i = 0; i < 10; i++) {              /* 10 columns across page */
        page(0.6884f, 8.5f);
        pgshift((float)i * 0.6884f, 0.0f);
        tmargin(-0.2f);
        lmargin(0.08f);
        rmargin(0.08f);
        linesp(2.5f);
        box();
        for (j = 0; j < 17; j++) {          /* 17 rows per column */
            if (ch > 199) break;
            sprintf(ascii, "\310^%d^", ch);
            if (ch == 127)
                fontchar[0] = '\0';
            else
                sprintf(fontchar, "\311@%c", (char)ch);
            lcrline(ascii, 0.1f, "", 0.0f, fontchar, 0.2f);
            ch++;
```

```
        }
    }
    endplot();
    stopplot();
}
```

Reference Font Character Tables

Below are high-resolution character plates generated directly by `FONTTABL` across the primary scientific and technical vector stroke fonts:

10.1 Simplex Roman Font Plate (simplex.fnt)

The fundamental high-speed stroke font. Single-stroke vectors optimized for rapid rendering on plotters, interactive CRT displays, and compact vector SVGs.

simplex.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	“	186	Í
34	''	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	”	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	ì	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	È	191	Ô
39	’	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	ƒ	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	æ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	–	62	>	79	O	96	‘	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	°
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.2 Complex Roman Font Plate (complex.fnt)

A publication-quality multi-stroke Roman serif typeface patterned after classic mathematical typography with balanced bracketed serifs and proportional stroke weights.

complex.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	“	186	Í
34	”	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	”	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	¡	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	È	191	Ô
39	’	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	ƒ	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	æ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	–	62	>	79	O	96	`	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	°
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.3 Triplex Bold Roman Font Plate (triplex.fnt)

A heavy triple-stroked headline font engineered for high-visibility plot titles, presentation slides, and large-format engineering drawings.

triplex.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	“	186	Í
34	"	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	”	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	¡	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	È	191	Ô
39	’	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	ƒ	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	æ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	−	62	>	79	O	96	‘	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	°
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.4 Duplex Roman Font Plate (duplex.fnt)

Double-stroked sans-serif font offering enhanced visual weight and crisp legibility on medium-resolution displays and desktop laser prints.

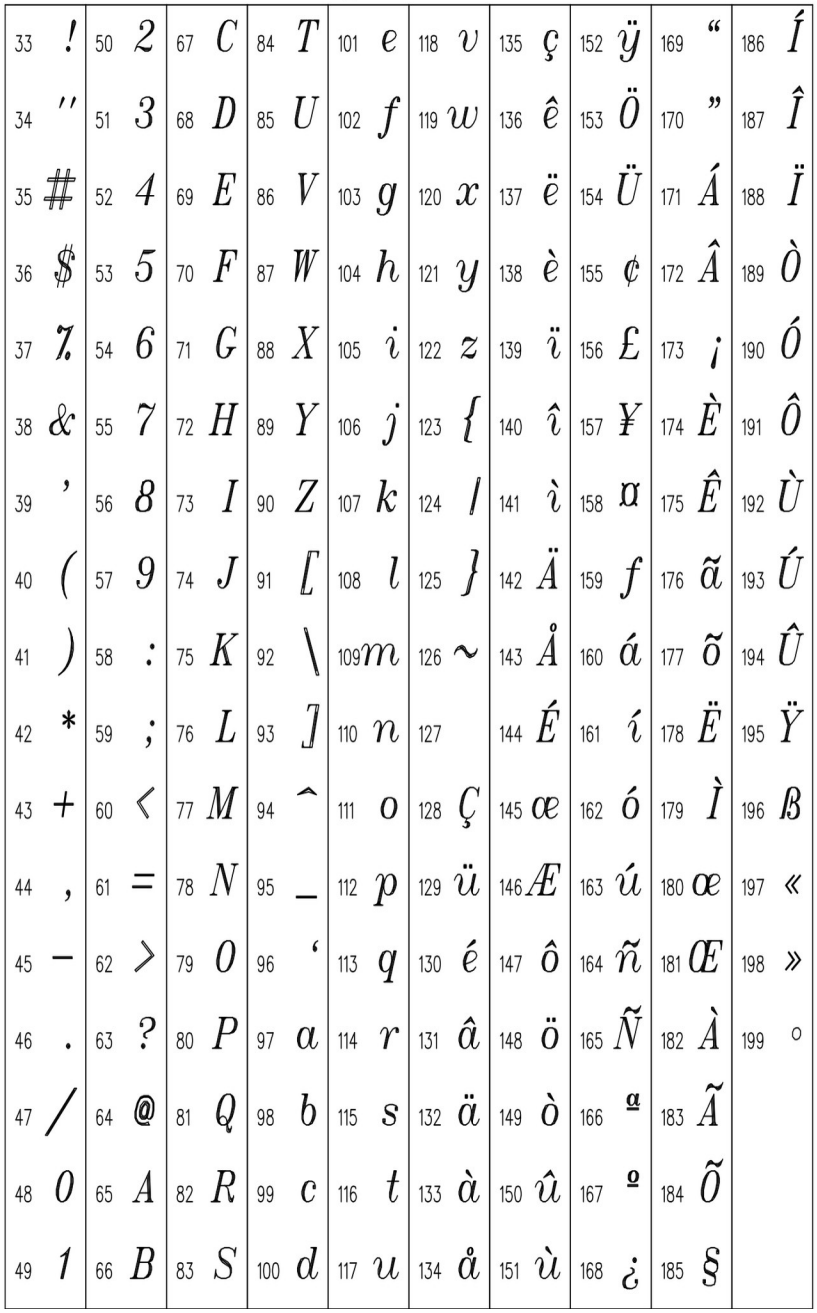
duplex.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	“	186	Í
34	"	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	”	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	ì	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	É	191	Ô
39	’	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	ƒ	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	œ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	–	62	>	79	O	96	‘	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	°
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.5 Triplex Italic Font Plate (tripital.fnt)

Triple-stroked slanted Roman serif font designed for emphasized title strings, book titles, and formal presentation headings.

tripital.fnt



10.6 Complex Italic Font Plate (compital.fnt)

Multi-stroke serif italic font designed for mathematical variables (x, y, z, t), physical parameters, and formal publication annotations.

compital.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	“	186	Í
34	”	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	”	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	¡	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	È	191	Ô
39	’	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	f	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	œ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	−	62	>	79	O	96	`	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	°
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.7 Complex Script Font Plate (comscr.fnt)

Formal multi-stroke cursive calligraphic script for elegant mathematical operators (e.g. Lagrangian L, Hamiltonian H, Fourier F) and commemorative inscriptions.

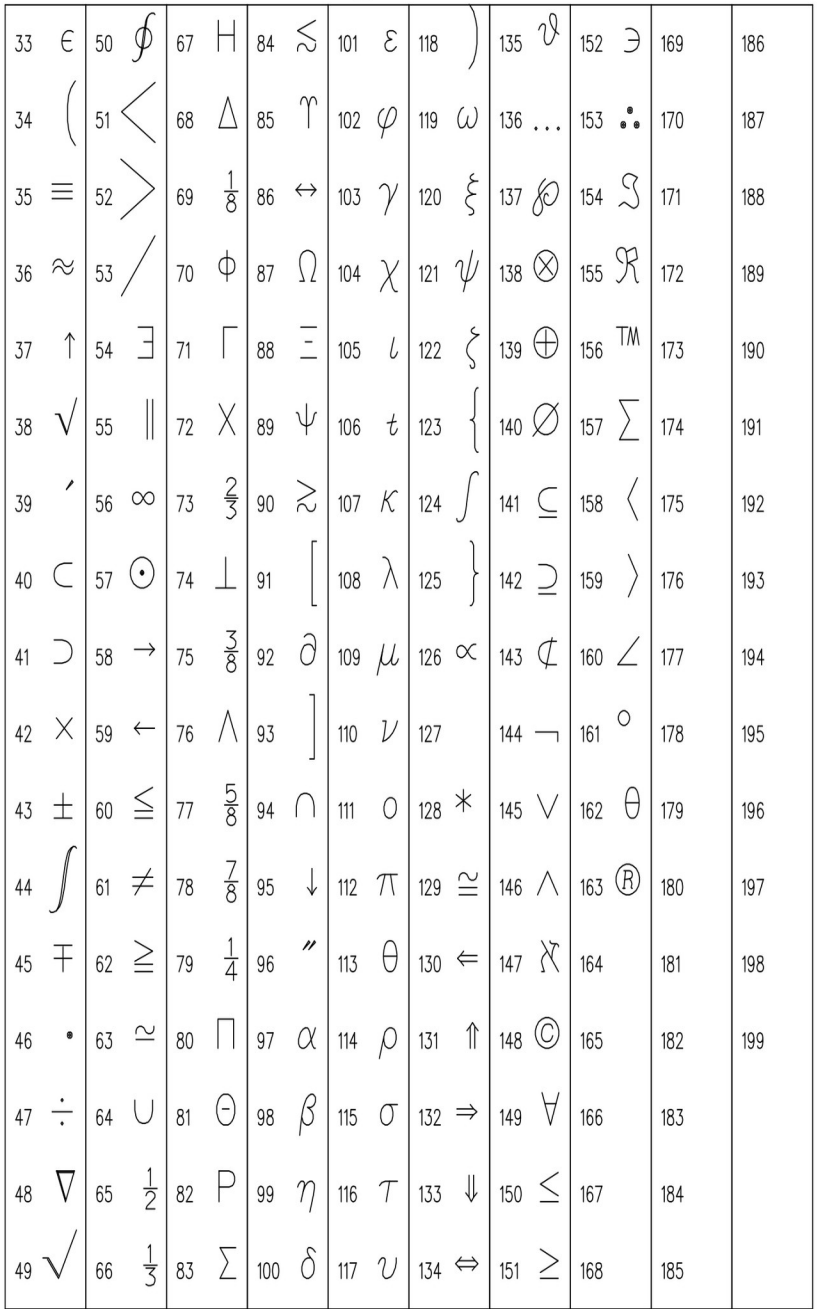
comscr.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135		152		169		186	
34	"	51	3	68	D	85	U	102	f	119	w	136		153		170		187	
35		52	4	69	E	86	V	103	g	120	x	137		154		171		188	
36	$	53	5	70	F	87	W	104	h	121	y	138		155		172		189	
37		54	6	71	G	88	X	105	i	122	z	139		156		173		190	
38	&	55	7	72	H	89	Y	106	j	123		140		157		174		191	
39	'	56	8	73	I	90	Z	107	k	124		141		158		175		192	
40	(	57	9	74	J	91		108	l	125		142		159		176		193	
41	)	58	:	75	K	92		109	m	126		143		160		177		194	
42	*	59	;	76	L	93		110	n	127		144		161		178		195	
43	+	60		77	M	94		111	o	128		145		162		179		196	
44	,	61	=	78	N	95		112	p	129		146		163		180		197	
45	–	62		79	O	96	‘	113	q	130		147		164		181		198	
46	.	63	?	80	P	97	a	114	r	131		148		165		182		199	
47	/	64		81	Q	98	b	115	s	132		149		166		183			
48	0	65	A	82	R	99	c	116	t	133		150		167		184			
49	1	66	B	83	S	100	d	117	u	134		151		168		185			

10.8 Simplex Greek & Math Font Plate (simgrma.fnt)

Single-stroke Greek alphabet (alpha, beta, gamma, etc.) and core mathematical operators, relations, and symbols designed for high-speed scientific formula rendering.

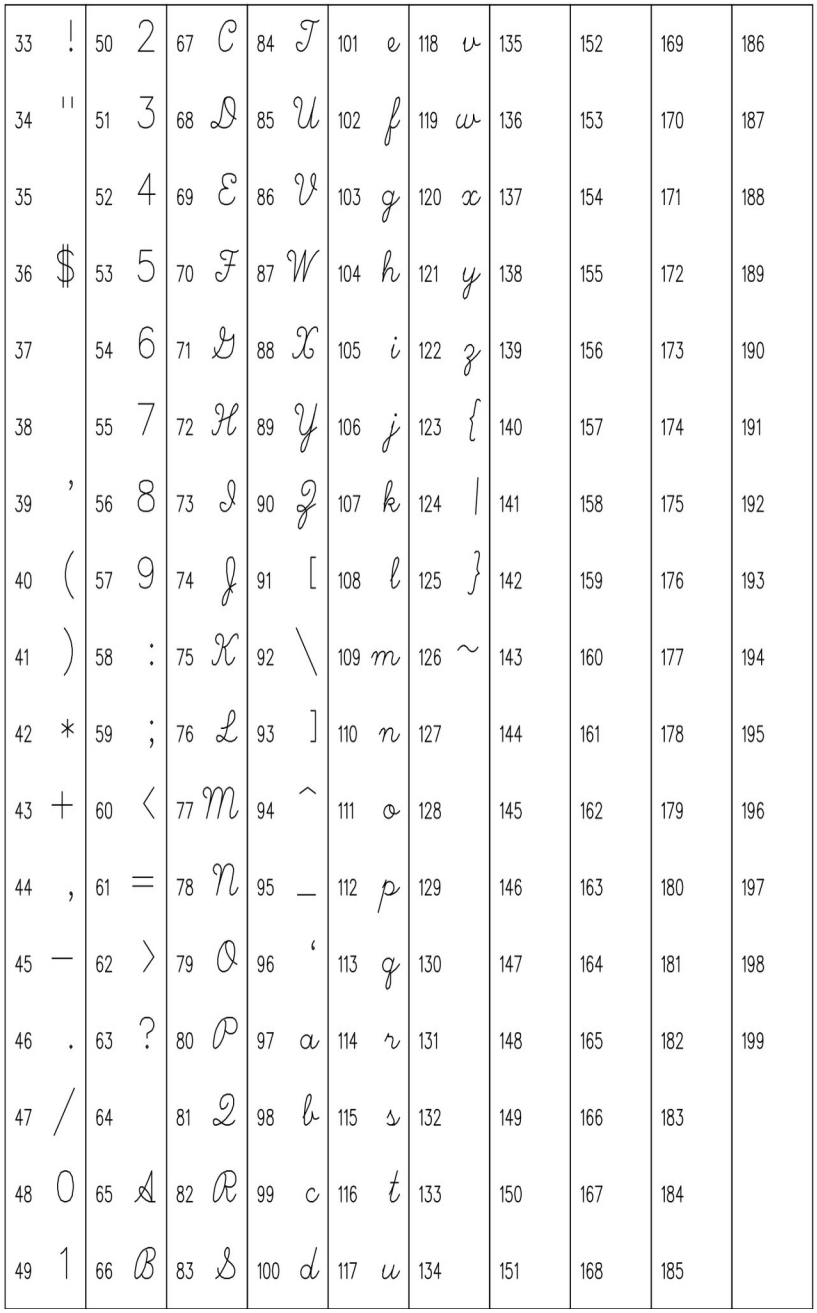
simgrma.fnt



10.9 Simplex Script Font Plate (simscr.fnt)

Single-stroke flowing cursive handwriting font for casual figure annotations, informal titles, and cursive mathematical variables.

simscr.fnt



10.10 Complex Greek & Math Font Plate (compgrma.fnt)

High-density multi-stroke Greek letters, contour integrals, nabla/del operator, partial derivatives, and mathematical logic symbols matching Complex Roman.

compgrma.fnt

33	∈	50	∮	67	H	84	≲	101	ε	118	)	135	ϑ	152	∋	169		186	
34	(	51		68	Δ	85	ϒ	102	φ	119	ω	136	…	153	∴	170		187	
35	≡	52	†	69	⅛	86	↔	103	γ	120	ξ	137	℘	154	ℑ	171		188	
36	≈	53	‡	70	Φ	87	Ω	104	χ	121	ψ	138	⊗	155	ℜ	172		189	
37	↑	54	∃	71	Γ	88	Ξ	105	ι	122	ζ	139	⊕	156	™	173		190	
38	√	55	∥	72	X	89	Ψ	106	t	123	{	140	∅	157	∑	174		191	
39	′	56	∞	73	⅔	90	≳	107	κ	124	∫	141	⊆	158	⟨	175		192	
40	⊂	57	⊙	74	⊥	91	[	108	λ	125	}	142	⊇	159	⟩	176		193	
41	⊃	58	→	75	⅜	92	∂	109	μ	126	∝	143	⊄	160	∠	177		194	
42	×	59	←	76	Λ	93	]	110	ν	127		144	¬	161	°	178		195	
43	±	60	≦	77	⅝	94	∩	111	o	128	*	145	∨	162	θ	179		196	
44	∫	61	≠	78	⅞	95	↓	112	π	129	≅	146	Λ	163	®	180		197	
45	∓	62	≧	79	¼	96	″	113	ϑ	130	⇐	147	ℵ	164		181		198	
46	•	63	≃	80	Π	97	α	114	ρ	131	⇑	148	©	165		182		199	
47	÷	64	∪	81	Θ	98	β	115	σ	132	⇒	149	∀	166		183			
48	∇	65	½	82	P	99	η	116	τ	133	⇓	150	≤	167		184			
49	√	66	⅓	83	Σ	100	δ	117	υ	134	⇔	151	≥	168		185			

10.11 Swiss Regular Font Plate (swiss.fnt)

Clean Neo-Grotesque sans-serif (Helvetica style) with solid polygon filled glyph interiors. Modern, highly readable, and versatile for technical charting.

swiss.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	“	186	Í
34	"	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	”	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	¡	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	È	191	Ô
39	’	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	ƒ	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	æ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	-	62	>	79	O	96	‘	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	•
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.12 Swiss Bold Font Plate (swissbld.fnt)

Heavy solid-filled sans-serif font. High-impact bold weight designed for prominent chart headings, callouts, and multi-panel subplot banners.

swissbld.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	“	186	Í
34	"	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	”	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	¡	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	È	191	Ô
39	’	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	ƒ	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	æ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	-	62	>	79	O	96	‘	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	•
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.13 Swiss Italic Font Plate (swissitl.fnt)

Slanted solid-filled Neo-Grotesque sans-serif for secondary subheadings, figure legends, and technical descriptions.

swissitl.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	“	186	Í
34	"	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	”	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	¡	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	È	191	Ô
39	’	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	ƒ	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	æ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	-	62	>	79	O	96	‘	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	•
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.14 Swiss Bold Italic Font Plate (swissbit.fnt)

Solid-filled bold oblique sans-serif for high-emphasis callouts, critical parameter warnings, and prominent graphical labels.

swissbit.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	“	186	Í
34	"	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	”	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	¡	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	È	191	Ô
39	’	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	ƒ	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	æ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	-	62	>	79	O	96	‘	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	•
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.15 Swiss Narrow Font Plate (swissn.fnt)

Compact, condensed solid-filled sans-serif designed for high-density data tables, tight axis tick intervals, and constrained chart margins.

swissn.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	“	186	Í
34	"	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	”	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	¡	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	È	191	Ô
39	’	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	ƒ	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	æ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	-	62	>	79	O	96	‘	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	•
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.16 News Roman Font Plate (news.fnt)

Classic solid-filled Roman serif typeface with traditional proportions, bracketed serifs, and high contrast inspired by classic newspaper typography.

news.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	“	186	Í
34	"	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	”	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	¡	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	È	191	Ô
39	’	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	ƒ	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	æ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	-	62	>	79	O	96	‘	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	•
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.17 News Greek & Math Font Plate (newsgrm.fnt)

Solid-filled Greek characters and mathematical operators matching the serif styling and weight of the News Roman typeface.

newsgrm.fnt

33	∈	50	∮	67	ℑ	84	≲	101	ε	118	⎫	135	ϑ	152		169		186	
34	⎛	51	®	68	Δ	85	ϒ	102	φ	119	ω	136	…	153		170		187	
35	≡	52	©	69	ℜ	86	⇔	103	γ	120	ξ	137	℘	154		171		188	
36	≈	53	™	70	Φ	87	Ω	104	χ	121	ψ	138	⊗	155		172		189	
37	⇑	54	∃	71	Γ	88	Ξ	105	ι	122	ζ	139	⊕	156		173		190	
38	√	55	‖	72	™	89	Ψ	106	ɨ	123	⎧	140	∅	157		174		191	
39	′	56	∞	73	∑	90	≳	107	κ	124	⎨	141	⊆	158		175		192	
40	⊂	57	∀	74	⊥	91	⎡	108	λ	125	⎩	142	⊇	159		176		193	
41	⊃	58	⇒	75	⟨	92	∂	109	μ	126	∝	143	⊄	160		177		194	
42	×	59	⇐	76	Λ	93	⎤	110	ν	127		144	¬	161		178		195	
43	±	60	≤	77	⟩	94	∩	111	ο	128	*	145	∨	162		179		196	
44	⎰	61	≠	78	∠	95	⇓	112	π	129	≅	146	∧	163		180		197	
45	∓	62	≥	79	°	96	″	113	θ	130	←	147		164		181		198	
46	•	63	≃	80	Π	97	α	114	ρ	131	↑	148		165		182		199	
47	÷	64	∪	81	Θ	98	β	115	σ	132	→	149		166		183			
48	∇	65	∋	82	Ρ	99	η	116	τ	133	↓	150		167		184			
49	ℵ	66	∴	83	Σ	100	δ	117	υ	134	↔	151		168		185			

10.18 Block Heavy Gothic Font Plate (block.fnt)

Heavy geometric solid-filled display font for large poster titles, major laboratory equipment schematics, and high-contrast structural banners.

block.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135	ç	152	ÿ	169	‘‘	186	Í
34	"	51	3	68	D	85	U	102	f	119	w	136	ê	153	Ö	170	’’	187	Î
35	#	52	4	69	E	86	V	103	g	120	x	137	ë	154	Ü	171	Á	188	Ï
36	$	53	5	70	F	87	W	104	h	121	y	138	è	155	¢	172	Â	189	Ò
37	%	54	6	71	G	88	X	105	i	122	z	139	ï	156	£	173	À	190	Ó
38	&	55	7	72	H	89	Y	106	j	123	{	140	î	157	¥	174	È	191	Ô
39	'	56	8	73	I	90	Z	107	k	124	|	141	ì	158	¤	175	Ê	192	Ù
40	(	57	9	74	J	91	[	108	l	125	}	142	Ä	159	ƒ	176	ã	193	Ú
41	)	58	:	75	K	92	\	109	m	126	~	143	Å	160	á	177	õ	194	Û
42	*	59	;	76	L	93	]	110	n	127		144	É	161	í	178	Ë	195	Ÿ
43	+	60	<	77	M	94	^	111	o	128	Ç	145	æ	162	ó	179	Ì	196	ß
44	,	61	=	78	N	95	_	112	p	129	ü	146	Æ	163	ú	180	œ	197	«
45	–	62	>	79	O	96	`	113	q	130	é	147	ô	164	ñ	181	Œ	198	»
46	.	63	?	80	P	97	a	114	r	131	â	148	ö	165	Ñ	182	À	199	°
47	/	64	@	81	Q	98	b	115	s	132	ä	149	ò	166	ª	183	Ã		
48	0	65	A	82	R	99	c	116	t	133	à	150	û	167	º	184	Õ		
49	1	66	B	83	S	100	d	117	u	134	å	151	ù	168	¿	185	§		

10.19 English Gothic Font Plate (engoth.fnt)

Traditional Old English fraktur/blackletter typeface for formal institutional headings, diploma text, and historical certificate plotting.

engoth.fnt

33	!	50	2	67	C	84	T	101	e	118	v	135		152		169		186	
34	"	51	3	68	D	85	U	102	f	119	w	136		153		170		187	
35		52	4	69	E	86	V	103	g	120	x	137		154		171		188	
36	$	53	5	70	F	87	W	104	h	121	y	138		155		172		189	
37		54	6	71	G	88	X	105	i	122	z	139		156		173		190	
38	&	55	7	72	H	89	Y	106	j	123		140		157		174		191	
39	'	56	8	73	I	90	Z	107	k	124		141		158		175		192	
40	(	57	9	74	J	91		108	l	125		142		159		176		193	
41	)	58	:	75	K	92		109	m	126		143		160		177		194	
42	*	59	;	76	L	93		110	n	127		144		161		178		195	
43	+	60		77	M	94		111	o	128		145		162		179		196	
44	,	61	=	78	N	95		112	p	129		146		163		180		197	
45	-	62		79	O	96	‘	113	q	130		147		164		181		198	
46	.	63	?	80	P	97	a	114	r	131		148		165		182		199	
47	/	64		81	Q	98	b	115	s	132		149		166		183			
48	0	65	A	82	R	99	c	116	t	133		150		167		184			
49	1	66	B	83	S	100	d	117	u	134		151		168		185			

10.20 Russian (Cyrillic) Font Plate (russian.fnt)

Complete Cyrillic alphabet (uppercase and lowercase) with authentic typographic proportions for scientific papers and engineering plots in Russian.

russian.fnt

33 Ц	50 2	67 Э	84 Т	101 Й	118 В	135	152	169	186
34 Ъ	51 3	68 Д	85 Ю	102 ф	119 Щ	136	153	170	187
35 е	52 4	69 Й	86 В	103 г	120 Х	137	154	171	188
36 ы	53 5	70 Ф	87 Щ	104 ж	121 у	138	155	172	189
37 ь	54 6	71 Г	88 Х	105 и	122 з	139	156	173	190
38 я	55 7	72 Ж	89 У	106 ч	123 {	140	157	174	191
39 '	56 8	73 И	90 З	107 к	124 |	141	158	175	192
40 (	57 9	74 Ч	91 [	108 л	125 }	142	159	176	193
41)	58 :	75 К	92 \	109 м	126 ~	143	160	177	194
42 Е	59 ;	76 Л	93]	110 н	127	144	161	178	195
43 Ъ	60 <	77 М	94 ^	111 о	128	145	162	179	196
44 Я	61 =	78 Н	95 _	112 п	129	146	163	180	197
45 Ь	62 >	79 О	96 `	113 ш	130	147	164	181	198
46 Ц	63 ?	80 П	97 а	114 р	131	148	165	182	199
47 Ы	64 @	81 Ш	98 б	115 с	132	149	166	183	
48 0	65 А	82 Р	99 э	116 т	133	150	167	184	
49 1	66 Б	83 С	100 д	117 ю	134	151	168	185	

10.21 Matrix Font Plate (matrix.fnt)

Dot-matrix simulation font replicating the aesthetic of 9-pin/24-pin impact matrix printers and hardware instrumentation data logger readouts.

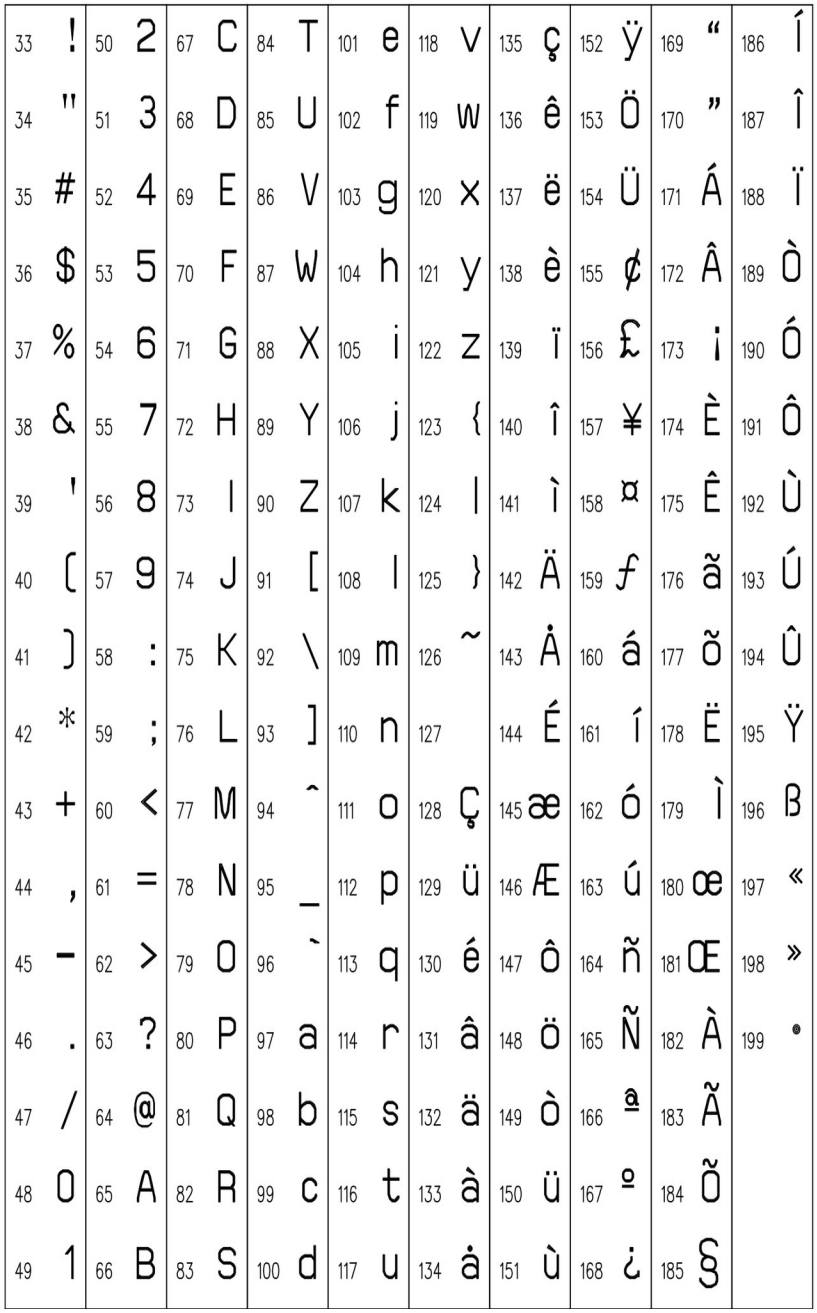
matrix.fnt

33 !	50 2	67 C	84 T	101 e	118 v	135 ç	152 ÿ	169	186
34 "	51 3	68 D	85 U	102 f	119 w	136 ê	153 Ö	170	187
35 #	52 4	69 E	86 V	103 g	120 x	137 ë	154 Ü	171	188
36 $	53 5	70 F	87 W	104 h	121 y	138 è	155 ¢	172	189
37 %	54 6	71 G	88 X	105 i	122 z	139 ï	156 £	173	190
38 &	55 7	72 H	89 Y	106 j	123 {	140 î	157 ¥	174	191
39 '	56 8	73 I	90 Z	107 k	124 |	141 ì	158 ₧	175	192
40 (	57 9	74 J	91 [	108 l	125 }	142 Ä	159 ƒ	176	193
41)	58 :	75 K	92 \	109 m	126 ~	143 Å	160 á	177	194
42 *	59 ;	76 L	93]	110 n	127	144 É	161 í	178	195
43 +	60 <	77 M	94 ^	111 o	128 Ç	145 æ	162 ó	179	196
44 ,	61 =	78 N	95 _	112 p	129 ü	146 Æ	163 ú	180	197
45 -	62 >	79 O	96 `	113 q	130 é	147 ô	164 ñ	181	198
46 .	63 ?	80 P	97 a	114 r	131 â	148 ö	165 Ñ	182	199
47 /	64 @	81 Q	98 b	115 s	132 ä	149 ò	166	183	
48 0	65 A	82 R	99 c	116 t	133 à	150 û	167	184	
49 1	66 B	83 S	100 d	117 u	134 a	151 ù	168	185	

10.22 Micro Bold Font Plate (microb.fnt)

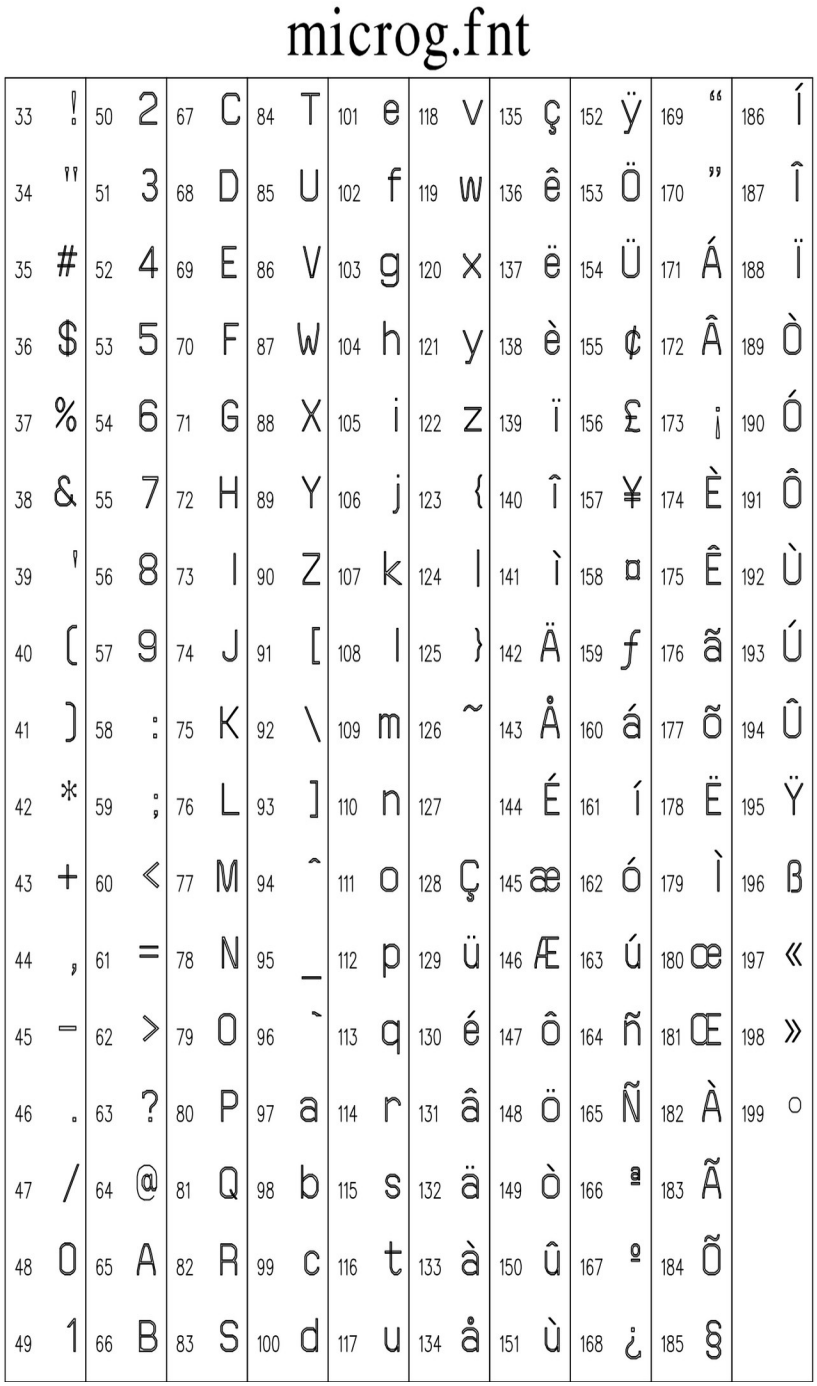
Compact sans-serif font engineered for maximum legibility at tiny point sizes, ideal for dense subscripts, footnotes, and multi-trace legend keys.

microb.fnt



10.23 Micro Gothic Font Plate (microg.fnt)

Fine-detail technical Gothic stroke font for densely packed scientific engineering schematics and architectural blueprints.



Complete Inventory of GraphiC Vector Stroke Fonts

Font Filename	Typographic Classification	Number of Strokes	Primary Use Case
simplex.fnt	Single-Stroke Roman Sans	Single	Standard axis labels, tick numbers, fast plotting
duplex.fnt	Double-Stroke Roman Sans	Double	Intermediate body labels, clean readable text
triplex.fnt	Triple-Stroke Bold Roman	Triple	Primary headings, major axis titles
tripital.fnt	Triple-Stroke Italic Roman	Triple	Emphasized mathematical variables
complex.fnt	Multi-Stroke Serif Roman	Multi	Publication-grade journal body text
compital.fnt	Multi-Stroke Serif Italic	Multi	Physical quantities and variables (x, y, z, t)
comscr.fnt	Multi-Stroke Script	Multi	Lagrangian, Hamiltonian, Fourier transformations (L, H)
compgrma.fnt	Complex Greek / Math	Multi	Greek alphabet ($\alpha, \beta, \gamma, \Omega$) and calculus symbols
simgrma.fnt	Simplex Greek / Math	Single	Rapid engineering Greek annotations
simscr.fnt	Simplex Script	Single	Cursive annotations
engoth.fnt	English Gothic (Blackletter)	Multi	Formal certificates, mathematical Lie algebras (g)
microb.fnt	Micro Bold	Double	Compact subscript annotations and high-density labels
microg.fnt	Micro Greek	Single	Compact Greek subscripts and superscripts
russian.fnt	Cyrillic Alphabet	Multi	International Cyrillic text and Slavic language charts
swiss.fnt	Swiss Neo-Grotesque	Multi	Modern Helvetica-style technical graphics
swissbld.fnt	Swiss Bold	Multi	Modern bold headings and presentation banners
swissitl.fnt	Swiss Italic	Multi	Slanted modern clean technical text
swissn.fnt	Swiss Narrow	Multi	High-density tables and congested legends
swissbit.fnt	Swiss Bold Italic	Multi	Heavy slanted titles
matrix.fnt	Dot Matrix	Stippled	Instrument telemetry emulation

Font Filename	Typographic Classification Emulation	Number of Strokes	Primary Use Case
block.fnt	Geometric Block Caps	Filled	Poster headings and architectural labels
news.fnt	Newspaper Century Serif	Multi	High-legibility serif text
newsgrm.fnt	News Greek / Math	Multi	Mathematical typography paired with News serif

10. Master Function Index & Technical Cross-Reference

This comprehensive alphabetical index documents every routine in the GraphiC scientific graphics library. For each function, the table maps the exact calling signature, C source implementation file (Source/*.c), manual section reference with internal anchor hyperlink, multi-language parity status across C, Python 3, Java 22, and Fortran 2003, and operational notes.

Understanding Language Parity and Auxiliary Functions

• **Full Parity (C, Python, Java, Fortran):**Exposed and verified across all four languages with idiomatic procedural and object-oriented wrappers.
• **C / Fortran:**Available in the core C engine and Fortran 2003 ISO C bindings module.
• **C-Only / Core:**Standard procedural C engine API routines.
• **Auxiliary / C-Only (No Direct Bindings):**Low-level C companion routines, internal hardware hooks, mathematical kernels, and memory tuning parameters. These routines are retained in C for low-level engine support, but deliberately have no high-level bindings because modern languages (Python, Java, Fortran) and modern operating systems handle these concerns natively (e.g. automatic garbage collection, dynamic multidimensional arrays, resolution-independent vector units, standard timing libraries, and DOM-based interactive web viewers).

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
`void acrop(char flag)`	Source/SVC.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Configures automatic plot boundary clipping on or off.
`void aname(char *alabel)`	Source/TRIANGLE.C	§7.9 Triangle Plots	**Full Parity (C, Python, Java, Fortran)**	Sets label string for Axis A in ternary triangle phase diagrams.
`void area2d(float xinch, float yinch)`	Source/BPLT.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Defines physical dimensions of 2D plot area in inches.
`void axesoff(int noaxes)`	Source/SVC.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Suppresses drawing of coordinate axes and tick marks.
`void axnamht(float inch)`	Source/GRAPHIC.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Sets text height for coordinate axis titles in inches.
`void bar(int nxdiv, float *x, float *y, int npts, float bwid, int mode, int *car, int *tpe)`	Source/BARPLT.C	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Plots grouped or stacked vertical bar charts with hatching.
`void barchart(int nbars, float *xarray, float *yarray, int npts, float width, int *carray, int *typearray)`	Source/BARPLT.C	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	High-level convenience routine for generating labeled bar charts.

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
`void bcolor(Uchar value)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Sets palette color index for the bottom/underside of 3D surface meshes.
`void bfreeq(void *ptr)`	Source/ memory_portable.c	§3 Memory Model / §7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Low-level heap deallocation kernel with memory block integrity validation. Omitted from high-level bindings as Python, Java, and modern Fortran employ automatic memory management and garbage collection.
`void bgnplot(char mode, char driver, char *tekfile)`	Source/BPLT.C	§7.1 Initialization Routines	**Full Parity (C, Python, Java, Fortran)**	Initializes GraphiC plotting session, allocates display driver and output stream.
`void blank(unsigned char attr)`	Source/BAUX_PORTABLE.C	§7.19 Memory, File I/O & Utility Routines	**C-Only / Core**	Clears terminal console screen buffer with specified ANSI attribute code.
`void bname(char *blabel)`	Source/TRIANGLE.C	§7.9 Triangle Plots	**Full Parity (C, Python, Java, Fortran)**	Sets label string for Axis B in ternary triangle phase diagrams.
`void box(void)`	Source/SVC.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Draws rectangular bounding frame border around the active 2D plot area.
`void box3d(void)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Renders a 3D perspective bounding cuboid enclosure frame.
`void BoxPlot(float x, float *y, int ydim, int outlier, int symnum, float width)`	Source/STATFUNS.C	§7.11 Pie Charts & Statistical Plots	**C-Only / Core**	Generates five-number summary box-and-whisker statistical distribution plots.
`void charspc(int width)`	Source/Ttfntplt.c	§4.4 Typography / §7.15 Text	**Full Parity (C, Python, Java, Fortran)**	Sets inter-character tracking and pitch expansion factor in inches for vector stroke fonts.
`int ci_echoq(void)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Reads single keystroke from console with terminal echo enabled. Auxiliary terminal utility for interactive command prompts.
`int ci_scq(void)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Reads extended keyboard scan code for cursor arrow keys and function keys. Hardware console hook replaced by modern GUI event listeners.
`int ciq(void)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Reads single raw keystroke character from standard input without console echo. Legacy interactive console menu polling; high-level environments utilize non-blocking event loops.
`void CMYto4q(float c, float m, float y, int sq[4])`	Source/Color.c	§7.14 Color & Palettes	**Auxiliary / C-Only (No Direct Bindings)**	Synthesizes a 4-pixel ordered dither pattern for cyan, magenta, and yellow color separation on bilevel devices. Omitted as modern display

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
				pipelines support 24-bit TrueColor rendering natively.
`void cname(char *clabel)`	Source/TRIANGLE.C	§7.9 Triangle Plots	**Full Parity (C, Python, Java, Fortran)**	Sets label string for Axis C in ternary triangle phase diagrams.
`void color(int col)`	Source/LINEATTR.C	§7.14 Color & Palettes	**Full Parity (C, Python, Java, Fortran)**	Sets active drawing palette color index for strokes and text.
`void ColorKeyL(float minv, float maxv, ...)`	Source/CNTPLT.C	§7.8 Contour Plots	**C-Only / Core**	Renders continuous-tone logarithmic color spectrum calibration scale key.
`void conmat(float *zmat, float *x, int nx, float *y, int ny, float orig, float step, float maxq, int *lnstyle, int labint)`	Source/CNTPLT.C	§7.8 Contour Plots	**Full Parity (C, Python, Java, Fortran)**	Generates 2D contour plot directly from a 2D elevation data matrix.
`void contcolor(int *colors, int border, ...)`	Source/CNTPLT.C	§7.8 Contour Plots	**Full Parity (C, Python, Java, Fortran)**	Sets color mapping scheme for contour elevation levels.
`void context(int plot_number)`	Source/SIMPLT.C	§7.4 Simultaneous Plots	**Full Parity (C, Python, Java, Fortran)**	Switches active plotting context among simultaneous composite subplots.
`void cross(int iscross)`	Source/SVC.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Enables drawing of central coordinate axes crossing at the origin (0,0).
`void CrossAt(float x, float y)`	Source/AXESDRAW.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Draws coordinate axes intersecting at a user-specified coordinate (x, y).
`unsigned int cstsq(void)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Polls keyboard input buffer status returning non-zero if a keystroke is waiting. Omitted in favor of host runtime event handling.
`void ctline(char *string, float height)`	Source/FNTPLT.C	§7.15 Text Routines	**Full Parity (C, Python, Java, Fortran)**	Renders centered line of text using active vector stroke font.
`void curson(int srow, int scol, float **fx, float **fy, int *find)`	Source/CURSORW.C	§7.16 Interactive Crosshairs & Digitization	**C-Only / Core**	Activates interactive graphical crosshair digitizer and returns coordinate.
`void curv3d(float *x, float *y, float *z, int npts)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Plots 3D spatial trajectory curve in perspective coordinate space.
`void curve(float *x, float *y, int npts, int sym)`	Source/CURVES.C	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Plots 2D polyline curve with optional line style and symbol markers.
`void curvefollow(int row, int col, float *x, float *y, int npts)`	Source/CURSORW.C	§7.16 Interactive Crosshairs & Digitization	**C-Only / Core**	Interactive crosshair digitizer with snap-to-curve coordinate tracking.

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
`void curvfill(float *x, float *y, int npts, float *x1, float *y1, int npts1, int pcolor, int curv)`	Source/SVC.C	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Fills area beneath or between curves with solid color or hatch pattern.
`void d3base(int isbase, int colorb)`	Source/D3plt.c	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Configures baseline elevation plane and skirt color for 3D perspective surface meshes.
`void d3color(int *array, int nlevels, int mesh)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Configures color elevation mapping palette for 3D surface meshes.
`void d3grid(int grid_flags)`	Source/D3plt.c	§7.10 Three-Dimensional Plots	**Auxiliary / C-Only (No Direct Bindings)**	Draws 3D perspective coordinate grid lines across back and base reference planes. Managed through composite 3D axis drawing options in high-level wrappers.
`void d3head(char *title, int place)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Renders centered perspective title heading above 3D surface plots.
`void d3line(float xf, float yf, float zf, float xt, float yt, float zt)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Draws 3D perspective line between two user space coordinates (x1,y1,z1) to (x2,y2,z2).
`void d3pltfnt(float x, float y, float z, char *string, float height, int angle)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**C-Only / Core**	Selects active font for 3D perspective coordinate axis labels.
`void dashf(int type)`	Source/LINEATTR.C	§7.13 Line and Symbol Styles	**Full Parity (C, Python, Java, Fortran)**	Defines custom user dash-space pattern for line drawing.
`void dateit(Uchar font)`	Source/BPLT.C	§7.2 Plot Termination Routines	**Full Parity (C, Python, Java, Fortran)**	Appends automated timestamp and library version watermark to bottom margin.
`float **dim2(int nr, int nc)`	Source/ memory_portable.c	§3 Memory Model / §7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Allocates dynamic contiguous 2D floating-point matrix with row pointer table in C. Omitted because Python (NumPy/lists), Java (`float[][]`), and Fortran (rank-2 arrays) allocate multi-dimensional arrays natively.
`void ellipse(float xc, float yc, float xr, float yr, float rotation, float th1, float th2, char degrees, int units)`	Source/CIRCLES.C	§7.12 Line-Drawing Routines	**C-Only / Core**	Draws outline of an ellipse or elliptical arc in user coordinates.
`int endplot(void)`	Source/BPLT.C	§7.2 Plot Termination Routines	**Full Parity (C, Python, Java, Fortran)**	Flushes pending graphics vectors, completes frame, and prompts for page advance.
`void erfcurve(float *x, float *y, int npts, int sym)`	Source/ERFPLT.C	§7.11 Pie Charts & Statistical Plots	**C-Only / Core**	Plots standard Gaussian cumulative normal distribution probability curve.

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
`void erfgrid(float ymin, float ystep, float ymax, char *yformat)`	Source/ERFPLT.C	§7.11 Pie Charts & Statistical Plots	**C-Only / Core**	Renders probability grid lines on normal error distribution axes.
`void erfscales(int nydiv, float *y, int npts)`	Source/ERFPLT.C	§7.11 Pie Charts & Statistical Plots	**C-Only / Core**	Establishes probability scaling coordinates for error function plots.
`void errorbar(float x, float xmin, float xmax, float y, float ymin, float ymax)`	Source/CURVES.C	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Draws vertical or horizontal symmetric/asymmetric error bars on data points.
`void fgrid(int fon, int ndiv)`	Source/GRIDOPTS.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Draws full rectangular grid mesh with specified frequency divisions.
`void fillellipse(int onoff, int pattern, int border)`	Source/CIRCLES.C	§7.12 Line-Drawing Routines	**C-Only / Core**	Draws filled solid or patterned ellipse in user coordinates.
`void fntchg(char sym)`	Source/Ttfntplt.c	§4.3 / §4.4 Typography	**Full Parity (C, Python, Java, Fortran)**	Low-level font table switching and Hershey vector stroke typeface selector.
`void font(int nfont, ...)`	Source/TTFNTLD.C	§7.15 Text Routines	**Full Parity (C, Python, Java, Fortran)**	Loads and activates Hershey vector stroke, TrueType, or Type 1 font.
`void frame(int frameon, int ftick)`	Source/SVC.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Draws rectangular bounding frame with inward or outward facing tick marks.
`void free2(float **matrix)`	Source/ memory_portable.c	§3 Memory Model / §7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Deallocates dynamic 2D matrix allocated by `dim2()`. Unnecessary in managed high-level runtimes.
`char *get_time(int timer_id)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Queries elapsed wall-clock execution time string since `set_time()` for benchmarking. High-level runtimes provide native high-precision timers (`perf_counter`, `nanoTime`).
`void GetDataPoint(float *x, float *y)`	Source/Cursorw.c	§7.16 Interactive Crosshairs	**Auxiliary / C-Only (No Direct Bindings)**	Reads single user data coordinate from active digitized crosshair. Replaced by client-side browser interactivity in modern SVG viewer (`viewer.html`).
`void getfilenameq(char *fname)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Parses file path to isolate base root filename and strip directory paths. Standard path libraries (`os.path`, `java.nio.file.Path`) supersede this helper.
`int gfclose(FILE *stream)`	Source/BAUX_PORTABLE.C	§7.19 Memory, File I/O & Utility Routines	**C-Only / Core**	Environment-aware file closing helper ensuring driver output buffer flush.
`FILE *gfopen(char *filename, char *mode, size_t bigbuff)`	Source/BAUX_PORTABLE.C	§7.19 Memory, File I/O & Utility Routines	**C-Only / Core**	Environment-aware file opener searching GRAPHIC directory and system paths.

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
`void gpcBeep(int freq, int duration)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Triggers audible tone alert on terminal console speaker at specified frequency and duration. Hardware-dependent console alert.
`void GPCFreeResourcesq(void)`	Source/ memory_portable.c	§3 Memory Model / §7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Explicitly releases global font tables, polyline buffers, and graphics device memory. Handled automatically on plot termination and process exit.
`int gpcGetCharWidth(int ch)`	Source/Ttfntplt.c	§4.4 Typography	**Auxiliary / C-Only (No Direct Bindings)**	Queries advance width of individual character glyph in active font. Low-level typographic metrics helper; `strwidth()` is preferred.
`void gpcTextMode(int mode)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Switches console between graphics display mode and alphanumeric text mode. Obsolete legacy DOS/Win16 video BIOS interrupt hook.
`void graf(char *xformat, float xori, float xste, float xma, char *yformat, float yori, float yste, float yma)`	Source/LINPLT.C	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Establishes linear 2D scaling and draws numbered coordinate axes.
`void graf3d(float xorig, float xstp, float xmax, float yorig, float ystp, float ymax, float zorig, float zstp, float zmax)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Establishes 3D perspective projection scaling and draws coordinate axes.
`void GRCtoIBM(int col, int *r, int *g, int *b, int *ibmc)`	Source/Color.c	§7.14 Color & Palettes	**Auxiliary / C-Only (No Direct Bindings)**	Translates GraphiC palette index to nearest IBM 16-color palette RGB representation. Legacy color quantization mapping.
`void grid(int gridon)`	Source/GRIDOPTS.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Draws standard Cartesian grid lines aligned with major axis tick marks.
`void GTfont(int id)`	Source/Ttfntplt.c	§4.5 GrafText PostScript	**Auxiliary / C-Only (No Direct Bindings)**	GrafText standard Adobe PostScript Level 1 printer font selector. Obsolete 30-font printer driver hook superseded by TrueType and modern SVG typography.
`void heading(char *title)`	Source/FNTPLT.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Renders centered publication title heading above the active plot area.
`void histogram(float *x, float *y, int npts, int type, int *colors)`	Source/BARPLT.C	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Constructs statistical frequency distribution bar histogram.
`void HLStoRGB(int h, int l, int s, int *r, int *g, int *b)`	Source/COLOR.C	§7.14 Color & Palettes	**C-Only / Core**	Converts Hue-Lightness-Saturation color coordinates to RGB color triplet.
`long HowMuchMem(void)`	Source/ memory_portable.c	§3 Memory Model / §7.19	**Auxiliary / C-Only (No**	Interrogates heap memory availability for dynamic array

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
		Utilities	**Direct Bindings)**	allocations. Legacy 16-bit conventional memory check rendered obsolete by 64-bit virtual memory.
`int intopix(float inches)`	`Source/SVC.C`	§7.19 Memory, File I/O & Utility Routines	**Full Parity (C, Python, Java, Fortran)**	Converts physical page inches to device raster pixels.
`void lconmat(float *zmat, float *x, int nx, float *y, int ny, float *level, int *lnstyle, int nlevel, int labint, char *formt)`	`Source/CNTPLT.C`	§7.8 Contour Plots	**C-Only / Core**	Generates 2D contour plot with logarithmic elevation contour intervals.
`void lcontour(GPC_FUNQ fun, float *x, int nx, float *y, int ny, float *level, int nlevel, int *lnstyle, int labint, char *format)`	`Source/CNTPLT.C`	§7.8 Contour Plots	**C-Only / Core**	Plots logarithmic elevation contour lines from irregularly scattered data points.
`void legend(int marker, char *string, int style, float strhgt, int scolor)`	`Source/LEGEND.C`	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Adds an entry (line style, symbol marker, label text) to plot legend.
`void legpos(int entries, float xleg, float yleg, int border)`	`Source/LEGEND.C`	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Sets physical location, entry count, and border style for plot legend box.
`void lglgscle(float *x, float *y, int npts)`	`Source/LOGPLT.C`	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Establishes logarithmic scaling on both horizontal and vertical axes.
`void linesp(float ratio)`	`Source/Ttfntplt.c`	§4.4 Typography	**Full Parity (C, Python, Java, Fortran)**	Sets line-spacing leading between multi-line text strings in inches.
`void lmargin(float marginch)`	`Source/Ttfntplt.c`	§4.4 Typography	**Full Parity (C, Python, Java, Fortran)**	Sets left bounding margin offset in inches for multi-line text blocks.
`void loglog(float xorig, float xmax, float yorig, float ymax)`	`Source/LOGPLT.C`	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Draws full log-log coordinate axes with multi-decade logarithmic grid.
`void ltline(char *string, float height)`	`Source/FNTPLT.C`	§7.15 Text Routines	**Full Parity (C, Python, Java, Fortran)**	Renders left-justified line of text using active vector stroke font.
`void matinverse(double **a, double **y, int n)`	`Source/MATSOLVE.C`	§7.17 Spline Interpolation & Numerical Methods	**C-Only / Core**	Calculates matrix inverse using Gauss-Jordan elimination with partial pivoting.
`void matsolve(double **a, int n, double *b)`	`Source/MATSOLVE.C`	§7.17 Spline Interpolation & Numerical Methods	**C-Only / Core**	Solves linear system of equations Ax = b using LU decomposition.
`float mean(float *x, int xdim)`	`Source/STATFUNS.C`	§7.11 Pie Charts & Statistical Plots	**C-Only / Core**	Calculates arithmetic mean of floating-point data array.

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
`float median(float *x, int xdim)`	Source/STATFUNS.C	§7.11 Pie Charts & Statistical Plots	**C-Only / Core**	Calculates median value of floating-point data array.
`void minmax(float *minq, float *maxq, float *xy, int npts)`	Source/ROUNDOFF.C	§7.19 Memory, File I/O & Utility Routines	**C-Only / Core**	Scans array for extrema and calculates rounded 'nice' scaling limits.
`void NewFile(char *NewName)`	Source/BPLT.C	§7.1 Initialization Routines	**Full Parity (C, Python, Java, Fortran)**	Redirects subsequent graphics vector output stream to a new file destination.
`void nohide(int visible)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Disables hidden-line removal for transparent wireframe mesh rendering.
`void numht(float inch)`	Source/GRAPHIC.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Sets character height for axis numeric tick labels in inches.
`void openoffq(char *fname)`	Source/baux_portable.c	§4.1 / §7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Opens `.OFF` character offset index table for Hershey vector stroke fonts. Low-level binary font loader primitive.
`void orel(float xorel, float yorel)`	Source/SVC.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Sets relative origin offset between multiple subplots in inches.
`void panel(float *x, float *y, int nlines, int pcolor, int border)`	Source/SVC.C	§7.18 Polygons & Panel Filling	**Full Parity (C, Python, Java, Fortran)**	Fills an arbitrary closed polygonal boundary with solid color or pattern.
`void panelinch(...)`	Source/SVC.C	§7.18 Polygons & Panel Filling	**Full Parity (C, Python, Java, Fortran)**	Fills closed polygon defined in absolute physical page inches.
`void panproc(void)`	Source/Panfil.c	§7.18 Polygons & Panel Filling	**Full Parity (C, Python, Java, Fortran)**	Flushes accumulated polygon fill panels to active graphics driver pipeline.
`void pgshift(float xshift, float yshift)`	Source/BPLT.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Translates physical plotting origin across page surface in inches.
`void physor(float xphys, float yphys)`	Source/SVC.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Establishes physical origin coordinates (bottom-left corner) on the page in inches.
`void pieplot(int dim, float *seg, int *color, float *offset, char *caption[], float size, int border, int txtclr)`	Source/PIEPLT.C	§7.11 Pie Charts & Statistical Plots	**Full Parity (C, Python, Java, Fortran)**	Draws proportional 2D pie chart with exploded sector offsets and labels.
`float pixtoin(int pixels)`	Source/SVC.C	§7.19 Memory, File I/O & Utility Routines	**Full Parity (C, Python, Java, Fortran)**	Converts device raster pixels to physical page inches.
`void pixtouser(int x_in, int y_in, float *x_out, float *y_out)`	Source/SVC.C	§7.19 Memory, File I/O & Utility Routines	**Full Parity (C, Python, Java, Fortran)**	Maps screen raster pixel coordinates back to floating-point user data space.
`void plane3d(float x1, float y1, float z1, float x2, float y2,`	Source/D3plt.c	§7.10 Three-Dimensional Plots	**Auxiliary / C-Only (No Direct Bindings)**	Renders an arbitrary 3D reference planar polygon in perspective coordinates. Specialized geometric primitive

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
`float z2, float x3, float y3, float z3, float x4, float y4, float z4, int col)`				for 3D floor/ceiling boundaries.
`void plotabsolute(int set_flag)`	Source/SVC.C	§7.4 Simultaneous Plots	**Full Parity (C, Python, Java, Fortran)**	Moves pen cursor to absolute device pixel coordinate.
`void plotadd(void)`	Source/SVC.C	§7.2 Plot Termination Routines	**Full Parity (C, Python, Java, Fortran)**	Enables multi-pass vector layering onto existing plot frame without clear.
`void plotrho(float rho, float theta)`	Source/SMITH.C	§7.6 Smith Charts	**Full Parity (C, Python, Java, Fortran)**	Plots complex reflection coefficient curve on microwave Smith chart.
`void plotrx(float r, float x)`	Source/SMITH.C	§7.6 Smith Charts	**Full Parity (C, Python, Java, Fortran)**	Plots complex impedance/admittance locus curve on microwave Smith chart.
`void pltfnt(float x, float y, char *string, float height, int angle)`	Source/TTFNTLD.C	§7.15 Text Routines	**Full Parity (C, Python, Java, Fortran)**	Selects active font for plot labels, headings, and tick marks.
`void polcurve(float *r, float *theta, int npts, int sym)`	Source/POLPLT.C	§7.7 Polar Plots	**Full Parity (C, Python, Java, Fortran)**	Plots data points in polar coordinates (radius, angle) on polar grid.
`void polgrid(float rmax, float rmin, int mode, char *larray[], int dim, int ntheta, int col)`	Source/POLPLT.C	§7.7 Polar Plots	**Full Parity (C, Python, Java, Fortran)**	Draws circular polar coordinate grid with radial range and angular rays.
`void prtfnt(float xinch, float yinch, char *string, float height, int angle)`	Source/TTFNTLD.C	§7.15 Text Routines	**Full Parity (C, Python, Java, Fortran)**	Selects active font for hardcopy printer output device drivers.
`void putlabel(char *str, float xpos, float ypos, float height, int border, int units)`	Source/Ttfntplt.c	§7.15 Text Routines	**Full Parity (C, Python, Java, Fortran)**	Draws text string at absolute coordinates with directional alignment (0=left, 1=center, 2=right).
`void rfit(float *x, float *y, int npts, int nits, float sfact, int nout, float *xout, float *yout)`	Source/RFIT.C	§7.17 Spline Interpolation & Numerical Methods	**C-Only / Core**	Fits smoothed bivariate surface to irregularly spaced 3D data points.
`int RGBcolor(int r, int g, int b)`	Source/LINEATTR.C	§7.14 Color & Palettes	**C-Only / Core**	Defines 24-bit TrueColor RGB stroke color directly.
`void RGBtoCMY(float r, float g, float b, float *c, float *m, float *y)`	Source/Color.c	§7.14 Color & Palettes	**Auxiliary / C-Only (No Direct Bindings)**	Converts 24-bit RGB values to subtractive Cyan-Magenta-Yellow color separations. Specialized color model conversion for color printing.
`void RGBtoCMYK(int r, int g, int b, int *c, int *m, int *y, int *k)`	Source/COLOR.C	§7.14 Color & Palettes	**C-Only / Core**	Converts 24-bit RGB values to 4-color process CMYK printing separations.
`void RGBtoHLS(int r, int`	Source/COLOR.C	§7.14 Color &	**C-Only / Core**	Converts RGB color triplet to

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
`g, int b, int *h, int *l, int *s)`		Palettes		Hue-Lightness-Saturation color space.
`int RGBtoIBM(int rgb[3])`	Source/COLOR.C	§7.14 Color & Palettes	**C-Only / Core**	Maps 24-bit RGB color to nearest standard 16-color IBM palette index.
`void rmargin(float marginch)`	Source/Ttfntplt.c	§4.4 Typography	**Full Parity (C, Python, Java, Fortran)**	Sets right bounding margin offset in inches for multi-line text blocks.
`void rotate(char isrot)`	Source/BOTHPLT.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Rotates page orientation by 90 degrees between portrait and landscape.
`void rowcolq(int row, int col)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Positions text cursor in terminal console window at (row, col) using ANSI escape sequences. Console layout utility.
`void rtline(char *string, float height)`	Source/FNTPLT.C	§7.15 Text Routines	**Full Parity (C, Python, Java, Fortran)**	Renders right-justified line of text using active vector stroke font.
`void saveplotq(char *name)`	Source/Simplt.c	§7.4 Simultaneous Plots	**Auxiliary / C-Only (No Direct Bindings)**	Serializes current plot environment and subplot transformation matrix to disk. Low-level binary state checkpointing.
`void scales(int nxdiv, int nydiv, float *x, float *y, int npts)`	Source/LINPLT.C	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Establishes user data to physical inch scaling ratios for 2D axes.
`void scatplot(char dot)`	Source/SVC.C	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Generates 2D scatter plot with specified symbol markers and no connecting lines.
`void sclfix(int isfixed)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Locks isometric 1:1:1 aspect ratio scaling across all 3D perspective axes.
`void scr_aputsq(char *str, int attr)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Directly writes styled string with video color attributes to console screen buffer. Low-level direct video memory write.
`void screentype(int mode)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Configures hardware display adapter and screen resolution mode. Obsolete DOS/VGA display adapter selector.
`void set_time(int timer_id)`	Source/baux_portable.c	§7.19 Utilities	**Auxiliary / C-Only (No Direct Bindings)**	Initializes high-resolution wall-clock timer benchmark for performance profiling. Replaced by standard runtime profiling tools.
`void settolerance(float tolerance)`	Source/R3PLT.C	§7.19 Memory, File I/O & Utility Routines	**Full Parity (C, Python, Java, Fortran)**	Sets numerical convergence tolerance and collinearity threshold for triangulation.
`void SetTTFull(int full)`	Source/Truetype.c	§4.2 / §4.4 Typography	**Auxiliary / C-Only (No Direct Bindings)**	Toggles full 256-glyph pre-caching vs. lazy on-demand character decoding in TrueType engine. Internal memory optimization knob.

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
`void SetTTKerning(int on_off)`	Source/Truetype.c	§4.2 / §4.4 Typography	**Auxiliary / C-Only (No Direct Bindings)**	Toggles pairwise TrueType kerning table evaluation. Internal typography engine flag; modern SVG renderers perform kerning automatically.
`void SetTTMaxError(float err)`	Source/Truetype.c	§4.2 / §4.4 Typography	**Auxiliary / C-Only (No Direct Bindings)**	Configures quadratic B-spline chordal tessellation tolerance for TrueType glyph outlines. Internal numerical curve tessellation tolerance.
`void SetTTUseMemory(int flag)`	Source/Truetype.c	§4.2 / §4.4 Typography	**Auxiliary / C-Only (No Direct Bindings)**	Configures heap memory caching for TrueType font file buffers versus disk streaming. Internal I/O caching flag.
`void simuplot(int numplots)`	Source/SIMPLT.C	§7.4 Simultaneous Plots	**Full Parity (C, Python, Java, Fortran)**	Initializes simultaneous multi-panel composite plot layout subsystem.
`void skew(float deg)`	Source/Ttfntplt.c	§4.4 Typography / §7.15 Text	**Full Parity (C, Python, Java, Fortran)**	Sets oblique italic slant angle in degrees for vector stroke fonts.
`void smdraw(Uchar lcolor)`	Source/SMITH.C	§7.6 Smith Charts	**Full Parity (C, Python, Java, Fortran)**	Draws RF microwave impedance Smith chart with resistance and reactance circles.
`void smithcir(float u, float v, float radius)`	Source/SMITH.C	§7.6 Smith Charts	**Full Parity (C, Python, Java, Fortran)**	Draws constant resistance or constant reactance circle on Smith chart.
`void smove(float dell)`	Source/SMITH.C	§7.6 Smith Charts	**Full Parity (C, Python, Java, Fortran)**	Moves pen to coordinate in microwave Smith chart coordinate space.
`void spline(float *x, float *y, int n, int n1, float s1, float s2, float *xn, float *yn)`	Source/SMOOTH.C	§7.17 Spline Interpolation & Numerical Methods	**Full Parity (C, Python, Java, Fortran)**	Calculates cubic spline interpolation curve passing through data points.
`void stack(char stacked)`	Source/CNTPLT.C	§7.8 Contour Plots	**Full Parity (C, Python, Java, Fortran)**	Draws stacked 3D contour elevation slices with perspective separation.
`void startplot(int bcolor)`	Source/BPLT.C	§7.1 Initialization Routines	**Full Parity (C, Python, Java, Fortran)**	Begins new page or frame and clears display buffer with background color.
`float StdDev(float *x, int xdim)`	Source/STATFUNS.C	§7.11 Pie Charts & Statistical Plots	**Full Parity (C, Python, Java, Fortran)**	Computes sample standard deviation of floating-point data array.
`void stopplot(void)`	Source/BPLT.C	§7.2 Plot Termination Routines	**Full Parity (C, Python, Java, Fortran)**	Terminates plotting session, closes device drivers, and releases resources.
`float strwidth(char *string, float height)`	Source/FNTPLT.C	§7.15 Text Routines	**Full Parity (C, Python, Java, Fortran)**	Computes physical width of a text string in inches using active font.
`void fntslant, fntthick, supsym, subsym()`	Source/FNTPLT.C	§7.15 Text Routines	**Full Parity (C, Python, Java, Fortran)**	Sets vertical baseline downward shift fraction for subscript typesetting.
`void supsym(char ch)`	Source/Ttfntplt.c	§4.3 / §4.4	**Full Parity (C,**	Sets vertical upward baseline

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
		Typography	**Python, Java, Fortran)**	shift fraction for superscript typesetting.
void surfun(GPC_FUNQ zfun, int ixpts, float xdel, int iypts, float ydel)	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Plots 3D perspective surface from continuous mathematical function z = f(x, y).
void surmat(float *zmat, int xdim, int ydim)	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Plots 3D perspective surface with hidden-line removal from 2D elevation grid.
void surmat2(float **zmat, int xdim, int ydim)	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Plots 3D perspective surface with dual top/bottom independent color shading.
void survis(char *str)	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Sets 3D surface mesh face visibility mode ('top', 'bottom', or 'both').
void symbol(float x, float y, int symnum)	Source/SYMBOLS.C	§7.13 Line and Symbol Styles	**Full Parity (C, Python, Java, Fortran)**	Plots single centered symbol marker glyph at user coordinate (x, y).
void sympick(int symbol)	Source/SYMBOLS.C	§7.13 Line and Symbol Styles	**Full Parity (C, Python, Java, Fortran)**	Selects standard marker glyph shape (circle, square, triangle, diamond, cross).
void tcolor(Uchar value)	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Sets palette color index for the top (upper) face of 3D surface meshes.
void tcurve(int width)	Source/SVC.C	§7.13 Line and Symbol Styles	**Full Parity (C, Python, Java, Fortran)**	Plots polyline curve using absolute physical inch coordinates.
void tfont(int wid)	Source/Ttfntplt.c	§4.4 Typography / §7.15 Text	**Full Parity (C, Python, Java, Fortran)**	Sets stroke pen width in device raster pixels for vector fonts.
void tickht(float height)	Source/GRIDOPTS.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Sets length of coordinate axis tick marks in inches.
void tickout(int is_out)	Source/GRIDOPTS.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Toggles whether axis tick marks point outwards away from plot or inwards.
void tifont(float wid)	Source/Ttfntplt.c	§4.4 Typography / §7.15 Text	**Full Parity (C, Python, Java, Fortran)**	Sets stroke pen width in physical inches for vector fonts.
void titlht(float inch)	Source/GRAPHIC.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Sets character height for main plot title heading in inches.
void tline(int width)	Source/SVC.C	§7.13 Line and Symbol Styles	**Full Parity (C, Python, Java, Fortran)**	Draws straight line segment between two coordinates in physical inches.
void tmargin(float margin)	Source/Ttfntplt.c	§4.4 Typography	**Full Parity (C, Python, Java, Fortran)**	Sets vertical margin spacing above plot titles in inches.
void trifun(GPC_FUNQ fun, int dim, float zmin, float zstep, float	Source/TRIANGLE.C	§7.9 Triangle Plots	**Full Parity (C, Python, Java, Fortran)**	Plots ternary phase surface from continuous function over triangular domain.

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
`zmax, int *lnstyle, int border)`				
`void trimat(...)`	Source/TRIANGLE.C	§7.9 Triangle Plots	**Full Parity (C, Python, Java, Fortran)**	Plots ternary phase equilibrium diagram with triangular coordinate grid.
`void upright(char uplab)`	Source/SVC.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Forces numeric tick labels on vertical Y-axis to be drawn horizontally.
`void usertoinch(float x_in, float y_in, float *x_out, float *y_out)`	Source/SVC.C	§7.19 Memory, File I/O & Utility Routines	**Full Parity (C, Python, Java, Fortran)**	Converts user floating-point data coordinates to absolute page inches.
`void usertopix(float x_in, float y_in, float z_in, int *x_out, int *y_out)`	Source/SVC.C	§7.19 Memory, File I/O & Utility Routines	**Full Parity (C, Python, Java, Fortran)**	Converts user floating-point data coordinates to device raster pixels.
`void uvector(float xfrom, float yfrom, float xto, float yto, float ltow, float size, char *type)`	Source/ARROW.C	§7.12 Line-Drawing Routines	**Full Parity (C, Python, Java, Fortran)**	Draws directional vector arrow between two user space data coordinates.
`float valueq(float n1, float n2, float hue)`	Source/Color.c	§7.14 Color & Palettes	**Auxiliary / C-Only (No Direct Bindings)**	Internal chromatic conversion kernel for HLS-to-RGB color transformation.
`void vector(...)`	Source/ARROW.C	§7.12 Line-Drawing Routines	**Full Parity (C, Python, Java, Fortran)**	Draws directional vector arrow between two coordinates in physical inches.
`void view(float a, float b, float c)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Establishes custom 4x4 perspective transformation matrix for 3D viewing.
`void volm3d(float x3axis, float y3axis, float z3axis)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Defines physical dimensions of 3D perspective bounding cube in inches.
`void vuabs(float xvu, float yvu, float zvu)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Sets 3D perspective viewpoint using spherical coordinates (radius, theta, phi).
`void vuangl(float phi, float theta, float radius)`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Sets 3D perspective viewpoint elevation and azimuth angles in degrees.
`void d3bars, plane3d, WaterfallPlot()`	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Plots stacked cascade of spectral waterfall curves with perspective offset.
`void wedge(float rmin, float rmax, float thmin, float thmax, int pattern, int border)`	Source/POLPLT.C	§7.7 Polar Plots	**Full Parity (C, Python, Java, Fortran)**	Draws solid or patterned angular sector wedge on polar coordinate plot.
`void widen(float factor)`	Source/Ttfntplt.c	§4.4 Typography / §7.15 Text	**Full Parity (C, Python, Java, Fortran)**	Sets horizontal glyph width expansion/compression multiplier for vector fonts.
`void x3name(char *xname)`	Source/D3PLT.C	§7.10 Three-Dimensional	**Full Parity (C, Python, Java,**	Sets label string for X-axis in 3D perspective plots.

Function & Signature	Source File	Manual Section	Binding Status	Purpose & Architectural Rationale
		Plots	**Fortran)**	
void xfgrid(int fon, int nx)	Source/LINPLT.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Draws vertical grid lines with custom sub-interval division counts.
void xlab(char on_off, char **strlab)	Source/LINPLT.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Assigns alphanumeric string labels to individual X-axis tick positions.
void xlog(float xorig, float xmax, char *format, float yorig, float ystep, float ymax)	Source/LOGPLT.C	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Draws semi-logarithmic axes with logarithmic X-axis and linear Y-axis.
void xname(char *xlabel)	Source/FNTPLT.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Sets descriptive label string for horizontal X-axis.
void y3name(char *yname)	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Sets label string for Y-axis in 3D perspective plots.
void yfgrid(int fon, int ny)	Source/LINPLT.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Draws horizontal grid lines with custom sub-interval division counts.
void ylab(char on_off, char **strlab)	Source/LINPLT.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Assigns alphanumeric string labels to individual Y-axis tick positions.
void ylog(char *format, float xorig, float xstep, float xmax, float yorig, float ymax)	Source/LOGPLT.C	§7.5 2-D Axes Routines	**Full Parity (C, Python, Java, Fortran)**	Draws semi-logarithmic axes with linear X-axis and logarithmic Y-axis.
void yname(char *ylabel)	Source/FNTPLT.C	§7.3 Page and Axis Options	**Full Parity (C, Python, Java, Fortran)**	Sets descriptive label string for vertical Y-axis.
void z3name(char *zname)	Source/D3PLT.C	§7.10 Three-Dimensional Plots	**Full Parity (C, Python, Java, Fortran)**	Sets label string for vertical Z-axis in 3D perspective plots.