プログラミング—言語・基礎・実践— 10−13
（１９９３．１．２９）

制約による描画システムとそのユーザインタフェース

大野敬史　佐渡一広

群馬大学 工学部 情報工学科

描画システムに制約を取り入れる一方法について述べる。図形や図形間に制約を与えることができると複雑な図が容易に作成できるようになる。しかし、制約を対話的に扱うためには、(1) 制約をどのように視覚的に表示するか、(2) 図の構成要素にどのように制約を与えるか、(3) 制約を与えるために複雑になりかつユーザに余計な作業を強いることがないか、が問題となる。本システムでは、非対話的にテキストを与えて描画するための描画言語と、対話的に作画をおこなうグラフィックエディタから構成されており、より効率の良い作画方法を目差している。

A interact drawing system with the constraint

Takashi Ohno　Kazuhiro Sado

Department of Computer Science, Faculty of Engineering, Gunma University

It is very convenient if we can make figures with constraint. There are following problems if we use constraints in the interactive environments: (1) How to express constraints, i.e. what constraint does each picture elements have is not clear, (2) How to define constraints, e.i. how to set a constraint with a mouse, (3) Some works, which are not needless when nonconstraint drawing system, must be done by user. This system has two methods, a constraint drawing language for text oriented use, and an interactive interface with a mouse.

1 はじめに

現在よく使われている描画システムでは、描く図形を、画面上の目的の位置に置いていくだけである。その為描いていく上で幾つかのユーザに対して煩わしい問題点が出てくる。

その例の一つとして、他の図形に関係する座標位置を指定できないことである。今、ある直線 l の二等分点より直線を引きたいとする。このようなことを正確に行なう為には、ユーザは直線 l の二等分点を見つけ出す為に、その両端の座標を読みだし、計算して見つけ出す必要がある。つまり、直線 l の二等分点を指定する方法がシステムには用意されていないのである。

また二つ目として、描かれた図形どうしが互いに独立して存在していることである。図形を描いていく上で、描かれた図形を移動したりして編集し直す作業は少なくない。しかもその作業は一つの図形に対して行なわれることは稀で、大抵他の多くの図形も編集し直すことになる。これは図形どうしが他の図形との位置関係を知らず、独立してその位置に存在する為である。

このような問題を解決するための方法として pic[1] のように関係によって図形を描くという方法がある。しかし、対話的に描画を行なうことができない。本論文では、このように図形や図形間に関係を扱えるようにし、さらにそれに対するユーザインタフェースを取り入れ対話的に描画できるシステムについて述べる。

本システムは図1に示すように、大きく分けて3つの部分「グラフィックエディタ部分」、「部品、制約管理部分」、「制約解法部分」と一つの「描画言語ファイル」からなる。

ユーザには、描画方法として「グラフィックエディタ部分」を介してグラフィカルに描く方法と、「部品、制約管理部分」に対して描画言語を与えて描く方法を提供している。

「制約解法部分」では、ThingLab と同様の方法 [2, 3, 4] を用いている。

「部品、制約管理部分」では描画言語で記述されたテキストを読み込み、描画することができる。これは、ユーザと非対話的な描画方法を提供するだけでなく、他のプログラム等の出力を読みとり、それを描画することもできる。一つの応用例として、アルゴリズムアニメーション [5] を生成するプログラムのアニメーション表示部分の使用が考えられる。アニメーションの描画に制約を用いることで、その生成を容易にすることができる。

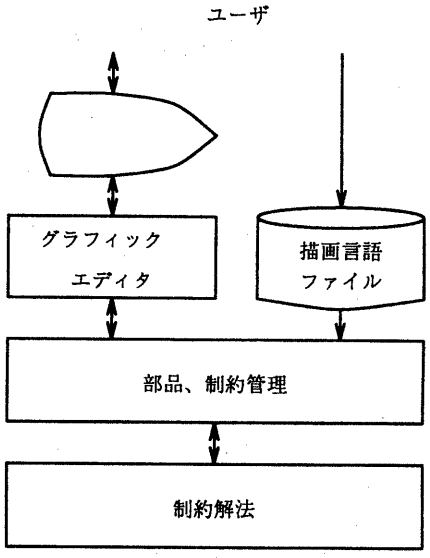


図 1: システム概要

ユーザにとって非対話的な方法では描画が困難であるので、より有用にするためには視覚的で対話的なインタフェースを提供する必要がある。しかし、対話的に制約を扱えるようにするには、幾つか問題点がある。制約を与える図形をどのように指定したらよいか、与えられた制約をどのように指定したら良いか、制約が複雑になってきた時どう対処したら良いか、などである。

以下、描画システムに制約を取り入れた時の、図形に対する制約の編集方法等について、これらの問題点を解決して、どのようにグラフィカルに行なえばよいか、および実際のその方法に基づいたシステムを作成し、その有用性について述べる。

2 描画言語

描画言語はテキストベースで描画を行なうことができるものである。

描画言語は Smalltalk [6] と Bertrand [2] をもとにしており、文法は Smalltalk に準じている。

例えば、点(10,10)から点(200,200)まで直線を引くならば、

```
Line p1:  10@10 p2:  200@200
   name:  #line1.
```

のようする。ここで、最後の name の部分はこの部品のインデックスである。これは、後でこの直線を指定したいとき、例えば、この直線の削除や移動、あるいはこの直線に制約を与える、といった時にこのインデックスを使う。

例えば、今 #line1 と #line2 というインデックスが付けられた直線に対して eq: という制約を与えるとしたら、

```
(usrParts at:  #line1) eq:
   (usrParts at:  #line2)
   name:#const1.
```

というように与える。このうち、usrParts というのは描いた図形を管理している変数である。描いた図形を指定したい時は、この変数とその図形のインデックスを上述のように使い指定する。また、最後の name の部分は図形と同様この制約のインデックスであり、後で指定したい時に使用する。実際に使用したい時には、

```
usrConst at:  #const1
```

とすることで得ることができる。この usrConst というのは与えた制約を管理している変数である。すでに与えてある制約を指定したい時は、この変数とその制約のインデックスを上述のように使い指定する。ここで、実際に図2のような図形を描くにはどうなるかを示す。図2の図形は外側に四角形があり、その内側にそれぞれの辺の中点を結んだ四角形があるような図形である。

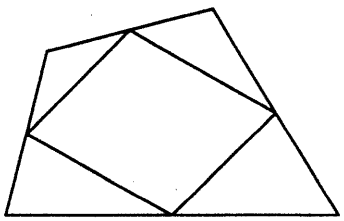


図 2: 作図例

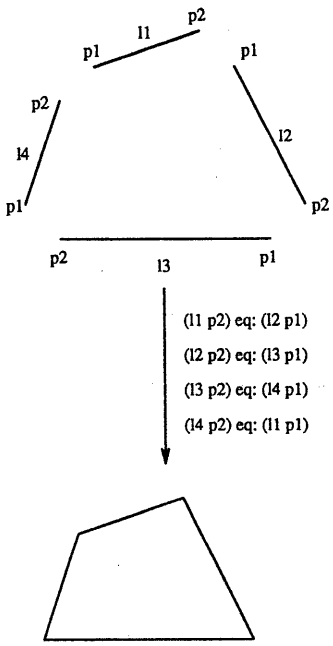


図 3: 四角形の構成

まず、内側と外側の四角形二つを描かなければならない。四角形は図3に示すように四つの辺と四つの制約からなっている。本システムでは ThingLab のように、このような部品と制約の集まりを一つの部品として扱えるようにしている。ここでは Quad という部品としてすでに登録されているとする。そこでこの部品を使い四角形を二つ描くには、

```
MyQuad p1:  10@10 p2:  50@5
   p3:  60@50 p4:  5@50
   name:  #quad1.
MyQuad p1:  110@130 p2:  200@105
   p3:  260@200 p4:  105@200
   name:  #quad2.
```

となる。

次に図4で示すように一方の四角形の頂点がもう一方の四角形の一辺の中点に来るように制約を与えなけ

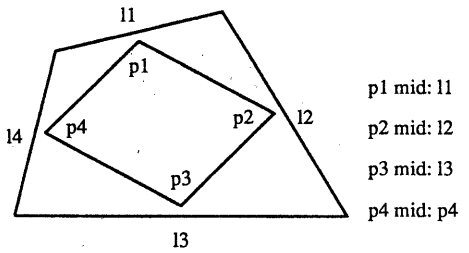


図 4: 例図の構成

ればならない。つまり、点 p1 でいうと、この点は、直線 l1 の中点に来なければならない。そのような制約は、

```
(usrParts at:  #quad1) p1
   mid:  (usrParts at:  #quad2) l1
   name:  #const1.
(usrParts at:  #quad1) p2
   mid:  (usrParts at:  #quad2) l2
   name:  #const2.
(usrParts at:  #quad1) p3
   mid:  (usrParts at:  #quad2) l3
   name:  #const3.
(usrParts at:  #quad1) p4
   mid:  (usrParts at:  #quad2) l4
   name:  #const4.
```

で与えられる。

これで、図2のような図形を描くことができる。

3 対話型作図

本システムでは、OPUS[7] で見られるように、視覚的に制約の扱いを行なうことを目的としている。しかし、OPUS では、制約が一方向であり、また、制約が増えて複雑になってくると対処しづらくなってしまう。このような点を考慮し視覚的な描画方法を実現している。

3.1 対話的操作による問題

制約を扱うためのインタフェースに新たに必要とされる機能は大まかに以下のようなものである。

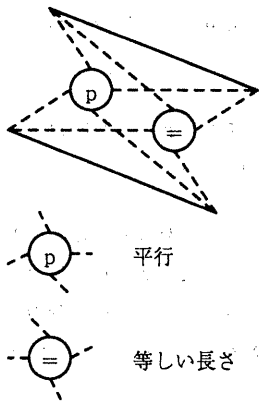


図 5: 制約のグラフィカルな表示例

- 図形を構成している部品の指定。
- 与えられている制約の呈示。
- 与えられている制約の指定。
- 制約を自動的に与える。

これらを実現するには幾つか問題点がある。

まず「図形を構成している部品の指定」である。制約を与えるためには、図形を構成しているどの部品に対して与えるかを指定できなければならない。

二番目が、「与えられている制約の呈示」である。これは三番目の「与えられている制約の指定」と一緒に考えることができる。Sketchpad[8] のように制約を視覚的な図形で表現 (図 5) し、それがどこに与えられているかを実際に描画している画面上に描く方法もある。しかしこの方法では、以下のような問題がある。

- 制約や描いた図形が複雑になった時、画面が見づらくなる。
- 重なりあった図形の一方に対して制約が与えられていると、どちらの図形に制約が与えられているのかがわからなくなる。
- 新たに制約を登録した時、視覚的にどう表示するか、つまりその制約を表す図形をどの様なものにするかも同時に登録する必要がある。

このため、このように制約を図形として表現することはせず、単にテキストのメニューとして何という制

約が与えられているかを呈示することにし、さらに、指定された図形に対して与えられた制約のみを呈示することにしている。

こうすることで、図形や制約が複雑になった時でも、制約を整理して呈示することができる。しかし、この方法の問題点として、対話的に描画を行なっている際、ある制約がどの図形に対して与えられているのかがわからなくなる。単に指定された図形にどんな制約が与えられているかをメニューで表示するだけでは、その与えられている制約の対称となる図形が存在する時、それがどの図形なのかがわからなければならない。そこでこのメニューより制約を選択することで、その制約の対称となる図形を得られるようにしている。

最後の「制約を自動的に与える」では、システム側がどの程度まで自動的に制約を与えるかの判断が問題である。ユーザの期待していないところにまで勝手にシステムが制約を与えてしまっては、逆に使いづらくなってしまう。この為、自動的に与えられる制約は、点と点を重ねた時、「その点どうしが同じ位置にある」という eq: の制約程度である。

3.2 作図の方法

このインタフェースは、テキストベースでは行なえない対話的な描画を可能にするためのものである。つまり、ユーザとテキストを処理する描画システムとの仲介役となっている。そして、前節で述べた描画言語での可能な描画をすべて対話的に行なえるようになっている。

ユーザから与えられた描画手続きはテキストを処理する描画システムが解釈できるようなテキストに直され、与えられる。その与えられたテキストを処理し、結果を再びインタフェースが受けとり、画面に反映することで、対話的に行なうことを実現している。

制約を付加的に取り入れるためには、前提として制約を取り入れた為にシステムになにかしらの制限がかけられてはならないということである。つまり、制約を取り入れていない描画システムの機能を制限することなしに、新しく制約を扱うための機能を取り入れる必要がある。このようなことを考慮し、制約を取り入れているので、単に図形を描画するならば、制約を取り入れていない描画システムと変わりなく描くことができる。

そして、制約を扱う時に上述したような新たな方法を提供しなければならない。以下に、これらの実現方法を実際の例を交えて説明する。

3.3 部品の指定

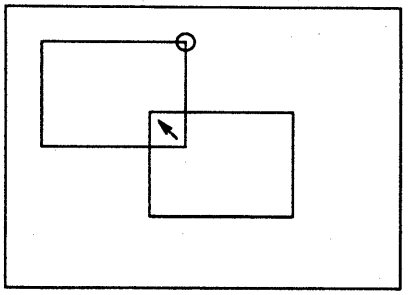
例えば、今、四角形が描かれていて、この四角形の右上の頂点に対して何か制約を与えたいとする。このような時には、単に四角形を指定するだけではなく、それを構成している部品の内の右上の点を指定する必要がある。

図 6(a) のように四角形が二つ描かれている時、現在カーソル (矢印) がある地点で、制約を与えるために左上の四角形の右上の頂点 (丸で囲ってある点) を指定したいとする。

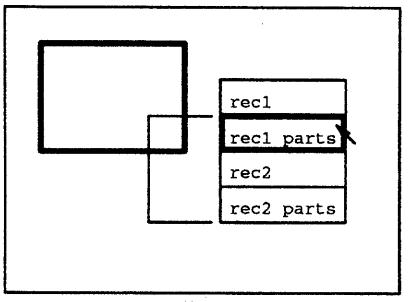
まず、この位置で指定することにより、図 6(b) の中央にあるようなメニューが表示される。このメニューにはこの位置で指定できるものが含まれている。今は、左上の四角形 (rec1)、その四角形の部品 (rec1 parts)、右下の四角形 (rec2)、その四角形の部品 (rec2 parts) が指定できる。カーソルをそのメニュー内のどこかに移動することによりその対応する図形が強調される。現在 rec1 parts にカーソルがあるため左上の四角形が強調されて表示されている。

次にメニューの rec1 parts を選ぶと、図 6(c) のようにメニューが変わる。四角形の部品を選んだので、四角形を構成している部品の指定を行なうメニューになる。先ほどの段階で、rec1 を選んでいたら、図形の指定は終了し、左上の四角形自体を指定したことになる。今ここで指定できるのは、四角形を構成している上の線 (line1)、その線の部品 (line1 parts)、右の線 (line2)、その線の部品 (line2 parts)、下の線 (line3)、その線の部品 (line3 parts)、左の線 (line4)、その線の部品 (line4 parts) の計八つである。今指定したいのは右上の頂点であるが、その頂点は、上の線の右の点と、右の線の上の点と二種類の方法で指定できる。この例ではどちらでもいいのでとりあえず上の線から選ぶことにする。現在カーソルが line1 parts にあるので上の線が強調されている。

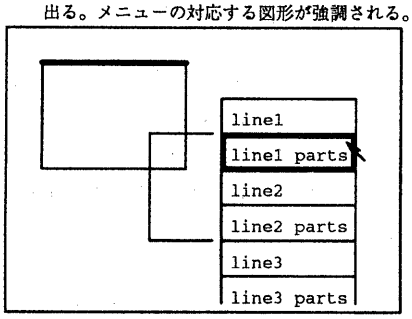
さらにメニュー line1 parts を選ぶと、図 6(d) のようにメニューが変わる。今度は線の部品を選んだので、指定できるのは、左の点 (point1)、その点の部品 (point1 parts)、右の点 (point2)、その点の部品



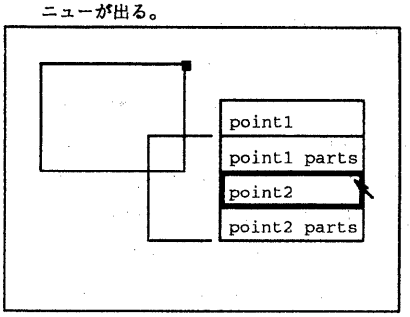
(a) 丸印の頂点を指定する。



(b) 図 (a) の位置で指定することでメニューが出る。メニューの対応する図形が強調される。



(c) 図 (b) のメニューを選択するとさらにメニューが出る。



(d) 図 (c) のメニューを選択するとさらにメニューが出る。これで点の指定ができる。

図 6: 図形を構成している部品の指定

(point2 parts) の計四つである。現在カーソルが point2 にあるので右の点が強調されている。ここでこの point2 を選ぶことで目的の部品を指定することができる。

この指定を行なった時点で、「グラフィックエディタ部分」は「部品、制約管理部分」に対して、

```
(usrParts at:  #rec1) line1 point1
```

というテキストコマンドを送る。つまり、このコマンドをテキストでシステムに送るのと同様なことを、今、対話的に行なったのである。

3.4 制約の呈示

例えば、ある部品に対してどのような制約が与えられているかを調べたいことがある。このような時、その部品に対してどのような制約が与えられているか、ということの他に、その与えられている制約が他のどの部品に与えられているかもわかるようにする必要がある。また、その調べたい部品を構成している部品に対しても再帰的に与えられている制約も呈示する必要がある。

これはどういうことかというと、具体的な例として、今直線に対してどのような制約が与えられているかを調べたいとする。この時その直線に与えられている制約を呈示する他に、その直線を構成している二つの点に与えられている制約、およびそのそれぞれの点を構成している x 座標値、y 座標値に与えられている制約を呈示する必要があるということである。なぜこのように部品に対して与えられている制約を呈示する必要があるかというと、今の例でいうと、この直線はこの直線自体に与えられた制約のみでなく、その部品に与えられた制約にも依存して位置関係が変化するためである。

図 7(a) のような図形の左上の四角形に対してどのような制約が与えられているかを調べたいとする。

制約の呈示に対して、先ほどの図形の部品の選択で左上の四角形を選ぶと、図 7(b) のようにメニューが表示される。このメニューより、現在この四角形に対しては何も制約が与えられておらず、line2 に対して eq という制約が、さらに line3 の point1 に対して mid という制約が、line3 の point2 に対して eq という制

約が与えられていることがわかる。

ここで図7(c)のように、メニューのline3のpoint1にカーソルを移動することによりそれに対応する点が強調されて表示される。また、図7(d)のように、メニューのmidにカーソルを移動することによりその制約が与えられている対象となる部品、つまりここでは中央の水平線が強調されて表示される。これにより、この四角形の右下の点は、中央の水平線に対してmidという制約が与えられていることがわかる。

テキストベースでは、特にこのように制約を呈示する方法は提供していない。なぜなら、対話的に行なっていると、以前与えた制約がどう与えたか忘れてしまう可能性があるが、テキストベースなら、その制約を与えている行を調べれば済むからである。しかし、与えられた制約をインスペクトすることでどの図形に対して与えられているのかを読みとることは可能である。ここで示した例では、midの制約をインスペクトすることで、左上の四角形の右下の点と中央の水平線を得ることはできる。

ここで、呈示されるこれらの制約はユーザが与えたものとしている。つまりある図形内の部品に与えられている制約、例えば前節で説明した、Quadという図形は、それを構成している部品の線に対して与えられている四つの制約は、ユーザが与えたものではなく、図形を新しく登録した時自動的に与えられる制約である。これらの制約はここで説明した「与えられている制約の呈示」では対象外とし、呈示されないようになっている。

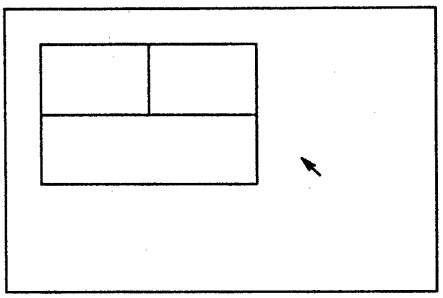
3.5 制約の指定

これは以前に与えた制約を削除したいといった時に必要となる。方法としては上述した「与えられている制約の呈示」方法と全く同じように行なう。

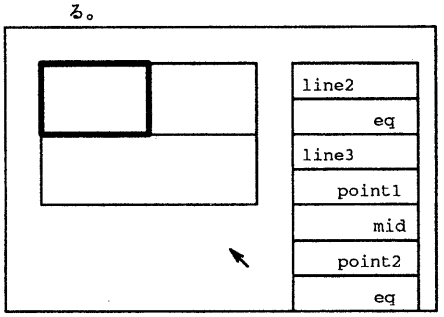
例えば、ある制約を削除したいといった時、制約の削除に対して、制約の呈示方法と同じように制約を選ぶことでその制約を削除することができる。

対話的な方法では、「与えられている制約の呈示」の方法と同じであるが、「グラフィックエディタ部分」と「部品、制約管理部分」とのやりとりは異なる。

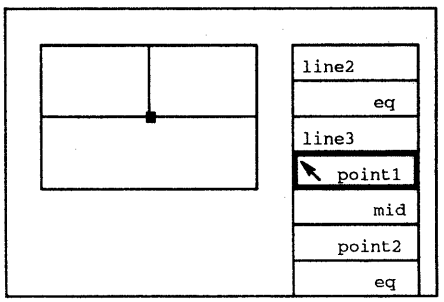
「与えられている制約の呈示」の方では、特に「部品、制約管理部分」に対して新たに操作するコマンドは送られなかった。しかし、この「与えられている制



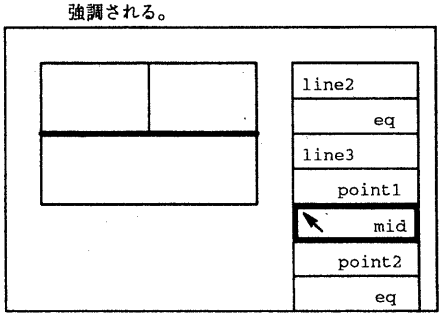
(a) 左上の四角形に与えられている制約を呈示する。



(b) 左上の四角形の指定でメニューが出る。



(c) メニューの図形を選択すると対応する図形が強調される。



(d) メニューの制約を選択すると対応する制約が与えられている図形が強調される。

図 7: 与えられている制約の呈示

約の指定」では、なにかしらのコマンドが送られる。

例えば、先ほどの呈示例で選んだ mid という制約を指定して削除するとしたら「グラフィックエディタ部分」はこの制約のインデックスを仮に #mid1 とした時、「部品、制約管理部分」に対して

```
(usrConst at:  #mid1) delete.
```

というコマンドを送る。

4 おわりに

制約を取り入れた描画システムにおける、テキストベースの非対話的な方法と、それをグラフィカルで対話的に行なう方法、及びそれぞれの結合について述べた。

グラフィックエディタで制約を扱うために、図形を階層的に指定する方法を用い、さらに、制約をテキストメニューとして呈示し、それを選択することでどこに与えられているかを確認できる方法を用いた。

この方法により、図形を構成している部品に対しても容易に制約を与えることが可能となり、さらに、複雑な制約が与えられている時でも描いた図形の邪魔にならずに呈示、指定ができるようになっている。そして、制約を付加的に取り入れ、そのためにシステムに制限がかからないように留意しているため、確実に制約によって、その有用性は向上しているといえる。

しかし、これらの方法による問題点として次のようなものがある。

1. ユーザ操作が複雑になる。制約を与える時に、図形が重なりあっていて簡単に指定できない場合、その図形を部品としている図形を選び、「この図形のこの部品」というように選ばなければならない。さらに、制約の指定の時も、「この制約」として選べるのではなく、その制約が与えられている図形を選び、「この図形に与えられているこの制約」というように選ばなければならない。

2. 制約の呈示を指定した制約のみに限っているので、全体的にどのように制約が与えられているかを見渡すことができない。

3. 制約を取り入れた分、その解法等を行なわなければならないので、その分システムが繁雑になってしまい、応答速度の問題がある。

4. どの制約を使うか、どこに制約を与えるかといった判断はすべてユーザ任せとなっているので、ユーザに対して逆に負担になることがある。

これらのことに関する解決法として、標準的な制約を自動的に判断することを検討している。

参考文献

[1] Brian W. Kernighan and Rob Pike. *The UNIX Programming Environment.* Bell Telephone Laboratories, 1984.

[2] Wm Leler. *Constraint Programming Languages.* Addison-Wesley, 1988.

[3] Alan Borning. The programming language apsects of ThingLab, a constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and System*, Vol. 3, No. 4, pp. 353–387, October 1981.

[4] Alan Borning, Robert Duisberg, Bjorn Freeman-Benson, Axel Kramer, and Michael Woolf. Constraint hierarchies. *OOPSLA '87 Conference Proceedings*, pp. 48–60, October 1987.

[5] 金谷延幸, 大野敬史, 佐渡一広. アルゴリズムアニメーションシステム. 第３３回プログラミングシンポジウム, pp. 13–24, 1992.

[6] Goldberg AR. *Smalltalk-80 The Language and its Implementation.* Addison-Wesley, 1983.

[7] S.E. Hudson et al. Interactive specification of flexible user interface displays. *ACM Transactions on Information Systems*, Vol. 8, No. 3, pp. 269–288, 1990.

[8] Ivan E. Sutherland. Sketchpad: A man-machine graphical communication system. *AFIPS Spring Joint Computer Conference*, pp. 329–346, 1963.