объединенный
институт
ядерных
исследований
дубна

E11-83-393

M.Rudalics*

AN INTELLIGENT GRAPHICS TERMINAL'S INTERMEDIATE DATABASE

Submitted to Symposium "Eurographics'83"
(Zagreb, Yugoslavia, 1983)

---

* On leave from Johannes Kepler University, Linz, Austria.

1983

1. INTRODUCTION

A database has to provide[5]: (1) A concise and understandable representation of the state of a system at any time, (2) efficient and high-level access to information, and (3) simple and consistent update as the system changes state. The Intelligent Graphics Terminal's Intermediate Database (INGRID) is an attempt to meet these requirements for the Graphical Kernel System - GKS [1,2], by defining a minimal set of data types for realizing on an intelligent graphics workstation:

- A "Workstation Dependent Segment Storage", consisting of segments, instances of segments, and s_functions, where latter reflect the concept of "pritimives put into a segment".
- A "display list", consisting of representations of segments, and r_functions for displaying graphic primitives.
- Bundles for storing and modifying attributes of output primitives.

All objects built of these data types (plus some static objects like the workstation state list and the workstation description table) reside in one common memory. The underlying hardware is an Intelligent Graphics Terminal[3] - IGT, consisting of three modules which may simultaneously access the memory via a common time-shared bus: (1) A "monitoring module", which realizes the protocol with the host computer and performs storage management and segment handling, (2) an "arithmetics module", which autonomously performs transformations and clipping of output primitives, and (3) a "display module", which contains the drivers for the input devices, and generates output for a vector display operating in combinational (store/refresh, write-thru) mode.

Implementation note: All three modules are based on the I8080 series of microprocessors. Arithmetics and display module are additionally equipped with bit-slice processors, where the latter has direct access to the common bus. For a detailed description of all modules, see[3].

The capabilities of the IGT do not include various linewidths, text in CHAR quality may assume four directions only. Fill areas and cell arrays are simulated, neither colours nor shielding are supplied.

2. PRELIMINARIES

For the description of our data types we use the notation of EL1[6,7]. From EL1 we borrow the built-in data types (modes): INT, REAL, BOOL, CHAR, and PTR - a pointer to a (or one of) specified type(s), STRUCT - like PL/I structures or Pascal records, VECTOR - an array with a fixed number of components, SEQ - an array with an arbitrary number of components, and ONEOF which selects a type from several alternatives. MODE - the type "data type", and EXPR - which constructs a function that may return a type, serve for defining new data types:

```
PAIR ← EXPR (M: MODE; MODE)
 (VECTOR (2, M));
```

PAIR is a function which takes a MODE as argument and returns a VECTOR of two components of that MODE.

```
PTR_PAIR ← EXPR (M: MODE; MODE)
 (VECTOR (2, PTR (M)));
```

PTR_PAIR is a function which takes a MODE as argument and returns a VECTOR of two pointers to objects of that MODE. Note that in EL1 each pointer is initialized with "NIL", a pointer to nothing. The mode definition operator is "::".

```
NDC_POINT :: PAIR (REAL);
DC_POINT :: PAIR (INT);
NDC_RECTANGLE :: PAIR (NDC_POINT);
DC_RECTANGLE :: PAIR (DC_POINT);
```

Implementation note: Operations on data of type INT and PTR by one process have been made continuous (i.e. indivisible) relatively to operations by another process, by changing the hardware mechanism for accessing the common bus. This has become necessary as the (microprogrammed) display process synchronizes with the monitoring process only from one refresh cycle to the next, while other processes may intermediately add and delete elements on the list the display process is working on.

3. OBJECTS

An object is a structure

```
OBJECT :: STRUCT
 (HEAD : OBJ_HEAD, TAIL : ONEOF (SEGMENT, INSTANCE,
  S_FUNCTION, REPRESENTATION, R_FUNCTION, BUNDLE));
```


where

```
OBJ_HEAD :: STRUCT
 (TYPE : INT, ALLOC : OBJ_ALLOC);
```

TYPE is the type of the object. OBJ_ALLOC contains information for the storage allocator, and is inaccessible to routines which implement GKS functions.

Implementation note: Storage management is based on a real-time two phase (marking/reclaiming) garbage collection algorithm. Deallocation has to be required explicitly for segments and representations. Instances, s_functions, and r_functions are collected by the storage manager as soon as the associated segment or representation has been deleted. Inserted segments which have been explicitly deleted are collected, when all segments they had been inserted to, have been deleted too. A detailed description of the storage manager appears in [4].

3.1. Segment

```
SEGMENT :: STRUCT
 (ID : INT, SEG : PTR_PAIR (SEGMENT),
  EXPORT : PTR_PAIR (INSTANCE),
  IMPORT : PTR_PAIR (INSTANCE),
  S_FUN : PTR_PAIR (S_FUNCTION),
  REP : PTR (REPRESENTATION),
  ATT : SEGMENT_ATTRIBUTES);
```

ID is the identifier (name) of the segment. In INGRID names (segment names and pick identifiers) are integers > 0. ID = 0 is reserved for echoing purposes. SEG references the previous and next segment - segments are ordered by their ID's (SEGMENT. SEG [1].ID < SEGMENT.ID < SEGMENT.SEG [2].ID is valid for all segments). EXPORT references the first and last instance which has been created, when the segment has been inserted into another segment. IMPORT references the first and last instance which has been created when another segment has been inserted into the considered segment. S_FUN points to the first and last s_function associated with the segment. Note that the first s_functions belonging to a segment have to retain the setting of primitives' attributes at the time of creation of the segment. REP points to the segment's representation.

```
SEGMENT_ATTRIBUTES :: STRUCT
 (IRG_FLAG : BOOL, TRANSFORMATION : VECTOR (6, REAL),
  VISIBILITY : BOOL, HIGHLIGHTING : BOOL, PRIORITY : REAL,
  DETECTABILITY : BOOL, REFRESH_MODE : BOOL);
```

IRG_FLAG indicates whether changes of the segment´s attributes may be performed immediately or have to be deferred until a regeneration of the display image occurs. This prevents a segment from appearing twice on the screen, e.g. when executing the following sequence of functions:

```
...
SET DEFERRAL STATE (ASAP, SUPPRESSED);
SET REFRESH_MODE (SEG_ID, FALSE);
SET VISIBILITY (SEG_ID, FALSE);
SET SEGMENT TRANSFORMATION (SEG_ID, TRANSFORMATION);
SET VISIBILITY (SEG_ID, TRUE);
```

"SET REFRESH_MODE" is a - actually our only - function not contained in ("escaping") the standard. It takes as arguments the name of a segment to be displayed according to a mode. By properly setting REFRESH_MODE and VISIBILITY, the following levels of optimization are feasible within our implementation (assume IRG_FLAG FALSE in all cases):

- REFRESH_MODE FALSE and VISIBILITY FALSE: The segment is not displayed, no representation has been created for it. Nevertheless the segment exists on the Workstation Dependent Segment Storage and may be used by the INSERT function. Under these circumstances the Workstation Dependent Segment Storage behaves like an extension of a Workstation Independent Segment Storage.

- REFRESH_MODE FALSE and VISIBILITY TRUE: The segment is displayed in store mode, thus economizing refresh time and memory space, as the segment´s representation may be deleted immediately after the display processor has proceeded it.

- REFRESH_MODE TRUE and VISIBILITY FALSE: The segment is not displayed, but may be redisplayed at once, i.e. without newly evaluating it. Refresh time is kept low, however the operator has to interact with the application program for selecting the segments he wants to see.

- REFRESH_MODE TRUE and VISIBILITY TRUE: The segment is permanently refreshed, this results in high refresh time and memory requirements.

Primitives out of segment are displayed in store mode.

3.2. Instance

An instance is created when a segment is inserted into the open segment. Instances are not explicitly accessible by GKS functions.

```
INSTANCE :: STRUCT
 (SEG : PTR_PAIR (SEGMENT),
  SOURCE_INS : PTR_PAIR (INSTANCE),
  TARGET_INS : PTR_PAIR (INSTANCE),
  ATT : INSTANCE_ATTRIBUTES);
```

SEG points to the target segment, i.e. the segment open at the time of invocation of the INSERT function, and to the source segment, i.e. the segment specified in the parameter list of the INSERT function. SOURCE_INS references the previous and next instance of the source segment, TARGET_INS the previous and next instance created when a segment is inserted into the target segment.

```
INSTANCE_ATTRIBUTES :: STRUCT
 (TRANSFORMATION : VECTOR (6, REAL),
  CLIPPING_RECTANGLE : NDC_RECTANGLE);
```

TRANSFORMATION contains the transformation, used as parameter for the INSERT function, multiplied with the transformation of the source segment valid at the time of insertion. CLIPPING_RECTANGLE is the clipping rectangle valid at the time of insertion and replaces all clipping rectangles in the source segment (and all segments inserted into the source segment) every time the target segment is evaluated.

3.3. S_Function

S_functions are data structures for "primitives put into a segment".

```
S_FUNCTION :: STRUCT
 (S_FUN : PTR (S_FUNCTION), CODE : ONEOF
  (S_CLIPPING_RECTANGLE, S_POLYLINE, S_POLYMARKER, S_TEXT,
  S_FILL_AREA, S_CELL_ARRAY, S_GDP, S_POLYLINE_INDEX,
  S_LINETYPE, S_POLYLINE_COLOUR_INDEX, S_POLYMARKER_INDEX,
  S_MARKER_TYPE, S_MARKER_SIZE_SCALE_FACTOR,
  S_POLYMARKER_COLOUR_INDEX, S_TEXT_INDEX,
  S_TEXT_FONT_AND_PRECISION, S_CHAR_EXPANSION_FACTOR,
```

S_CHAR_SPACING, S_TEXT_COLOUR_INDEX, S_CHAR_VECTORS,
S_TEXT_PATH, S_TEXT_ALIGNMENT, S_FILL_AREA_INDEX,
S_FILL_AREA_COLOUR_INDEX, S_ASPECT_SOURCE_FLAGS,
S_PICK_IDENTIFIER));

S_FUN points to the next s_function associated with the same segment.

S_CLIPPING_RECTANGLE :: NDC_RECTANGLE;

S_POLYLINE :: STRUCT
(NR_OF_POINTS : INT, COORDS_OF_POINTS : SEQ (NDC_POINT));

S_POLYMARKER :: STRUCT
(NR_OF_POINTS : INT, COORDS_OF_POINTS : SEQ (NDC_POINT));

S_TEXT :: STRUCT
(STARTING_POINT: NDC_POINT, LENGTH_OF_STRING : INT,
CHAR_STRING : SEQ (CHAR));

S_FILL_AREA :: STRUCT
(NR_OF_POINTS : INT, COORDS_OF_POINTS : SEQ (NDC_POINT));

S_CELL_ARRAY :: VECTOR (3, NDC_POINT);

S_GDP :: STRUCT
(GDP_IDENTIFIER : INT, NR_OF_POINTS : INT,
COORDS_OF_POINTS : SEQ (NDC_POINT));

S_POLYLINE_INDEX : INT;

S_LINETYPE :: INT;

S_POLYLINE_COLOUR_INDEX :: INT;

S_POLYMARKER_INDEX: INT;

S_MARKER_TYPE :: INT;

S_MARKER_SIZE_SCALE_FACTOR :: REAL;

S_POLYMARKER_COLOUR_INDEX :: INT;

S_TEXT_INDEX :: INT;

S_TEXT_FONT_AND_PRECISION :: STRUCT
(FONT : INT, PRECISION : VECTOR (2, BOOL));

S_CHAR_EXPANSION_FACTOR :: REAL;

S_CHAR_SPACING :: REAL;

S_TEXT_COLOUR_INDEX :: INT;

S_CHAR_VECTORS :: PAIR (NDC_POINT);

The character height and width vectors as described in the GKS metafile section (see also 4.5.5 in /1/).

S_TEXT_PATH :: VECTOR (2, BOOL);

S_TEXT_ALIGNMENT :: STRUCT
(HORIZONTAL_ALIGNMENT : VECTOR (2, BOOL),
VERTICAL_ALIGNMENT : VECTOR (3, BOOL));

S_FILL_AREA_INDEX :: INT;

S_FILL_AREA_COLOUR_INDEX :: INT;

S_ASPECT_SOURCE_FLAGS :: VECTOR (10, BOOL);

S_PICK_IDENTIFIER :: INT;

3.4. Representation

Segments are displayed with the help of representations.

REPRESENTATION :: STRUCT
(SEG: PTR (SEGMENT), REP : PTR_PAIR (REPRESENTATION),
R_FUN : PTR_PAIR (R_FUNCTION));

SEG points to the represented segment, REP points to a previous and next representation. Representations are ordered according to the descending priority of the associated segment (for all representations holds:

REPRESENTATION.REP [1].SEG.ATT.PRIORITY >=
REPRESENTATION.SEG.ATT.PRIORITY >=
REPRESENTATION.REP [2].SEG.ATT.PRIORITY). This has the nice advantage, that for pick input the priority of the associated segment need not be investigated, as any picked primitive always belongs to the segment with the relative highest priority the display process has encountered so far. R_FUN points to the first and last r_function belonging to this representation.

3.5. R_function

R_functions are "commands" for the display process.

```
R_FUNCTION :: STRUCT
 (R_FUN : PTR (R_FUNCTION, REPRESENTATION), CODE : ONEOF
  (R_POLYLINE, R_POLYMARKER, R_TEXT, R_STROKE_TEXT,
  R_FILL_AREA, R_CELL_ARRAY, R_GDP, R_POLYLINE_BUN,
  R_LINETYPE, R_POLYLINE_COLOUR_BUN, R_POLYMARKER_BUN,
  R_MARKER_TYPE, R_MARKER_SIZE, R_POLYMARKER_COLOUR_BUN,
  R_TEXT_BUN, R_TEXT_FONT, R_TEXT_COLOUR_BUN,
  R_CHAR_SIZE, R_CHAR_DISPLACEMENT, R_TEXT_DIRECTION,
  R_FILL_AREA·BUN, R_FILL_AREA_COLOUR_BUN,
  R_ASPECT_SOURCE_FLAGS, R_PICK_IDENTIFIER));
```

R_FUN points to the next r_function associated with the same representation. The last r_function associated with a representation contains a back pointer to the representation itself.

```
R_POLYLINE :: SEQ (DC_POINT);

R_POLYMARKER :: SEQ (DC_POINT);

R_TEXT :: STRUCT
 (STARTING_POINT : DC_POINT, CHAR_STRING : SEQ (CHAR));

R_STROKE_TEXT :: SEQ (DC_POINT);

R_FILL_AREA :: SEQ (DC_POINT);

R_CELL_ARRAY :: SEQ (DC_POINT);

R_GDP :: STRUCT
 (GDP_IDENTIFIER : INT, NR_OF_POINTS : INT,
  COORDS_OF_POINTS : SEQ (DC_POINT));

R_POLYLINE_BUN :: PTR (POLYLINE_BUNDLE);

R_LINETYPE :: INT;

R_POLYLINE_COLOUR_BUN :: PTR (COLOUR_BUNDLE);

R_POLYMARKER_BUN :: PTR (POLYMARKER_BUNDLE);

R_MARKER_TYPE :: INT;

R_MARKER_SIZE :: INT;
```



```
R_POLYMARKER_COLOUR_BUN :: PTR (COLOUR_BUNDLE);

R_TEXT_BUN :: PTR (TEXT_BUNDLE);

R_TEXT_FONT :: INT;

R_TEXT_COLOUR_BUN :: PTR (COLOUR_BUNDLE);

R_CHAR_SIZE :: INT;

R_CHAR_DISPLACEMENT :: DC_POINT;
```

The position of a "next" character relative to the position of a "previous" character.

```
R_TEXT_DIRECTION :: VECTOR (2, BOOL);

R_FILL_AREA_BUN :: PTR (FILL_AREA_BUNDLE);

R_FILL_AREA_COLOUR_BUN :: PTR (COLOUR_BUNDLE);

R_ASPECT_SOURCE_FLAGS :: VECTOR (8, BOOL);

R_PICK_IDENTIFIER :: INT;
```

Implementation note: One DC_POINT is encoded in four bytes (32 bits). As the maximum size of our display screen is 4096x4096 discrete addressable points - resulting in 24 bits actually addressable space for one DC_POINT - encoding a DC_POINT leaves eight bits free. Two of them are presently used:

- One bit is employed in R_POLYLINE, R_STROKE_TEXT, R_FILL_AREA and R_CELL_ARRAY to indicate whether the line leading to a point has to be displayed with the required intensity or with intensity zero, i.e. invisible.Invisible lines may appear as a result of the clipping process.The same bit is used in R_POLYMARKER for the indication whether displaying a marker at this point has to be suppressed for MARKER_TYPE not equal one (i.e. not the smallest displayable dot). This provides against the unpleasant effect of "partially visible" markers appearing wrapped around, after dynamically modifying the MARKER_TYPE entry of a polymarker bundle.

- One bit is used to indicate the last DC_POINT of each R_POLYLINE, R_POLYMARKER, R_STROKE_TEXT, R_FILL_AREA, and R_CELL_ARRAY, and the last CHAR in R_TEXT.CHAR_STRING.

The pointers in R_POLYLINE_BUN, R_POLYMARKER_BUN, ... are interpreted by the display process as calls of the correspondent bundle. After copying the values of "realized" components into its internal registers, the display process continues with R_FUNCTION.R_FUN.

3.6. Bundle

Bundles are specific GKS entities. In INGRID settable bundles are created dynamically: Whenever a SET .. INDEX or a SET .. REPRESENTATION function is issued,referencing a bundle not existing so far,a new bundle with the appropriate index is constructed. Once created, bundles may not be deleted until the workstation is closed.

```
BUNDLE :: STRUCT
 (HEAD : BUNDLE_HEAD, TAIL : ONEOF (POLYLINE_BUNDLE,
  POLYMARKER_BUNDLE, TEXT_BUNDLE, FILL_AREA_BUNDLE,
  COLOUR_BUNDLE));
```

All bundles have a header,

```
BUNDLE_HEAD : STRUCT
 (INDEX : INT, STORE_FLAG : BOOL);
```

which contains an INDEX - bundles of the same type are ordered according to this INDEX - and a STORE_FLAG which indicates whether the associated bundle has been actually (i.e. not all output primitives using this bundle have been clipped to "non existence") used for store mode output since the last update. When a SET .. REPRESENTATION function modifies at least one of the bundle´s components and the associated STORE_FLAG is TRUE, a new frame action becomes necessary. With every clearing of the display surface the STORE_FLAGs in all bundles are reset to FALSE. Bundles may be of one of the following types:

```
POLYLINE_BUNDLE :: STRUCT
 (BUN : PTR (POLYLINE_BUNDLE), LINETYPE : INT,
  INTENSITY : PTR (COLOUR_BUNDLE));
```

BUN is a pointer to the next polyline bundle, LINETYPE is the realized linetype, INTENSITY is a pointer to the colour bundle containing the realized polyline intensity.

```
POLYMARKER_BUNDLE :: STRUCT
 (BUN : PTR (POLYMARKER_BUNDLE), REFRESH_FLAG : BOOL,
  MARKER_TYPE : INT, MARKER_SIZE : INT,
  INTENSITY : PTR (COLOUR_BUNDLE));
```


BUN points to the next polymarker bundle, REFRESH_FLAG indicates whether this bundle has been actually used for refresh mode output.When this flag is TRUE and a SET POLYMARKER REPRESENTATION function modifies the MARKER_SIZE component of the bundle, REFRESH_FLAG is reset to FALSE and a reevaluation (no implicit regeneration) of all refreshed segments is performed. MARKER_TYPE is the realized marker type, MARKER_SIZE is the realized marker size, INTENSITY is a pointer to the colour bundle containing the realized polymarker intensity.

```
TEXT_BUNDLE :: STRUCT
 (BUN : PTR (TEXT_BUNDLE), REFRESH_FLAG : BOOL,
  FONT : INT, PRECISION : VECTOR (2, BOOL),
  CHAR_SPACING : REAL, CHAR_EXPANSION_FACTOR : REAL,
  INTENSITY : PTR (COLOUR_BUNDLE));
```

BUN is a pointer to the next text bundle, REFRESH_FLAG indicates whether this bundle has been actually used for refresh mode output.When this flag is true and a SET TEXT REPRESENTATION function modifies at least one of the bundle´s components PRECISION, CHAR_SPACING or CHAR_EXPANSION_FACTOR, REFRESH_FLAG is reset to FALSE and a reevaluation of all refreshed segments is performed. As neither character spacing nor character expansion factor may be evaluated exactly in all cases, this reevaluation will not always cause a visible effect on the display image. FONT is the realized text font of this bundle, PRECISION is the text precision required for this bundle. CHAR_SPACING is the character spacing required for this bundle and CHAR_EXPANSION_FACTOR is the required character expansion factor for this bundle. INTENSITY is a pointer to the colour bundle containing the realized text intensity.

```
FILL_AREA_BUNDLE :: STRUCT
 (BUN : PTR (FILL_AREA_BUNDLE),
  INTENSITY : PTR (COLOUR_BUNDLE));
```

BUN is a pointer to the next fill area bundle, INTENSITY is a pointer to the colour bundle containing the realized fill area intensity.

```
COLOUR_BUNDLE :: STRUCT
 (BUN : PTR (COLOUR_BUNDLE), INTENSITY : INT);
```

BUN is a pointer to the next colour bundle, INTENSITY is the calculated (realized) intensity of this bundle.

4. CONCLUSIONS AND FURTHER EXTENSIONS

An intermediate database of a graphics terminal with sufficient capabilities for transforming and clipping output primitives and dynamically modifying their attributes may consist of a few building blocks. A straightforward implementation of INGRID, comprising memory management and object handling (without transformation and clipping), written in PL/M-80 on an Intellec (MDS) Development System, required about 8K byte of 8080 object code.

Without great efforts a full implementation of the Graphical Kernel System may be (at least conceptually) accomplished: Apart from the realization of a metafile and a Workstation Independent Segment Storage (both may be simulated with existing data types by adding some flags to the definitions of segments) and a modification of bundles - presently bundles do not contain "set" components which are needed by GKS' INQUIRE functions - we need normalization transformations and an event queue: Normalizations should be created when they are referenced for the first time, a method we have already used for bundles.

```
NORMALIZATION :: STRUCT
 (NUMBER : INT, NORM : PTR_PAIR (NORMALIZATION),
  WINDOW : WC_RECTANGLE, VIEWPORT : NDC_RECTANGLE);
```

NUMBER contains the transformation number of the normalization transformation, NORM is a pair of pointers to other normalizations (normalizations will be ordered by descending priority), WINDOW and VIEWPORT are window and viewport of the normalization transformation, where:

```
WC_RECTANGLE :: PAIR (WC_POINT);
WC_POINT :: PAIR (REAL);
```

An event queue is a linked list of events:

```
EVENT :: STRUCT
 (EV : PTR (EVENT), SIM_FLAG : BOOL, REPORT : ONEOF
  (LOCATOR_REPORT, STROKE_REPORT, VALUATOR_REPORT,
  CHOICE_REPORT, PICK_REPORT, STRING_REPORT));
```

EV is a pointer to the next event - events are ordered by increasing event times, SIM_FLAG a flag indicating the occurrence of a simultaneous event: When SIM_FLAG is TRUE the next event should be considered as simultaneous to the present event.

Events are not to be confused with prompt and echo types for input devices. Latter are realized with our basic data types. So, a realization of STROKE echo type "3" requires one segment with s_polymarkers (which are transformed at each workstation update) and one representation with r_polymarkers. CHOICE echo type "5" is realized with the help of a segment with absolute highest priority, STRING echo with the help of an r_text, initially filled with spaces.

```
LOCATOR_REPORT :: STRUCT
 (NORMALIZATION_NUMBER : INT, POSITION : WC_POINT);

STROKE_REPORT :: STRUCT
 (NORMALIZATION_NUMBER, NR_OF_POINTS : INT,
  POINTS_IN_STROKE : SEQ (WC_POINT));

VALUATOR_REPORT :: REAL;

CHOICE_REPORT :: INT;

PICK_REPORT :: STRUCT
 (STATUS : BOOL, SEGMENT_NAME : INT,
  PICK_IDENTIFIER : INT);

STRING_REPORT :: STRUCT
 (STRING_LENGTH : INT, STRING : SEQ (CHAR));
```


ACKNOWLEDGEMENTS

The author wishes to thank V.Prikhodko for his constant encouragement, and A. and H.Leich, W.Mikut and B.Naumann for their constructive comments on this project. Material support has also been provided by B.Buchberger and J.Weiss.

REFERENCES

1. Draft International Standard ISO/DIS 7942, Information Processing Graphical Kernel System (GKS), Functional Description, Version 7.2, NI-5.9/1-83. Nov. 1982.
2. FORTRAN Interface of GKS 7.2, NI-5.9/40-82. Erlangen, Oct. 1982.
3. Leich H., Levchanovsky V., Prikhodko V. Multimikroprozessorsystem zur Steuerung eines Intelligenten Graphischen Terminals (IGT). To appear in: Nachrichtentechnik Elektronik. Also: JINR, P10-83-7, Dubna, 1983 (in Russian).
4. Rudalics M. A Workstation Dependent Segment Storage Manager. Submitted to: 5th Conference on Mathematical Modelling, Programming and Mathematical Methods for Solving Physical Problems. Dubna, 1983.

5. Smith J.M.S., Smith D.C.P. Database Abstractions: Aggregation. Comm.ACM 20,6 (June 1977), 405-413.
6. Wegbreit B. The ECL Programming System. Proc. AFIPS 1971 FJCC, vol.39, AFIPS Press, Montvale, N.J., 1971, pp.253-262.
7. Wegbreit B. The Treatment of Data Types in EL1. Comm. ACM 17,5 (May 1974), 251-264.

Received by Publishing Department
on June 10,1983.

SUBJECT CATEGORIES OF THE JINR PUBLICATIONS

Index	Subject
1.	High energy experimental physics
2.	High energy theoretical physics
3.	Low energy experimental physics
4.	Low energy theoretical physics
5.	Mathematics
6.	Nuclear spectroscopy and radiochemistry
7.	Heavy ion physics
8.	Cryogenics
9.	Accelerators
10.	Automatization of data processing
11.	Computing mathematics and technique
12.	Chemistry
13.	Experimental techniques and methods
14.	Solid state physics. Liquids
15.	Experimental physics of nuclear reactions at low energies
16.	Health physics. Shieldings
17.	Theory of condenced matter
18.	Applied researches
19.	Biophysics

Рудалич М. E11-83-393
Промежуточная база данных интеллектуального графического терминала

Промежуточная база данных интеллектуального графического терминала управляет хранением, восстановлением и манипуляцией описаний графических объектов в памяти терминала. Она объединяет в одной общей памяти представления "приборно-зависимая база данных для сегментов", описанная в стандарте ГКС, и "дисплейный список" для терминала, обеспечивающего построение графических примитивов на экране векторного дисплея в запоминающем и регенеративном режимах.

Работа выполнена в Лаборатории вычислительной техники и автоматизации ОИЯИ.

Препринт Объединенного института ядерных исследований. Дубна 1983

Rudalics M. E11-83-393
An Intelligent Graphics Terminal´s Intermediate Database

The Intelligent Graphics Terminal´s Intermediate Database (INGRID) handles storage, retrieval and manipulation of the description of graphic items in the memory of an Intelligent Graphics Terminal. INGRID combines on one storage level the concepts of a Workstation Dependent Segment Storage, as described in the Graphical Kernel System (GKS), and a display list for a workstation providing output of graphic primitives on a vector display in store and refresh mode.

The investigation has been performed at the Laboratory of Computing Techniques and Automation, JINR.

Preprint of the Joint Institute for Nuclear Research. Dubna 1983