ソフトウェア工学 37－3
（１９８４．７．２５）

仮想デバイス・インターフェイス（ＶＤＩ）の実現

近藤明男，今宮淳美
（山梨大学　工学部　計算機科学科）

１．はじめに

システムの開発や保守に要するコストを抑えるために，ソフトウエアに可搬性を持たせることが重要である．この観点からグラフィックス・ソフトウエア・パッケージの標準化案が種々具体化し，応用プログラムに可搬性を与えることに成功している［服部８３］．しかし異なるグラフィックス装置を使用する場合，グラフィックス装置毎のグラフィックス・パッケージが必要となる．このグラフィックス・パッケージをグラフィックス装置に独立した部分とグラフィックス装置に依存した部分に分離し，その間の標準インターフェイスを定義することで，グラフィックス・パッケージにも可搬性を持たせることができる．本稿では，異なる複数のグラフィックス装置を持つ環境におけるこのインターフェイスの実現と，実現するにあたっての問題点，解決方法を述べる．

２．ＶＤＩ概要

２．１　ＶＤＩの必要性

従来グラフィックス装置を駆動するのに，グラフィックス・パッケージが使用されている．このグラフィックス・パッケージを標準化することで，応用プログラムに可搬性を与えることに成功している．よく知られた汎用グラフィックス・パッケージにはＣｏｒｅやＧＫＳがある［南出８３］［図形８３］．しかしこれらのグラフィックス・パッケージを異なるグラフィックス装置毎にインプリメントする必要がある．一方，現在稼働しているシステムに新しいグラフィックス装置を導入するときも，その装置のためにグラフィックス・パッケージをインプリメントしなければならない．

そこで，グラフィックス・パッケージの各グラフィックス装置に独立する部分と装置に依存する部分を分割し，その間の標準インターフェイスを定義することで，汎用グラフィックス・パッケージに可搬性を与えることができる．このインターフェイスをＤＩ／ＤＤインターフェイスまたは仮想デバイス・インターフェイスと呼んでいる［今宮８３］．また，新しいグラフィックス装置を導入するときも，このインターフェイスに合せたデバイス・ドライバを作成することで，容易に既存のシステムに導入できる．この様子を図１に示す．

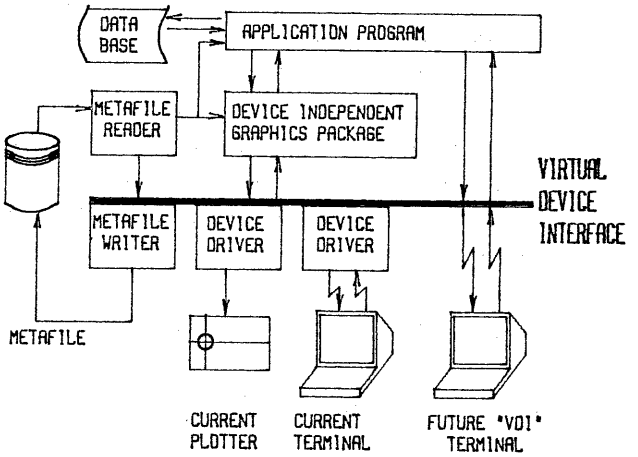

図１．汎用グラフィックス・システム

２．２　ＶＤＩの利点

このような標準インターフェイスを定義することで，次のような利点が期待できる．

（１）　グラフィックス・パッケージの作成者は，個々のグラフィックス装置を意識せずにグラフィックス・パッケージの装置独立部分の作成に専念出来る．

（２）　デバイス・ドライバの作成者は，応用プログラムやグラフィックス・パッケージの機能にとらわれずデバイス・ドライバを作成できる．

（３）　ひとつのグラフィックス・パッケージで様々なグラフィックス装置を駆動できるので，応用プログラマは様々なグラフィックス・パッケージを習得する必要がない．

2．3　ANSI−VDI概要

ANSI(American National Standard Institute)X3H33タスク・グループはグラフィックス・ソフトウエア・パッケージの装置独立部分と装置依存部分との間のインターフェイスVDIの標準化案を提案している［ANSI81］［ANSI82a］［ANSI82b］．以下に，本インプリメンテーションで採用したANSI−VDI案の概要を述べる．

2．3．1　ANSI−VDI目標

ANSIではVDIの設計目標として，次の8項目を掲げている：

（1）　装置独立のグラフィックス・パッケージの作成者に標準インターフェイスを与える．

（2）　装置依存のグラフィックス・デバイス・ドライバ作成者に標準インターフェイスを与える．

（3）　グラフィックス装置の機能，状態についての問い合せと応答の機構を持つ．

（4）　グラフィックス装置のもつ非標準機能をアクセスするためのエスケープ機構を持つ．

（5）　複数のデバイス・ドライバを並行してサポートできる．

（6）　VDIの機能の拡張を許す．

（7）　VDIの機能の分散（グラフィックス装置自体，デバイス・ドライバ，VDIより上位のいずれか）を許す．

（8）　オーバヘッドを小さくするために，ユーザが直接VDIを使用できる．

2．3．2　VDI機能分類

VDIで定義する機能は，要求機能と非要求機能に大別できる．

（1）　要求機能（Required Functions）

VDIより下位で必ず用意する機能の集合．グラフィックス装置自体がサポートしていない場合は，デバイス・ドライバでその機能をエミュレートしなければならない．

（2）　非要求機能（Nonrequired Functions）

VDI機能として定義しているが，必ずしもサポートすることを強制しない機能の集合．非要求機能の使用に際しては，デバイス・ドライバに対して事前にサポートしているか否か問い合せる必要がある．デバイスかデバイス・ドライバでサポートしていれば利用できるが，サポートしていない場合はVDIより上位でエミュレートしなければならない．

これらの機能の分類は流動的なもので，多くのデバイスで使用できるようになった非要求機能は，要求機能の集合へ移行することができる．

2．3．3　制御機能

VDIでは，LDC（Logical Device Coordinate)と呼ばれるユーザ定義の2次元正規化座標系を用いる．VDIの入出力はすべてこの平面内で行われる．LDC内の範囲を指定して，それをディスプレイ面に指定したビューポートに写像することができる．この写像にはLDCでの縦横比を保持するモードと縦横の縮尺を任意に設定できるモードがある（図2）．これら写像の範囲指定やモードの

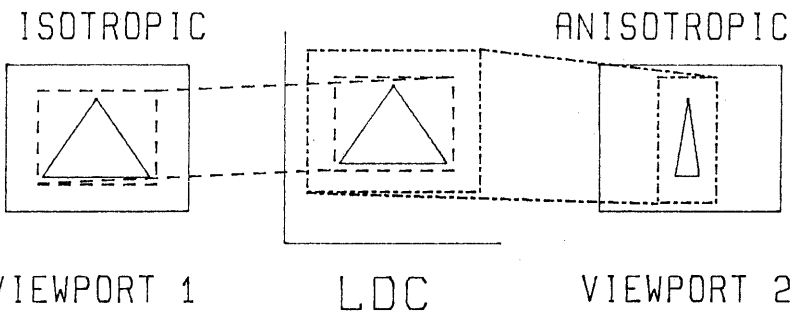

図2．LDC−ビューポート写像

指定は制御機能に含まれる．制御機能には他に，ＶＤＩの初期化やＶＤＩの使用終了，非要求機能のサポートの問い合せ，エスケープ機能のアクセスがある．

２．３．４　出力機能

ＶＤＩの出力要素には，現在点の移動，折れ線，マーカ記号，塗りつぶし多角形や円，円弧の表示，ピクセル列，テキスト列の表示等がある．これらの出力要素にはそれぞれカラー，線幅，線種，マーカ種，塗りつぶしの種類等の属性を指定することができる．

２．３．５　セグメント機能［ＡＮＳＩ８２ａ］

セグメントとは，論理的に扱える図形の最小要素であり，イメージの動的変更に有効である．セグメント内容は，セグメント本体とセグメント属性でできており，セグメント識別子で識別する．セグメント本体はＶＤＩ出力要素と出力（静的）属性でできており，セグメント生成中に定義される出力要素はそのセグメントの一部となる．セグメント属性はすべて動的なものであり，プログラムまたはインタラクションにより可変である．それらには可視属性，検出属性，強調，イメージ変換，表示優先度，検出優先度がある．これらのセグメント機能は，ＶＤＩでは非要求機能に分類される．

２．３．６　入力機能［ＡＮＳＩ８２ｂ］

ＶＤＩの論理入力装置は，６種の論理入力クラスでできている．論理入力装置は論理入力クラスとクラス内の装置番号で指定する．論理入力クラスは，ユーザ・プログラムに返すデータ型で分類する．

（１）　ロケータ（Ｌｏｃａｔｏｒ）：　ＬＤＣ空間での単一の座標値を返す．
（２）　ストローク（Ｓｔｒｏｋｅ）：　ＬＤＣ空間内の座標列を返す．
（３）　バリュエータ（Ｖａｌｕａｔｏｒ）：　実数値を返す．
（４）　ピック（Ｐｉｃｋ）：　セグメント識別子と出力要素識別子を返す．
（５）　チョイス（Ｃｈｏｉｃｅ）：　選択肢の範囲内のひとつを表す整数値を返す．
（６）　ストリング（Ｓｔｒｉｎｇ）：　文字列を返す．

これらの論理入力装置それぞれに，３つの入力モードの何れかを指定することができる．

（１）　リクエスト・モード：　このモードでは，入力はオペレータからの有効な入力値の受けとり，オペレータのブレーク動作，または指定タイムアウトで終了する．同期入力のモードである．
（２）　サンプル・モード：　このモードでは，オペレータの入力動作を待たず，現在値を入力値として返す．
（３）　イベント・モード：　このモードでは，オペレータの発した入力がキューに格納される．ユーザ・プログラムはＡＷＡＩＴ－ＥＶＥＮＴの呼び出しで入力を得ることができる．非同期入力のモードである．

３．インプリメンテーションとその問題点

ＡＮＳＩワーキング・ドキュメント提案のＶＤＩを参考に，山梨大学工学部図形画像システム室でＶＤＩを実現した［近藤８４］．これを以下ではＹＶＤＩと呼ぶことにする．その詳細とＡＮＳＩ提案（１９８１年度版，最終案とは異なる）との違いと問題点を述べる．

３．１　ＹＶＤＩインプリメンテーション

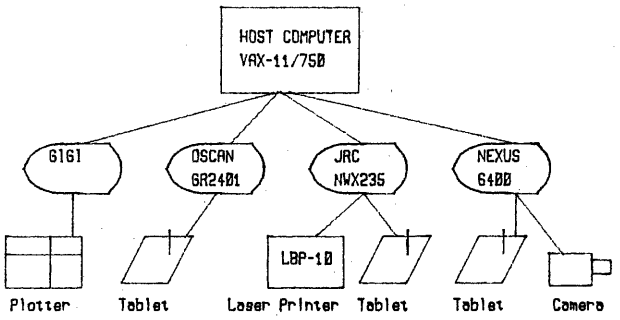

図３．山梨大学工学部図形画像システム室ハードウェア

山梨大学工学部図形画像システム室のハードウェア概略を図３に示す．ＹＶＤＩはホスト・コンピュータＶＡＸ－１１／７５０上にＦｏｒｔｒａｎでインプリメントした．図４にＹＶＤＩソフトウエア・モジュール図を示す．以下に各モジュールの概略を説明する．

（1） ＶＤＩエントリ： ＶＤＩ機能サブルーチンの集合である．ここでエラーのチェック，データ転送領域へのデータの格納，ワークステーション状態の変更等を行い，デバイス・ドライバ・コントローラを呼び出す．

（2） デバイス・ドライバ・コントローラ：ワークステーション状態保持領域の内容にしたがい，それぞれのデバイス・ドライバを呼び出す．

（3） デバイス・ドライバ： データ転送領域の内容にしたがい，ワークステーションを駆動する．入力機能が要求された場合は，入力装置から得た値をデータ転送領域に格納する．

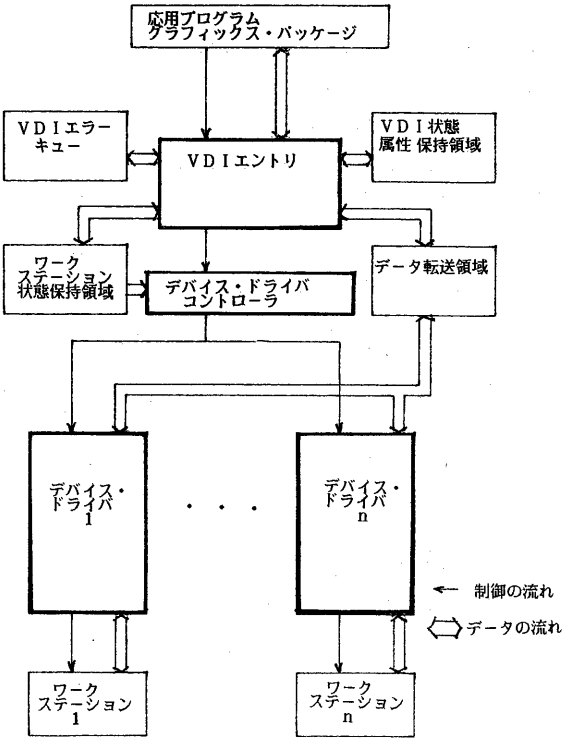

図4．ＹＶＤＩソフトウエア・モジュール図

３．２ 複数並行デバイス・ドライバのサポート

ＡＮＳＩでは，ＶＤＩの目標の一つに複数のデバイス・ドライバを同時サポートできることを掲げている．しかしそのための機能の設定や機構については，問題点として保留されている．ＹＶＤＩではこの機構をＶＤＩに持たせるとともに，機能の設定をおこなう．

複数のデバイス・ドライバを同時サポートする機構として，次の３方法が考えられる．

（1） すべてのデバイス・ドライバを参照し，選択されているデバイス・ドライバだけを呼び出す方法．

（2） デバイス・ドライバの動的リンケージによる方法［ＲＥＥＤ８１］．

（3） デバイス・ドライバをサブプロセスとする方法．

（1）の方法は，すべてのデバイス・ドライバをリンクしておき，応用プログラムに選択されているデバイス・ドライバにのみ制御を渡す．これは通常のサブルーチン呼び出しとして標準Ｆｏｒｔｒａｎで実現できる．しかしすべてのデバイス・ドライバをリンクしておく必要があるので，ただひとつのデバイス・ドライバしか使用しない応用プログラムでもサイズが大きくなってしまうという欠点がある．

（2）の方法は，あるデバイス・ドライバが応用プログラムに選択されると，そのデバイス・ドライバのみリンク・ロードする機構をもつ．これは使用するデバイス・ドライバしか参照しないので，応用プログラムのサイズの増大は抑えることができる．しかし実行時の動的リンキング・ローディングの機構は，ホスト・コンピュータのＯＳのレベルでの処理が必要となる．

（3）の方法は，あるデバイス・ドライバが応用プログラムに選択されると，そのデバイス・ドライバのためのサブプロセスを生成し，グラフィックス装置に依存する部分の処理はサブプロセスで行なうようにする方法である．このとき，データの受渡しはプロセス間通信により行なう．サブプロセスの生成やプロセス間通信機構は，ホスト・コンピュータのシステム・サービスで提供されているのでインプリメンテーションが容易である．また動的にデバイス・ドライバのサブプロセスを生成，削除するので，応用プログラムのサイズが増大することはない．しかしプロセス間通信に要する時間が問題である．

ＹＶＤＩでは，（1）の方法を改良して採用した．応用プログラムがＶＤＩ機能を呼び出すと，ＶＤＩですべ

てのデバイス・ドライバが読むことのできるデータ構造をデータ転送領域に生成する（図５参照）．このデータ構造は，ＶＤＩ機能コードとそれに付帯するパラメタ群でできている．次にデバイス・ドライバ・コントローラを呼ぶ．このサブルーチンは，選択されているデバイス・ドライバだけに制御を渡す．このサブルーチンは，応用プログラムの必要に応じて修正することができる．

これにより使用するデバイス・ドライバだけをリンクすることができるので，応用プログラムのサイズの増大を抑えることができる．

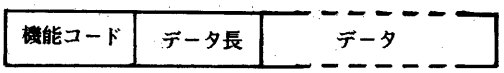

図５．各デバイス・ドライバに共通のデータ構造

３．３　機能分類とエミュレーション

ＡＮＳＩ案では，ＶＤＩ機能が要求機能と非要求機能に分類されることは前に述べた．各デバイス・ドライバがサポートしていない非要求機能は，デバイス・ドライバより上位でエミュレートしなければならない．このため，非要求機能については必ずしも物理装置の特性を考慮しなくてよいとはいえない．この問題については，

（１）　ＶＤＩで装置に独立したエミュレート・ルーチンを用意し，非要求機能のサポート問い合せとエミュレートの実行を自動化する，

（２）　すべてを要求機能として，グラフィックス装置でサポートしていない機能はデバイス・ドライバで必ずエミュレートする．

という解決が考えられる．（１）のような方法が理想的であるが，そのための機構が大きくなり，必ずしもエミュレートを要しない場合の無駄が多い．（２）では，物理装置によってはエミュレートできない機能が存在し，すべてを要求機能とするわけにはいかない．例えばストレージ型ディスプレイでは通常ピクセル列の表示ができない．またフラット・ベッド型のプロッタで図形の塗りつぶしや内部パターン（ハッチング等）をエミュレートすることはできるが，時間がかかりすぎる．

このためＹＶＤＩでは次のような方針を採る．

（１）　要求機能でエミュレートでき実行に時間のかからない非要求機能は，デバイス・ドライバ内でエミュレートを行なう．たとえば塗りつぶさない多角形等はデバイス・ドライバ内でエミュレートできる．

（２）　エミュレートに時間のかかる非要求機能は，ＶＤＩ利用者がエミュレートを行なう．この場合は，描画の精度と実行時間とのトレード・オフとなる．たとえばプロッタでの塗りつぶし等がこれに相当する．

（３）　エミュレートの不可能な非要求機能は，何も行わないかまたは代替の方法でエミュレートする．たとえばプロッタやストレージ型ディスプレイでのピクセル列の表示はハッチングでエミュレートすることができる．

ＶＤＩの上位のグラフィックス・パッケージがＶＤＩの非要求機能を使用するには，図６に示すような手続をとればよい．

```
begin
   inquire（非要求機能名，result）；
   case result of
yes  :          非要求機能の実行
emulated  :     （＊描画精度と時間とのトレード・オフで＊）非要求機能の実行
no  :           （＊描画精度と時間とのトレード・オフで＊）非要求機能のエミュレート
   end
end;
```

図６．非要求機能のアクセス手続

３．４　入力機能のサポート

ＶＤＩで入力機能をサポートするためにＡＮＳＩ−ＶＤＩ案の入力機能を検討し，実現方法を述べる．

３．４．１　入力装置

ＡＮＳＩ案［ＡＮＳＩ８２ｂ］では，入力装置と出力装置を全く分離したものとして扱っている．しかし現在のインタラクティブ・グラフィックス応用に用いられる入力装置は，出力装置と一体化しているワークステーションの一部と考えるのが現実的である．このためＡＮＳＩ案のように入力装置と出力装置を分離して考えると，以下の問題点がある．

（１）　装置の持つインテリジェント機能（例えば入力のエコー）が十分に活かせない．

（２）　ワークステーション型の装置では，入力値を得るためにコマンドを送る必要がある．このためイベント・モードで複数のワークステーションからの入力を得ようとすると，常に各ワークステーションにコマンドを送りつずけるポーリングが必要で，実行速度の低下を招く．

（３）　多くの入力装置をホスト・コンピュータがサポートすると，同一論理クラス内の装置番号が増大してしまう．

上記のような問題のため，ＹＶＤＩでは対象とするグラフィックス装置をワークステーションとし，ＡＮＳＩ案の機能を部分的に変更する．ここでのワークステーションとは，ＧＫＳの入出力カテゴリのワークステーションと類似する［図形８３］．すなわち以下の事項を仮定する：

（１）　一般に１個の出力ビュー面と１個以上の入力装置を持ち，それらは結合している．入力装置または出力装置だけのワークステーションも存在してよいが，たとえば出力装置だけのワークステーションに対する入力の要求は無視する．

（２）　インテリジェント機能として，入力キューやセグメント・バッファなどを持っている．このような機能のない装置では，その機能をデバイス・ドライバかあるいはＶＤＩより上位でエミュレートする．

このように仮定することで，

（１）　入力キューやセグメント・バッファ等の装置の持つ機能を十分活用でき，応用プログラムの負担を軽減できる．

（２）　入力要求は，ワークステーション識別子とクラス，装置番号で指定するので，使用しないワークステーションがあっても入力装置番号のずれや増大がない．

３．４．２　入力モデル

ＡＮＳＩ案［ＡＮＳＩ８２ｂ］では，入力モデルとしてＧＫＳの入力モデルを採用している．これは前述のように，各入力装置に対して入力モードを設定できる．一方現行のワークステーション型装置では，Ｃｏｒｅの入力モデルを採用していて，それをインテリジェント機能として持つ装置が多くある．これは図７に示すように，論理入力クラスにより入力モードが決っており，陽に指定することはできない．したがってＣｏｒｅモデルの入

＊　ロケータ　　−−　サンプル型
＊　バリュエータ　−−　サンプル型
＊　ピック　　−−　イベント型
＊　キーボード　−−　イベント型
＊　ストローク　−−　イベント型
＊　ボタン　　−−　イベント型

図７．Ｃｏｒｅ論理入力装置

Ｃｏｒｅ型分類	ＶＤＩモード	装置状態	入力装置の操作
サンプル型	サンプル・モード	使用可	物理装置からサンプルする．
	リクエスト・モード	使用可	入力要求があるときだけボタンをアソシエイトする．
	イベント・モード	使用可	イベント・モードの間ボタンとアソシエイトしておく．
イベント型	サンプル・モード	使用不可	初期値を返す．
	リクエスト・モード	使用不可	入力要求があるときだけ装置を使用可の状態にする．
	イベント・モード	使用可	イベント・モードの間だけ装置を使用可の状態にする．

図８．Ｃｏｒｅモデル入力装置でのＶＤＩシミュレーション

力装置を持つワークステーションでは，

（1） イベント型入力装置によるサンプル・モードの入力ができない．

（2） イベント・モードにない入力装置からの入力も，入力キューに混入する．

という問題がある．

（1）については，内部にバッファを設けることで解決する．このバッファに初期値を保持しておき，これをサンプル・モードの入力値として用いる．この値は，リクエスト・モードで入力があるたびにその入力値に更新する．一方，2度め以降のサンプル入力の要求に対しては，ｎｕｌｌ値を返す．

（2）については，Ｃｏｒｅ型ワークステーションのイベント型装置を使用するときに，ＶＤＩの入力にしたがい装置の使用可，不可の指定を行なう．すなわちリクエスト・モードでは通常装置は使用不可状態にあるが，ＲＥＱＵＥＳＴ−ＩＮＰＵＴが発されたときだけ装置を使用可状態にする．こうすると，入力キューにリクエスト・モードの装置の入力値が混入することはない．図8にＶＤＩの入力モードをシミュレートするときのＣｏｒｅモデル入力装置の操作を示す．しかしこの方法では，リクエスト・モードでのエコーがでないという欠点もある．

4．おわりに

本稿では，ＶＤＩの実現とその問題点及びその解決案を述べた．実現したＶＤＩは，

（1） 図形データがＶＤＩより上位でビューイング・パイプラインを通ること，

（2） イメージ変換が2次元であること，

を満たせば，多くのグラフィックス応用に十分対応できると考えられる．

今後，3次元対応のインテリジェント機能を持ったグラフィックス装置の普及が考えられるが，ＶＤＩでもこれに対応する必要がある．またファームウエアでＶＤＩ機能を実現する端末，ＶＤＩ端末の開発が必要である．

本研究は山梨大学工学部図形画像システムを用いて行った．システム管理委員会メンバの方々の協力に感謝いたします．

参考文献：

[ＡＮＳＩ８１] X3H33 Task Group POSITION PAPER:VIRTUAL DEVICE INTERFACE and VIRTUAL DEVICE METAFILE,American National Standard Institute (December 1,1981)

[ＡＮＳＩ８２ａ] X3H33 Task Group:Segmentation Functions,(October 2,1982)

[ＡＮＳＩ８２ｂ] X3H33 Task group:Input Functions,(July 7,1982)

[服部８３] 服部幸英：標準化の思想と原理，ＰＩＸＥＬ’８３ ９−１０，(September 1983)

[今宮８３] 今宮淳美：ＡＮＳＩの標準化動向，ＰＩＸＥＬ’８３ ９−１０，(September 1983)

[近藤８４] 近藤明男：拡張性を考慮した仮想デバイス・インターフェイスに関する研究，山梨大学卒業論文，(March 1984)

[南出８３] 南出仁志：使う側の立場から見たＣＯＲＥシステムのまとめ，ＰＩＸＥＬ’８３ ９−１０，(September 1983)

[ＲＥＥＤ８１] T.N.Reed:EXPERIENCE IN THE DESIGN AND SUPPORT OF A GRAPHICS DEVICE DRIVER INTERFACE,EUROGRAPHICS'81,North-Holland Publishing Company, (1981)

[図形８３] 図形処理研究会：ＧＫＳの機能，ＰＩＸＥＬ’８３ ９−１０，(September 1983)

付録）　ＶＤＩ機能リスト

```
Required Outout Functions;
    MOVE                                POLYLINE
    POLYMARKER                          RETURN_CURRENT_POSITION

Nonrequired Output Functions;
    SET_COLOR                           SET_INTENSITY
    SET_LINE_WIDTH                      SET_LINE_STYLE
    SET_MARKER                          SET_INTERIOR_STYLE
    SET_HATCH_STYLE                     SET_PATTERN_STYLE
    SET_BIT_PLANE_ACCESS_MASK           SET_BIT_PLANE_DISPLAY_MASK
    SET_COLOR_TABLE_ENTRIES             PIXELS
    POLYGON                             CIRCLE
    RECTANGLE                           ARC
    ARC_CLOSE

Required Text functions;
    TEXT                                SET_TEXT_HEIGHT
    SET_TEXT_ALIGNMENT

Nonrequired Text Functions;
    DESIGNATE_CHARACTER_SET             INVOKE_CHARACTER_SET
    SET_TEXT_WIDTH                      RETURN_TEXT_EXTENT
    SET_TEXT_PRECISION                  SET_CHARACTER_ORIENTATION
    SET_CHARACTER_PATH                  SET_INTERCHARACTER_SPACE
    SET_TEXT_FONT                       SET_FONT_TABLE_ENTRIES

Required Control Functions;
    INQUIRE                             ESCAPE
    INITIALIZE_VIRTUAL_DEVICE           TERMINATE_VIRTUAL_DEVICE
    RETURN_VIRTUAL_DEVICE_STATUS        CLEAR_VIRTUAL_DEVICE
    SET_MAPPING_MODE                    SET_LDC_RANGE
    SET_MAPPING_REFERENCE_POINTS        OPEN_WORKSTATION
    CLOSE_WORKSTATION                   ACTIVATE_WORKSTATION
    DEACTIVATE_WORKSTATION

Nonrequired Segmentation Functions;
    CREATE_SEGMENT                      CLOSE_SEGMENT
    EXTEND_SEGMENT                      DELETE_SEGMENT
    DELETE ALL SEGMENTS                 RENAME_SEGMENT
    SET_SEGMENT_VISIBILITY              SET_SEGMENT_DETECTABILITY
    SET_SEGMANT_HIGHLIGHTING            SET_SEGMENT_DISPLAY_PRIORITY
    SET_SEGMENT_PICK_PRIORITY           SET_SEGMENT_IMAGE_TRANSFORMATION
    INTERROGATE_OPEN_SEGMENT            INTERROGATE_SEGMENT_NAME
    INTERROGATE_ALL_SEGMENT_NAMES       INTERROGATE_SEGMANT_ATTRIBUTES
    NEGOTIATE_SEGMENTATION_FUNCTIONS    NEGOTIATE_DYNAMIC_ATTRIBUTES

Nonrequired Input Functions;
    INITIALIZE_INPUT_DEVICE             RELEASE_INPUT_DEVICE
    SET_INPUT_DEVICE_MODE               SAMPLE_INPUT
    REQUEST_INPUT                       AWAIT_EVENT
    FLUSH_EVENT_QUEUE                   SET_PROMPT_STATE
    SET_ECHO_STATE                      SET_ACKNOWLEDGEMENT_STATE
    INQUIRE_INPUT_DEVICE_SET            INQUIRE_INPUT_DEVICE_STATE
    INQUIRE_EVENT_QUEUE_STATE
```