グラフィクスとＣＡＤ 17－5
（１９８５・５・３１）

ＣＧＩ入力機能の実現

Implementation of CGI input functions

牧　喜代司，近藤明男，今宮淳美
Kiyoshi Maki, Akio Kondoh, and Atsumi Imamiya
山梨大学　工学部　計算機科学科
Department of Computer Science, Yamanashi University,
4-3-11 Takeda,Kofu,400, Japan

This paper describes the implementaion of ANSI computer graphics interface(CGI) input functions on a micro computer under Unix.
The actor like processes are used to implement CGI input mechanism.
The issue of the implementaion and same alternatives to them are given.

１．はじめに

現在、コンピュータ・グラフィックスではユーザ・インターフェイスを重視する傾向にある。ワークステーションは数多くのグラフィックス・デバイスを包含し、ユーザ・インターフェイスを強化している。しかし、既存のグラフィックス・パッケージでは、グラフィックス・デバイスを追加・削除するたびに、デバイス・ドライバをつくり直さなければならない。このため考えだされたのが、コンピュータ・グラフィックス・インターフェイス（ＣＧＩ）である。以前は、仮想デバイス・インターフェイス（ＶＤＩ）とよばれていた。

ＣＧＩは、グラフィックス・デバイスを統合したワークステーションとグラフィックス・パッケージのインターフェイスのＡＮＳＩ標準化案である。この標準化案によって、グラフィックス・デバイスに依存する部分をグラフィックス・パッケージから分離し、グラフィックス・パッケージに可搬性を持たせることができる。本稿では、小型コンピュータの標準的なオペレーティング・システムであるＵｎｉｘ上に、ＣＧＩの入力機能を主として実現する。実現する際の問題点も明らかにし、その解決案を検討する。

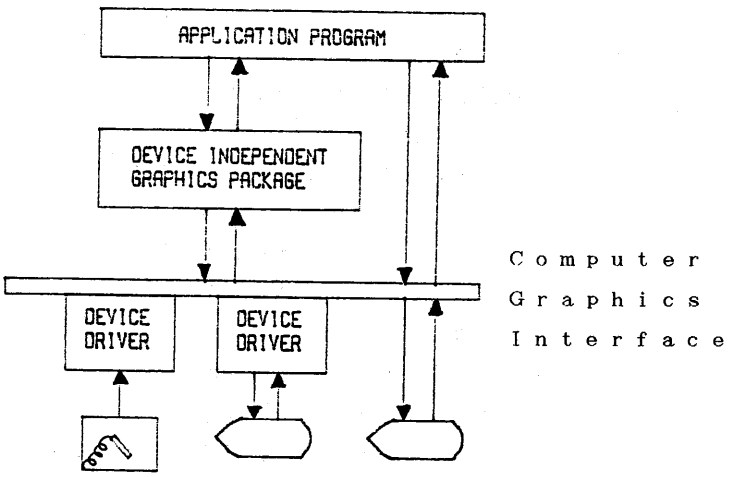

図１．　ＣＧＩ関係モデル

２．ＣＧＩ概要

２．１　ＣＧＩの必要性

現在、グラフィックスにおいて、標準化されたグラ

フィックス・パッケージが広く利用されている。グラフィックス・パッケージにより、応用プログラムに可搬性を持たせている。代表的なグラフィックス・パッケージとしてSIGGRAPHのＣｏｒｅやISO のＧＫＳが挙げられる。しかし、グラフィックス・デバイスをグラフィックス・システムに新しく導入すると、デバイス・ドライバを新しく作成しなければならない。

この問題はグラフィックス・パッケージにＣＧＩを導入することで解決できる。グラフィックス・パッケージを、グラフィックス・デバイスに依存する部分と依存しない部分に分け、２つの部分間のインターフェイス（Computer Graphics Interface ）を標準化することで、グラフィックス・パッケージに、より有効な可搬性を持たせることが可能である。グラフィックス・デバイスを新しく導入する場合にはＣＧＩに基づくデバイス・ドライバを用意することで、グラフィックス・システムに導入できる。グラフィックス・パッケージを新しく導入する場合には何の変更もなくグラフィックス・システムに導入できる。この様子を図１に示す。

２．２　ＡＮＳＩ　ＣＧＩ

現在ＣＧＩについてはＡＮＳＩ（American National Standards Institute ）Ｘ３Ｈ３において標準化作業が行なわれている。本稿では、１９８４年ＡＮＳＩ／Ｘ３Ｈ３ワーキング・ドキュメント［ＡＮＳＩ８４］のＣＧＩ実現についてである。

２．３　ＣＧＩの特徴

ＣＧＩ標準案の特徴は次の３つの項目である。

（１）デバイス独立なグラフィックス・システムを、一貫した方法で幅広いグラフィックス・デバイスとインターフェイスが可能である。

（２）グラフィックス・デバイスのデバイス依存部分を分離することによって、システム間のソフトウェア交換を助ける。

（３）デバイスの製作を使いやすいグラフィックス能力という点から説明する。

２．４　ＣＧＩの入力機能

２．４．１　ＣＧＩの入力基本モデル

入力、出力、入出力の３種類のＣＧＩがある。入力ＣＧＩは、１個以上の論理入力デバイスと、１個以上のトリガを持つ。入出力ＣＧＩは、入力ＣＧＩと出力ＣＧＩ両方の機能および入出力デバイスの追跡機能（トラッキング）を持つ。図２参照。

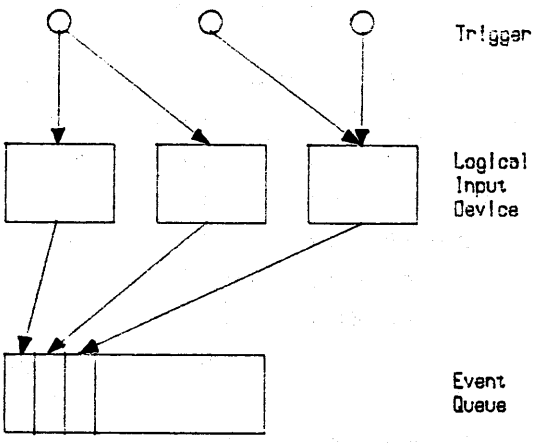

図２．　ＣＧＩの入力基本モデル

２．４．２　論理入力デバイスとトリガ

論理入力デバイス（ＬＩＤ）は、物理デバイスとは異なる論理上のデバイスである。各ＬＩＤはユーザに返すデータ・タイプに基くクラスに分類される。

○ロケータ …… 単一の座標値を返す。
○ストローク …… 座標値列を返す。
○バリュエータ …… 実数値を返す。
○チョイス …… 選択肢の範囲内の１つを表わす整数値を返す。
○ストリング …… 文字列を返す。
○ピック …… セグメント識別子を返す。

物理デバイスがＬＩＤのどのクラスに属するかはインプリメンテーションに依存する。

ＬＩＤは１つ以上のトリガと結合するメジャでできている。

トリガは、ＣＧＩ上の応用ソフトウェアに（図形）入力を同期させる手段である。トリガがメジャに入力の有効を知らせることを、トリガの発火と呼ぶ。トリガの発火は、物理的動作で実現される。

メジャとは、トリガの発火を受けデバイスから入力値をつくりだすプロセスである。各トリガは１つ以上のメジャ（ＬＩＤ）と結合してよい。ＣＧＩでは、トリガとメジャの結合および解除機能要素がある。

２．４．３　ＬＩＤの使用

ＬＩＤから入力を得るには、ＬＩＤを初期化する必要がある。そして、ユーザまたは応用プログラムからのコマンド、すなわち発せられた入力要素がインタラクションの方法を決定する。その方法には、次の３つの項目がある。

（１）サンプル操作

特別なオペレータ動作なしに、入力の現在値を得る。ＣＧＩを通して指定デバイスにSAMPLE_INPUT要素が発せられると、ＬＩＤはいつもメジャ値を返す。

（２）リクエスト操作

ユーザ・ソフトウェアとオペレータ間にインタラクションを与える。ユーザ・ソフトウェアがREQUEST_INPUT 要素を発すると、（メジャ値を返すために）ＬＩＤはトリガの発火を待つ。

（３）イベント・モード

入力トリガが発火されると、ＬＩＤはメッセージをイベント・キューに格納する。各メッセージはトリガの発火を表わし、そのトリガに結合するＬＩＤのうちのひとつのメジャ値を持っている。ユーザはAWAIT_EVENT 要素でイベント・キューからイベント・メッセージを取り出すことができる。

２．４．４　イベント・キュー

入力、入出力ＣＧＩは、イベント・モードである全てのＬＩＤからのイベント・メッセージを格納する単一イベント・キューを持つ。これは、インプリメンテーション依存容量のＦＩＦＯキューである。

２つ以上のメッセージが同一のトリガで生成される場合、このメッセージを同時メッセージという。同時メッセージは、メッセージ・リンク・フィールドを真にすることで、単独のメッセージから識別される。同時メッセージは、キュー内で連続して保持されねばならない。また同時メッセージでのグループ最後のメッセージのメッセージ・リンク・フィールドは偽である。そして、同一グループ内の同時メッセージの順序は定義されていない。

２．４．５　エコーとトラッキング

ＬＩＤのメジャ値についてのフィードバックを反響化（echoing ）と呼ぶ。ＣＧＩではトラッキングとエコーの２つの方法がある。

トラッキングは、同一の入出力ＣＧＩを用いて入力値をフィードバックする機構である。

エコーは別々の入力、出力のＣＧＩ間のフィードバック機構である。リクエストが進行する間ループして、入力デバイスをサンプルし出力ＣＧＩにECHO_UPDATE 要素を送りエコーを更新する。

３．　設計と実現での問題点

各入力モジュールをプロセスと考えてＣＧＩ入力機能を実現する。

３．１　Ｕｎｉｘにおけるプロセス・モデル

３．１．１　プロセス間通信

Ｕｎｉｘはもともとｔｓｓ専用ＯＳであったので、システムのイベントは端末とディスクが想定されている。このためプロセス間通信の機能は限られている。このイベントはタイミングを与えることであり、タイミングと情報を与えるメッセージの概念とは異なる。

Ｕｎｉｘにおけるプロセス間通信では、パイプ、ソケット、シグナルの３方法がある。ソケットは、バークレイ版Ｕｎｉｘ特有の機能であり、ＡＴＴ版にはない。各方法の特徴を示す。

（１）パイプ

データの到着を待つ、待ち合わせ同期である。ファイルと同様に簡単に使える。Ｓｙｓｔｅｍ Ⅲで高速化された。

（２）ソケット

ネットワーク間通信用なので、使い方が複雑。多機能である。ＡＴＴ版にはない。

（３）シグナル

イベントを与えるのに用いられる。高速に非同期割り込み処理を行なえるが、メッセージを送ることができない。

本稿ではＳｙｓｔｅｍ Ⅲ準拠Ｕｎｉｘを用いるため、プロセス間通信はパイプを用いて実現する。

３．１．２　ａｃｔｏｒ的プロセスの概要

本インプリメンテーションでは、プロセスをＡＣＴＯＲ理論におけるａｃｔｏｒ［米沢７９］と考える。たとえばイベント・キューをひとつのａｃｔｏｒと考えると、イベント・メッセージを受け取りａｃｔｏｒ内部にキューとして保持する。AWAIT_EVENT メッセージを受け取るとキューからイベント・メッセージを

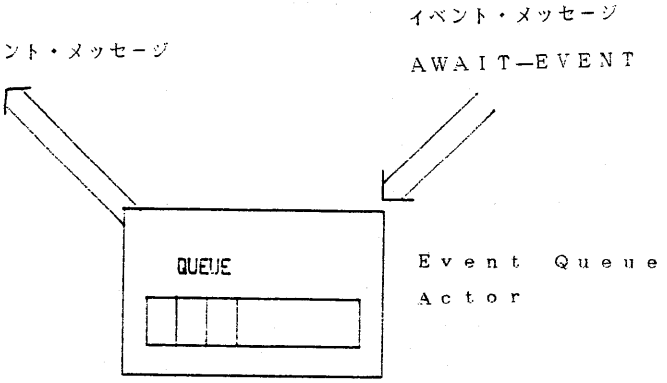

図３．　ａｃｔｏｒの例

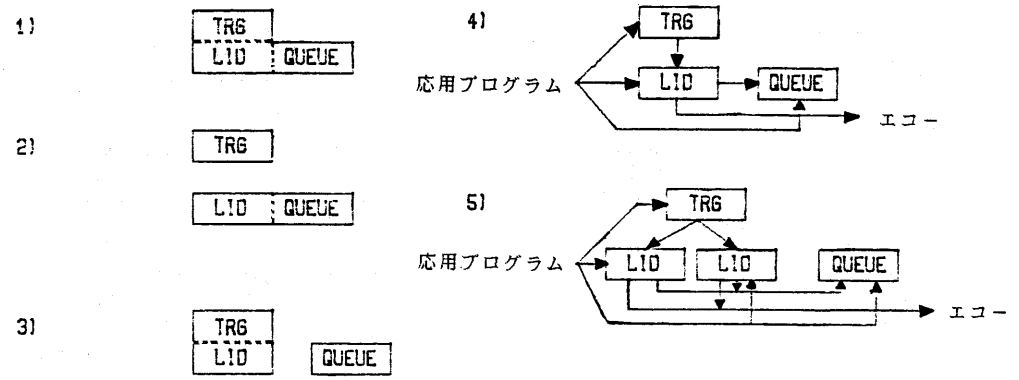

図４．　プロセス・モデル

取り出す。この様にプロセスは、メッセージを受け取ることにより活性化される。プロセスはループに留ることなく速やかにメッセージ処理を終え、メッセージ入力待ちとなる。この様子を図３に示す。

これにより、プロセスは割り込み処理を必要とせず統一のとれた記述が可能である。

３．１．３　プロセス・モデルの検討

ＣＧＩの入力機能メッセージは互いに並列な流れの、次の３項目にまとめられる。

（１）ユーザから各ＣＧＩプロセスへの制御

（２）トリガからＬＩＤ（トリガの発火）

トリガとＬＩＤとイベント・キュー間を流れる。

（３）ＬＩＤから反響部分（トラッキング、エコー）

ＬＩＤから反響部分へ流れる。

図４では、いくつかのＣＧＩプロセス・モデルについてメッセージの流れを表わしている。図４から明らかなように、メッセージの流れはＬＩＤにもっとも多く集まる。したがって、ＬＩＤをプロセスとして独立させるほうがよい［図４、４）および５）］。

３．２　同時イベント・メッセージの問題点

イベント・モードにある複数のＬＩＤがトリガと結合している場合、トリガの発火により複数のイベント・レポートがイベント・キューに送られる。このイベント・レポートを同時イベント・メッセージという。ＣＧＩでは同時イベント・メッセージのイベント・キュー内でのグループ化を規定している。ＬＩＤが複数プロセスの場合、同時イベント・メッセージのイベント・キューへの到着順序は不定である。

同時イベント・メッセージのグループ化に３つの解決案を考えることができる。

（１）ＬＩＤ同士でグループ化を行ない、メッセージをイベント・キューに送る。

（２）ばらばらにイベント・キューに入っている同時メッセージを、イベント・キュー内における操作でグループ化する。

（３）ＣＧＩの仕様である１つのイベント・キューをやめて各トリガにイベント・キューを持たせる。

ＬＩＤによる解決案［図５］

トリガから、イベント・メッセージが、そのトリガと結合しているＬＩＤを順に流れる。

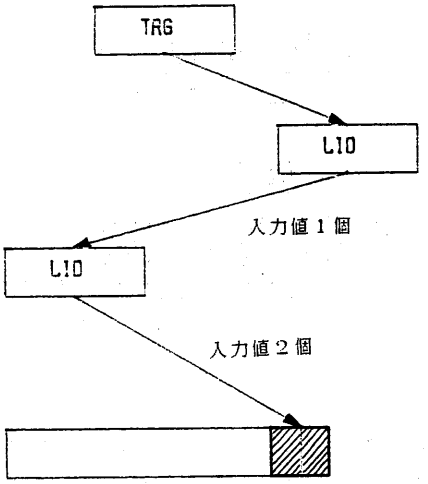

図５．　ＬＩＤによる案

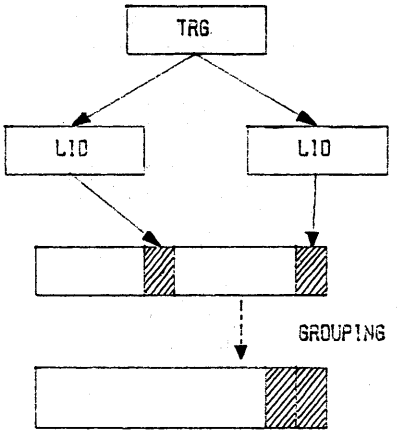

図６．　イベント・キュー内での案

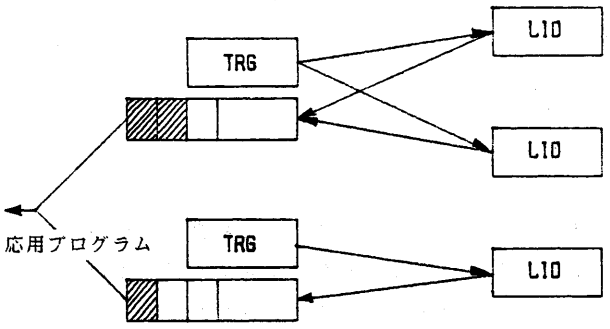

図７．　トリガにイベント・キューを持たせる案

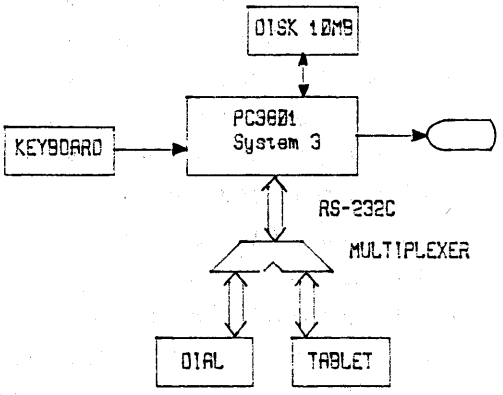

図８．　実現環境

ＬＩＤを通るたびにイベント・メッセージに、入力値を加える。結合しているすべてのＬＩＤを通過後、イベント・メッセージはイベント・キューへ送られる。

この方法の欠点は２つある。１つは、トリガの発火と入力値の時間にずれがあるので、高速連続な入力において入力値が正確でなくなることである。もう１つの欠点は、同時イベント・メッセージがイベント・キューへ入るのが遅れるので、キュー内での順序が、トリガの生成順とは異なることである。

イベント・キュー内での解決案［図６］

この方法は、トリガを発したイベント・メッセージは同時に各ＬＩＤへ送られる。そしてＬＩＤはイベント・キューへ直ちにメッセージを送る。イベント・キューの管理プロセスは同時メッセージをメッセージ・リンク・フィールドとタイム・スタンプ値で判断しグ

ループ化を行なう。この方法の欠点は、キュー内において挿入と削除が行なわれるので、キューのデータ構造が複雑になることである。

トリガにイベント・キューを持たせる解決案［図７］

各トリガがイベント・キューを各々ひとつずつ持つ。トリガから発したメッセージは、連結するＬＩＤへ送られる。ＬＩＤは入力値をトリガに送り返す。トリガはこの入力値をキューに保持する。イベント・メッセージの取り出しの際には、各トリガのキューの先頭のタイムスタンプ値を比較して、最も古いイベント・メッセージを取り出す。

この方法は、ＣＧＩのモデルとは異なるが、ＣＧＩ機能要素の仕様を満たすことができる。

本稿では、（２）のイベント・キュー内での解決案を実現した。

３．３　その他の問題点

ＣＧＩの効率を上げるためにプロセスを細かく分けた場合には、システムのパイプ数の上限に触れることがある。一般的なＵｎｉｘシステムでは同時に使用できるパイプの数は２０程度である。

４．　ＣＧＩインプリメンテーション

４．１　実現環境

コンピュータがＰＣ９８０１，入力デバイスはＲＳ－２３２Ｃマルチプレクサ接続のタブレットとダイヤルを用いる。ＵｎｉｘはＳｙｓｔｅｍⅢ準拠ＰＣ－ＵＸである。図８参照。

４．２　ＶＤＣについて

ＣＧＩではＶＤＣ（Virtual Device Coordinates）として整数型と実数型が選択できる。本インプリメンテーションではＣＰＵがマイクロコンピュータであるので、整数型を用いた。

```
associate(lidp, msglin)      /* associate in LOCATOR LID process */
int lidp;          /* lid pointer */
char meglin;       /* message line */
{   int v,trg;

   sscanf(msglin, "%d%d¥n", &v, &trg);
   if (searchass(lidp, trg)==NFOUND)
   {   lidstl[lidp].asstrg[lidstl[lidp].assnum]] = trg;
       lidstl[lidp].assnum++;
       if (lidstl[lidp].assnum>MAXTRG) errp();
       sprintf(msglin, "%d%d¥n", MS_ASS, trg, msglin);
       pwrite(LOCATOR, lidp, TRG, trg, msglin);
    }
}
```

図９．　アソシエート関数

４．３　実現機能の例

図９は、ロケータＬＩＤのプロセス中に存在するトリガとＬＩＤの結合を生成する関数である。sscanfは文字からの書式付き変換関数、sprintf は文字への書式付き変換関数である。pwriteは他プロセスへのメッセージの出力関数である。

結合情報はＬＩＤごとのリストになっている。この関数では、与えられた結合を捜し、存在しなければリストに加える。そして結合トリガに結合情報を送る。これは、結合情報がトリガとＬＩＤの両方のプロセスに必要だからである。

４．４　実現の現状

ＣＧＩはすべてＣ言語で記述した。

実現した要素は、入力プリミティブが５、制御および属性が１１、反響化が５、状態問い合わせが４、能力問い合わせが１９、合計４４要素である。

ソース・プログラムの大きさは、現在約２０００行である。

現在、バリュエータＬＩＤプロセス、コントロールプロセス（応用プログラムと、他のプロセス間の橋渡し）を除くプロセスが完成しており、各要素の組み合わせをテスト中である。

５．　おわりに

ＣＧＩインプリメンテーションにおいて、Ｕｎｉｘは決して使いやすいＯＳではない。それはＵｎｉｘがデバイスとして端末とディスクしか想定していないためである。ＣＧＩ端末には多くのデバイスが接続されている。このデバイスを高速で扱うためには、リアルタイム処理が必要である。Ｕｎｉｘにリアルタイム機能を加えることは、ＣＧＩにおいて必須である。

今後の問題として、ユーザに対する反応速度の向上がある。ハードウェア、ソフトウェアの両面からの検討が必要である。Ｕｎｉｘのオーバ・ヘッドも問題である。

もうひとつの問題として、ＣＧＩとＵｎｉｘのネットワーク機能の結合がある。ユーザの、ＣＧＩ利用を便利なものとしなければならない。

参考文献

［米沢７９］　米沢明憲：ＡＣＴＯＲ理論について
情報処理Vol.20 No.7 P.580-589 (July 1979)

［ＡＮＳＩ８４］　ＡＮＳＩ／Ｘ３Ｈ３：Virtual Device Interface(VDI) Functional description (March 1984)