Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

M. Bakker

At last an ISO C binding of GKS

Computer Science/Department of Interactive Systems Report CS-R9022 June

At Last an ISO C Binding of GKS

Miente Bakker

Centre for Mathematics and Computer Science
Sector for Technical Support
Kruislaan 413
1098 SJ Amsterdam
The Netherlands
miente@cwi.nl

ABSTRACT

The C[4] bindings of GKS[1] and other semantic computer graphics standards are long overdue. While GKS was completed in 1985 and GKS–3D[2] (and PHIGS[3]) became international standard in 1988, none of their C bindings could be standardized, for the simple reason that the C language itself was not a standard. Instead, a host of de facto GKS/C bindings appeared.
This paper will give the flavour of the ISO C binding of GKS; the main features will be outlined.
1983 CR Categories: D.3.0, I.3.0, I.3.4.
Keywords & Phrases: computer graphics standardization, GKS, GKS-3D, PHIGS, language binding.
Note: the present text will be submitted to *Computer Graphics Forum*.

1. Introduction

When GKS was completed in 1985, its bindings in Fortran 77, Pascal and Ada were reasonably stable. Not so with GKS/C. Because the C language was only a de facto standard [8], which was slowly being standardized within ANSI and ISO, the designers of GKS/C and other C bindings of computer graphics standards could only patiently wait until C had reached another milestone, before they could make another step with their binding. The predictable result was a host of mutually incompatible de facto C bindings of GKS (see [9]). Applications running on such a C implementation could not run on other C implementations.

This unacceptable situation started to come to an end in December 1989, when C became Draft International Standard (DIS), the next-to-last version before International Standard. Within months, GKS/C and GKS-3D/C also became DIS. Completion of these bindings is expected in the end of 1991.

The following aspects of GKS/C are highlighted:

- The data types used;
- The use of macros;
- The function interface;
- The #include file gks.h;
- Memory management;
- User error handling;
- Compatibility with PHIGS/C and other GKS bindings.

2. Mapping of GKS Entities to C Data Types

C is a rich programming language, which is very suitable for the implementation of GKS. Some main reasons for implementing GKS in C are:

- The allocation and deallocation capabilities;
- The capability of defining structured data types;
- The #include mechanism.

Hence it seemed obvious that the designers of GKS/C within ISO (henceforth called the binders) would use the richness of C to the full for their project. That this did not happen had a number of reasons:

- The possibility of simulating other GKS

1985	1st Working Draft (WD) of GKS/C[5]
	GKS International Standard
1986	2nd WD of GKS/C
1988	Third WD of GKS/C
	1st WD of GKS–3D/C[6]
	1st Draft Proposal (DP) of C
	1st DP of PHIGS/C[7]
	GKS-3D International Standard
1989	DPs of GKS/C and GKS-3D/C
	C Draft International Standard (DIS)
	2nd DP of PHIGS/C
1990	DISs of GKS/C and GKS-3D/C

Table 1. Chronology of GKS/C

bindings by putting cross-language interfaces on top of GKS/C required the use of simple C data types;

- Part of the C programming community preferred simple data types in GKS/C;
- The use of deeply nested data types would decrease the performance of GKS/C.

As a result of these trade-offs, a moderate use of the C structuring capabilities was made for the design of GKS/C. The following conventions were made :

- GKS data types are mapped directly to similar C data types (see table 2);
- GKS data that are related are packed into some structured data type (see table 3);
- As much as possible, generic types are used; for instance, for the clipping rectangle, the workstation window, the viewport, etc., the same data type Glimit is used and for points in WC space, NDC space and DC space, the data type Gpoint is used;
- Structures should not become too long and not too deeply nested;
- Name convention: all structure names start with a capital G and are further in lower case; all fields of enumeration types are in upper case and start with capital G; in order to avoid syntactical problems, each enumeration field is given a prefix.

This means for instance that all the components of the POLYLINE FACILITIES (the available linetypes, linewidths and predefined line colours) are packed into the structure Gline_facs but the LOCATOR DEVICE STATE is not wrapped into some structure, because the components are not related and because that structure would become too complex.

GKS		C binding
I	integer	Gint
R	**real**	**Gfloat**
		Gdouble
S	**string**	**char ***
P	2D point	Gpoint
P3	**3D point**	**Gpoint3**
L	list of 2D points	Gpoint_list
L3	**list of 3D points**	**Gpoint_list3**
V	2D vector	Gvec
V3	**3D vector**	**Gvec3**
N	name	Gint
		char *
E	enumeration type	typedef enum
CLR	**colour bundle**	**Gcolr_rep**

Table 2. Mapping of GKS data types

3. Use of macros

The C language provides the possibility of macros. For the C binding of GKS, these macros are used for constants like:

- Error numbers;
- Function names for the error handler;
- GKS Metafile Item types;
- Registered graphical items, like linetypes, prompt and echo types.

4. The GKS/C Function Interface

In the C binding of GKS almost each GKS function has been mapped to a C function of type void. **The name of this C function is some abbreviation of the GKS function with blanks replaced by underscores. It is written in lower case and it always has the sentinel g. For instance, INQUIRE MAXIMUM NORMALIZATION TRANSFORMATION NUMBER is represented by the C function ginq_max_norm_tran_num.**

The parameters of the GKS/C functions obey the following rules:

- They are prototyped;
- Output parameters are of pointer type;
- Structured input parameters are of type const Gtype *, hence they cannot be changed by GKS/C.

For instance, the C binding for SET COLOUR REPRESENTATION would be

Basic Types	

typedef int Gint;
typedef float Gfloat;
typedef double Gdouble;
typedef void *Gstore;	

Enumeration Types	

Aspect Source Flag	**typedef enum {
	GASF_BUNDLED,
	GASF_INDIV
	} Gasf;**

Fill Area Interior Style	**typedef enum { GSTYLE_HOLLOW, GSTYLE_SOLID, GSTYLE_PAT, GSTYLE_HATCH } Gfill_int_style;**
Structured Types	

Point	typedef struct {
	Gfloat x, y;
	} Gpoint;

Polyline Bundle	**typedef struct {
	Gint type;
	Gdouble width;
	Gint colr_ind;
	} Gline_bundle;**

Pattern Bundle	typedef struct {
	Gint_size dims;
	Gint *colr_array;
	} Gpat_rep;

Implementation Dependent Types	

Choice Data Record	**typedef union {
	struct {
	impl. dep.
	} pet_r1;
	...
	struct {
	impl. dep.
	} pet_r4;
	} Gchoice_data;**

GKSM Item Data Record	**typedef struct {
	Gint type;
	Gint length;
	union {
	impl. dep.
	}
	} Gitem_data;**

Table 3. Examples of GKS/C data types

```
void gset_colr_rep(
        Gint                ws_id,
        Gint                colr_ind,
        const Gcolr_rep     *colr_rep
);
```

5. The #include file gks.h

Each GKS/C application program must include the file gks.h. This file contains the following information:

- All GKS/C data types;
- All GKS/C macros;
- All GKS/C functions defined as extern void;
- Implementation dependent information, for instance macros for the available workstation types, available text fonts, prompt and echo types, etc.

```
#include "gks.h"
#define MY_WS (1)
main ()
{
        gopen_gks(GKS_ERR_FILE,
          (size_t)(-1));
        gopen_ws(MY_WS, WS_CONN_1,
          WS_TYPE_1);
        gactivate_ws(MY_WS);
        gdeactivate_ws(MY_WS);
        gclose_ws(MY_WS);
        gclose_gks();
}
```

Table 4. Simple program example

6. Memory Management

Several GKS inquire functions (and also the ESCAPE function) return information of *a priori* unknown length to the application. The crucial question for the binders was: who allocates the workspace for the information, the application or GKS? The answer was: the application, but this still left some problems unsolved, for instance:

- How is the application notified, if too little workspace has been allocated?;
- Should GKS/C support the facility to return only slices of information to the application?

This issue was resolved the following way:

- GKS inquire functions that return simple lists (lists of segment names, polyline indices, workstation types) to the application are bound in GKS/C to functions that return slices of fixed length to the application; In the following sample progarm, INQUIRE LIST OF POLYLINE INDICES is returning the 3nd-12th defined polyline index from the workstation state list:

```
#
Gint_list    line_inds;
Gint         length;
line_inds.num_ints = 10;
line_inds.ints =
             (Gint *)callac(10,
             sizeof(Gint));
#
ginq_list_line_inds(...,
             2, 10,...,
             &line_inds, &length);
#
```

The output parameter length denotes the actual number of defined polyline indices;

- For GKS functions which return more complex data of unknown length to the application, storage is allocated and deallocated by two special utility functions: CREATE STORE and DELETE STORE. CREATE STORE allocates a pointer to workspace which can be further filled by the GKS/C function. DELETE STORE deallocates all the allocated workspace.
  A typical sample program would be

```
#
Gstore       store;
Gpat_rep     *pat_rep;
#
gcreate_store(&err_ind, &store);
ginq_pat_rep(ws_id, ..., &store,
             &err_ind, &pat_rep);
#
gdel_store(store);
#
```

7. User Error Handling

GKS defines the facility for the application programmer to use his own error handler, which replaces the GKS error handler. In practice, inconvenient linking problems occur, when a user-written error handler has to replace the implementation's error handler of the same name. Therefore GKS/C has introduced the utility function

```
void gset_err_hand(
             const void (*new_hand),
             void (**old_hand));
```

which transfers the error handling from the error handler old_hand() to the customer's error handler new_hand(). As a service, the pointer to the previous error handler is returned to the application for future use.

8. Compatibility with PHIGS/C and other GKS bindings

A crucial issue in the design of GKS/C was the compatibility with other GKS bindings and the compatibility with PHIGS/C.

The compatibility with e.g. GKS Fortran was motivated by the wish to build Fortran shells on top of GKS/C and vice versa (see e.g. [10]). As a result of this compatibility issue, only modest use was made of the structuring capabilities of C. Other features of GKS/C that result from the compatibility with GKS/Fortran are the numeric values of the enumeration types in GKS/Fortran and the separation of the mandatory part and the optional part of input data records in some inquire functions.

The compatibility with PHIGS/C was a logical consequence of the compatibility of GKS-3D and PHIGS, which was expressed by a GKS-3D shell on top of PHIGS (see [11]). The main results of this compatibility were:

- Data types which stem from similar GKS and PHIGS data (e.q. enumerations, rectangles, point lists) are identical in GKS/C and PHIGS/C, except for the sentinels.
- Functions with identical name in GKS and PHIGS are identical in GKS/C and PHIGS/C, except for the sentinels.

References

1. ISO 7942 1985(E) - Graphical Kernel System (GKS) - functional description.
2. ISO 8805 1988(E) - Graphical Kernel System for Three Dimensions (GKS-3D) - functional description.
3. ISO/IEC 9592-1 1989(E) - The Programmer's Hierarchical Interactive Graphics System (PHIGS)
   - Part 1: functional description.
4. ISO/IEC 9899 199x(E), Programming Languages - C;
   Draft International Standard, December 1989.
5. ISO/IEC 8651-4 - 199x(E) - Graphical Kernel

System (GKS) - language bindings - part 4: C;
Draft Proposal, April 1989;
Draft International Standard, to appear in August 1990.

6. ISO/IEC 8806-4 199x(E) - Graphical Kernel System for Three Dimensions (GKS-3D) - language bindings - part 4: C;
Draft Proposal, April 1989;
Draft International Standard, to appear in August 1990.
7. ISO/IEC 9593-4 199x(E) - The Programmer's Hierarchical Interactive Graphics System (PHIGS) - language bindings - part 4: C;
Second Draft Proposal, November 1989.
8. B.W. Kernighan, D.M. Ritchie, *The C programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1978.
9. K.M. Wyrwas, W.T. Hewitt, *A Survey of GKS and PHIGS Implementations October 1988*, Computer Graphics Forum, **8** (1), pp. 49-59, March 1989.
10. S.I. Feldman, P.J. Weinberger, *A Portable Fortran 77 Compiler*, UNIX Programmer's Supplementary Documents Volume 1 (PS1), 4.3 Berkeley Software Distribution, Virtual VAX-11 Version, 1986.
11. P.J.W. ten Hagen, L.R.H. Kessener, B.P. Rouwhorst, *A GKS Shell for PHIGS Implementations*, ISO TC 97 SC 21 WG 2 N541, March 1987.