筑波大学大学院修士課程

理工学研究科修士論文

三次元ビジュアルプログラム編集環境の構築

宮　下　貴　史

平　成　12　年　3　月

筑波大学大学院修士課程

理工学研究科修士論文

三次元ビジュアルプログラム編集環境の構築

宮　　下　　貴　　史

主任指導教官　電子・情報工学系　田中 二郎

目次

論文要旨 **1**

1 序論 **2**

2 三次元グラフエディタにおける要素技術 **4**

2.1 三次元図形の操作技術 4

2.2 三次元空間の情報表示技術 5

2.2.1 自動レイアウト機能 5

2.2.2 半透明表示機能 6

2.3 三次元図形の作成技術 7

2.3.1 ワイヤーフレームモデル作成技術 7

2.3.2 サーフェースモデル作成技術 7

2.3.3 ソリッドモデル作成技術 8

3 三次元グラフエディタ **9**

3.1 これまでのグラフ操作と自動レイアウト 9

3.2 三次元での自動レイアウト 10

3.3 直接操作手法 11

3.4 拡張直接操作手法 12

3.5 操作手法と自動レイアウトとの統合の問題点 13

3.5.1 ユーザの意図しないレイアウト 13

3.5.2 グラフ内のノード移動の難しさ 14

3.6 操作手法と自動レイアウトとの統合の改善 15

3.6.1 レイアウトの工夫 15

3.6.2 グラフ内ノード移動への対応 15

3.7 提案した手法を用いたグラフ作成例 16

3.8 三次元グラフエディタの実装 18

4 三次元グラフエディタの評価実験 22

4.1 実験内容 22

4.1.1 実験結果 24

4.1.2 実験結果に関する考察 26

5 三次元グラフの効率的な入力法 28

5.1 効率的な入力 28

5.2 これまでの入力法 29

5.3 入力操作の問題点 29

5.3.1 入力を妨げるレイアウト 29

5.3.2 マウスによる三次元移動の難しさ 29

5.3.3 操作の繁雑さ 30

5.4 入力操作の改善 30

5.4.1 レイアウトの一時停止操作 30

5.4.2 操作平面を用いた三次元空間の移動 31

5.4.3 少ないクリック数と少ないドラッグ&ドロップ数 31

5.5 新入力法 31

5.5.1 複数の操作を一度にまとめた操作型 31

5.5.2 再利用型 35

5.6 各入力法のクリック数とドラッグ&ドロップ数 37

5.7 複数の入力法を組み合わせた効率的な入力法 39

5.8 新一筆書き型入力法、セミオーダー型入力法とテキスト利用型入力法を組み合せた作成例 39

6 関連研究 42

6.1 三次元グラフに関する研究 42

6.2 効率的な図形の入力法に関する研究 42

7 まとめ 44

A 三次元グラフエディタの詳細 50

A.1 三次元グラフエディタの作成 50

A.1.1 ノードの生成と配置の工夫 50

A.1.2 グラフのグループ化の工夫 50
A.1.3 三次元スプリング・モデルの実装の工夫 51
A.2 layout.cpp ソース 51
A.3 glbox.cpp ソース (一部) 61

図一覧

3.1 複雑なレイアウト 10
3.2 視点の見え方による違い 13
3.3 ユーザの意図しないレイアウト 14
3.4 グラフ内ノード移動の難しさ 15
3.5 三次元アイコン 17
3.6 APPEND プログラム 17
3.7 ノード生成 18
3.8 サブグラフ作成 18
3.9 サブグラフの結線 18
3.10 APPEND 作成 18
3.11 実行画面 19
3.12 結線可能なノード 20
3.13 ノードをドラッグ 21
3.14 エッジの生成 21
3.15 グラフ作成中の自動レイアウト 21

4.1 サンプルプログラム 23
4.2 工夫したノード移動機能の評価 24
4.3 評価結果 25
4.4 操作慣れによる操作時間への影響 25
4.5 工夫したレイアウト機能の実験結果 26
4.6 工夫したノードの移動機能の実験結果 27

5.1 入力を妨げるレイアウト 30
5.2 操作の難しさ 30
5.3 細胞分裂型入力法 32
5.4 一筆書き型入力法 33

5.5 重ね合わせ型入力法 34
5.6 新一筆書き型入力法 35
5.7 セミオーダー型入力法 36
5.8 テキスト利用型入力法 37
5.9 プログラム 41
5.10 新一筆書き型入力法１ 41
5.11 新一筆書き型入力法２ 41
5.12 セミオーダー型入力法 41
5.13 テキスト利用型入力法 41

要旨

近年の計算機の急激な能力向上を背景として、コンピュータグラフィックス (CG) の三次元化が普及している。視覚的プログラムの三次元化によってプログラムの操作や作成手法、自動的に再配置し図形を分かりやすくする自動レイアウト機能も三次元向けにする必要がある。本論文では、グラフの直接操作手法と自動レイアウトを統合した手法を提案し、評価実験によって有効性を示す。これまで二次元のグラフで自動レイアウトのアルゴリズムとして用いられたスプリングモデルを三次元向けに改良した三次元スプリングモデルや操作の工夫等について述べる。また、一筆書き入力法などを組み合わせた効率的な入力法について提案し、入力法の有効性について作成例を用いて述べる。

第 1 章

序論

近年の計算機の急激な能力向上を背景として、ビジュアルプログラム (VP) の研究が多く行われている [1, 2, 3, 4, 5]。 VP とは図形・アイコンなどの視覚的表現を用いて表されたプログラムである。これまで多くの VP はノードやエッジで表現されている [6, 7]。ノードやエッジで表された VP の多くは、二次元空間で扱われてきた。我々は、 VP の三次元化によって以下のメリットがあると考えている [8]。

取扱うプログラム量の拡大： VP は量がかさばる傾向にあり、大規模プログラムに対応できないという問題点がある。プログラムを三次元に配置すれば、一画面で扱えるプログラム量は飛躍的に向上する。

リアリティの向上：我々の住む現実世界は三次元空間である。プログラム環境も三次元空間で構築した方がリアルな表現が可能となり、図形がわかりやすくなる。

レイアウトの自由度向上：二次元では、いくら見やすくレイアウトしようとしても図形が交差したり重なったりすることが多々あるが、三次元空間では自由度が一つ増えるため重なりや交差を避ける事ができる。

我々は三次元化によって得られるメリットを享受できる三次元ビジュアルプログラミングシステム (**3D-PP**)[9, 10] の研究を進めている。

これまで二次元 VP の多くはインタラクションデバイスとしてコンピュータディスプレイやマウスを用いてきた。三次元物体を表示、操作するためにヘッドマウントディスプレイ (HMD) やデータグローブなどの三次元向けの特殊なデバイスがある。しかし、我々はコンピュータディスプレイやマウス等の一般的な環境で表示や手軽な操作を行なうことに興味がある。コンピュータディスプレイやマウスは二次元物体の表示や操作に親和性のあるデバイスであるので、三次元物体の表示や操作に新しい手法が必要になる。

これまで二次元図形の操作には直接操作手法 [24] が多く用いられてきた。我々がこれまで研究してきた二次元 VP を操作する時も直接操作手法を多く用いた。図形を操作した後、我々は VP の可読性の向上に自動レイアウト機能 [27] を用いてきた。

三次元図形の場合、図形の操作と視点移動操作が必要になる。通常、ユーザの視点の移動や三次元物体の配置にはワールド座標系、カメラ座標系などが使われる。作成された三次元図形はワールド座標を用いて三次元空間に配置される。視点の位置はカメラ座標によって配置される。我々は、三次元 VP を操作する時に二次元 VP の場合と同様な直接操作によって三次元図形の操作が実現すれば、操作が容易になると考えた。しかし、直接操作手法を三次元図形の操作にそのまま適応するだけでは操作が十分容易になるとは言えない。例えば、三次元空間に浮かんだ幾つかの三次元図形の相対的位置を把握するには視点の移動を頻繁に行う場合が多い。我々は、ワールド座標にあるノードやエッジを直接操作、またカメラ座標における視点の直接操作を容易にする操作手法が必要であると考えた.

一方、視点の移動に関してであるが、そもそも コンピュータディスプレイは二次元物体の表示に対して親和性のあるデバイスである。これまでコンピュータディスプレイはユーザからの視点から一度に一方向からの画像を写すので、三次元物体の形状や配置を理解するために視点を移動する必要がある。三次元空間では物体の構造を把握するために視点の移動を頻繁に行なうことが多々ある。しかし、三次元 VP の構造を理解するために視点の移動だけでは十分とは言えない。これまで、二次元 VP の構造を理解するために我々は、ノードやエッジを見やすく分かりやすい位置に自動的に配置する自動レイアウト機能を適用してきた。我々は三次元 VP に自動レイアウト機能を適用し、三次元向けに拡張することによりノードやエッジを自動的に再配置しプログラムの可読性向上が可能であると考えた。

これまで二次元 VP では、直接操作と自動レイアウトを別々に処理していた。操作と自動レイアウトを変則的に処理することができなかった。我々は、直接操作と自動レイアウトの統合によってそれぞれの変則的な処理が可能になるだけでなく、三次元 VP の編集操作がより効率的になると考え、システムを実装した。

第 2 章

三次元グラフエディタにおける要素技術

我々は、三次元のグラフを作成するにあたりユーザの操作性の向上が重要であると考えている。本章では、操作性向上のための技術と三次元のグラフの理解を容易にする技術について述べる。

2.1 三次元図形の操作技術

これまで二次元空間で図形を扱うためのインタラクションデバイスとしてマウスを用いてきた。三次元空間で図形を扱うためにはマウスと三次元向けの特殊なデバイスがある。三次元向けの特殊なデバイスは一般的なインタラクションデバイスではない。我々は、一般的な環境下での三次元グラフの作成に興味がある。そこで、二次元空間での図形操作で一般的なマウスを用いて三次元の図形を操作する技術が必要になる。マウスを用いた三次元の図形操作には、間接操作と直接操作がある。間接操作では、図形の姿勢や位置を疑似的なトラックボールなどの GUI の部品を用いて行う操作 [11] や三次元空間の壁に正面図、側面図、立面図に相当する影を表示して影を使って行う操作 [12] がある。直接操作では、直接図形をドラッグ＆ドロップすることによって姿勢や位置の操作を行う。直接操作は間接操作と比べて直観的な操作であると言える。我々は直接操作手法を用いて三次元物体を操作するために、ピック処理 [13] を適用している。ピック処理とはマウスカーソルでどの物体を操作しているか三次元空間の情報から三次元物体の情報を取り出す処理である。この処理では、マウスカーソルからコンピュータディスプレイに垂直な直線を引き、この直線と交わった最初の三次元物体を操作対象としている。

また、三次元空間では、図形の姿勢の 3 自由度と位置の 3 自由度の合計が 6 自由度である。6 自由度を持った図形をこれまでと同じようにマウスで操作することは難し

い。図形のすべての自由度の操作を可能にしてしまうとかえって操作は複雑になってユーザは混乱してしまう。そこで図形の自由度を制限することによってマウスでも三次元の図形を容易に操作できると思われる。

これまで三次元の図形の位置を把握するためのデバイスとして三次元向けの特殊なディスプレイがある。三次元向けの特殊なディスプレイは一般的なディスプレイではない。我々は、マウスの場合と同様に一般的なディスプレイを用いて三次元の図形を表示する。そこで、ディスプレイに表示された三次元図形の位置を把握し易くする技術が必要になる。三次元図形の位置を把握し易くするために三面図を用いた方法がある。しかし、三面図を用いる方法は直観的でない。ユーザは、三次元の図形の位置を把握するために三方向の位置を把握する必要がある。手軽に三次元図形の位置を把握するために光延ら [25] は付加情報を用いた方法を提案している。付加情報は三次元空間に表示された "地面" や "影" 等を指す。"地面" はユーザが三次元図形の位置を把握するための基本位置となる。三次元図形から "地面" に写る "影" によってユーザは三次元図形の位置の把握を容易にする。

2.2 三次元空間の情報表示技術

一般に図形情報は、文字情報とくらべて量がかさばる傾向にある。しかし、現在あるディスプレイでは、情報をすべて二次元で表示するには限界があり、 Small Screen Problem[20] と呼ばれる。二次元空間では扱うことのできる情報量に限界があったが三次元空間ではより多くの情報を扱うことができる。しかし、ディスプレイは二次元の図形の表示に親和性があるデバイスである。三次元の図形を表示する場合は表示に工夫が必要になる。例えば、三次元空間で階層構造のグラフを作成する場合、グラフが画面からはみ出してしまうことがある。また、深い階層を持ったグラフは複雑になることがありグラフを表示するための工夫が必要である。また、図形が複雑に表示された場合ユーザは図形から有用な情報を得る事ができない。そこで情報の構造を明確に美しく見せる工夫が必要である。このような工夫として、情報の構造を明確に美しく見せる「自動レイアウト機能」と包含グラフの表示を可能にする「半透明表示機能」を挙げる。

2.2.1 自動レイアウト機能

グラフは、具体的な情報やシステムなどの要素間の関係を表現するのに便利である。グラフは通常、木、有向グラフ、無向グラフ、複合グラフの各クラスに分類できる。

各クラスの特徴を活かしたレイアウトは実際的であり、かつ効果的である。ユーザにとって有用なグラフにするレイアウトについて多くの研究がされている [27]。有効な木のレイアウトアルゴリズムとして Waker[21] の線形時間アルゴリズムがある。このアルゴリズムでは、根をグラフの最上位に配置し、各頂点は各階層に対応した平行線上に左から右へ頂点が配置される。有向グラフのアルゴリズムとしては、 Sugiyama ら [22] のアルゴリズムが提案されている。各頂点を対応する各階層に配置し、頂点を結ぶ辺は階層上で折れ点を持つ折れ線として配置される。無向グラフでのレイアウトアルゴリズムは、グラフ理論指向のアルゴリズムと力指向のアルゴリズムに分類できる。グラフ理論指向のアルゴリズムでは、 Batini ら [23] が優れた方法を提案している。交差する辺の数を除去し、辺を折れ点を用いて直交化する。最後に辺の総線長と描画面積が最少になるよう頂点を配置する。力指向のアルゴリズムでは、 Eades[28] がスプリングモデルを提案している。スプリングモデルでは、辺をバネに置き換えてバネの力が系の最少エネルギー状態になるように配置する。グラフを自動的にレイアウトし、良い図を描くことができると図の描き換えなど面倒な操作を簡単化することができる。良くない図はユーザに混乱や過ちを招く可能性が高いが、良い図はユーザに図の意味を素早く正確に伝達する可能性が高くなる。

2.2.2 半透明表示機能

三次元図形の立体表現を立体モデルといい、主なモデルとしてワイヤーフレームモデル、サーフェースモデル、ソリッドモデルがある。ワイヤーフレームモデルは、立体を輪郭線群によって表現されたモデルである。サーフェースモデルは、立体を多角形群により表現されたモデルである。ソリッドモデルは、立方体、球、シリンダなどのプリミティブと名付けられた単純な立体図形を組み合わせて作られたモデルである。これらのモデルを入れ子にして、階層構造を持ったグラフを表現することができる。しかし、これらのモデルをそのまま入れ子にしても階層構造のグラフの表現には十分ではない。階層構造のグラフの表現に有効な半透明表示手法がある。暦本は半透明表示を利用した階層構造情報のための三次元視覚化技法として InformationCube[15] を提案している。 InformationCube によって Unix ディレクトリは半透明表示されたキューブの入れ子での表現が可能になった。この視覚化技法では、階層が深くなると透明度が減少して見えにくくなる。グラフは比較的簡素に表現されて、ユーザにとって理解が容易な三次元グラフの階層表現が可能になる。

2.3 三次元図形の作成技術

三次元図形を作成する技術として大きく三つに分けることができる。“ワイヤーフレームモデル作成技術” と “サーフェースモデル作成技術” と “ソリッドモデル作成技術” である。“ワイヤーフレームモデル作成技術” では、ユーザが点や線を空間に描画することで三次元図形を作成する。精密に図形を作成するという点では優れているが効率的に作成するという点では十分であるとは言えない。“サーフェースモデル作成技術” では、多角形群を用いて立体を作成する。ユーザは各多角形の表面に色や画像を張り付けたりすることでワイヤーフレームモデルよりもリアルなモデルを作成することができる。しかし、多角形の表面について操作する手間が増えるので十分効率的に作成できるとは言えない。“ソリッドモデル作成技術” では、システム側で予め用意した雛型 (または、基本図形) を使い、組み合わせることで三次元図形を作成する。精密な図を作成するには幾つもの図形の組み合わせを必要とする。精密な図形よりもむしろ簡素な図形の作成に向いていると言える。ソリッドモデルは、CSG 表現 (Constructive Solid Geometry) を用いて作成される。CSG 表現とは平面や曲面などの基本図形を組み合わせることによって複雑な図形の作成を可能にする表現である。ここでは、“ワイヤーフレームモデル作成技術” と “サーフェースモデル作成技術” と “ソリッドモデル作成技術” について述べる。

2.3.1 ワイヤーフレームモデル作成技術

ユーザは点と線を基本にワイヤーフレームモデルを作成する。効率的な作成の工夫として、曲線の作成に関数を用いていことがある。関数を使うことによって立体の曲面を容易に作成できる。安福ら [16] は曲線や直線をドローツールとして利用した作成手法を提案している。三次元空間にある曲線や直線を水平移動や回転移動することによって移動の軌跡から簡単に立体図形を作成することを可能にしている。

2.3.2 サーフェースモデル作成技術

多角形表面をよりリアルに表現するには、色や画像を張り付けただけでは十分ではない。例えば、ざらざらした表面を多角形で表現しようとすると膨大な数の多角形が必要になる。そこで、物体表面の法線ベクトルを画素ごとに意図的に変更することで疑似的にざらざらした表面や凹凸感を陰影のみで表現するバンプマッピング [14] という方法がある。

2.3.3　ソリッドモデル作成技術

図を作成する際、我々は同じような図を繰り返し作成することがある。頻繁に作成、利用するような基本図形については予め用意することで一から図を作成する必要はなくなる。また、基本図形を幾つか組み合わせることによって複雑な図形を簡単に作成することができる。基本となる組み合わせ方には、和、差、積の 3 種類がある。和の演算は二つの図形を結合した形状を生成する。差の演算は、一方の図形からもう一方の図形を差し引いた形状を生成する。積の演算は、二つの図形の共通部分の形状を生成する。

我々は、ソリッドモデル作成技術が三次元グラフの作成を容易にする技術として重要であると考えている。

第 3 章

三次元グラフエディタ

これまで我々が 3D-PP で考えていたグラフの作成は生成したノードをエッジでつなぐという単純な操作の繰り返しから成り立ち、作成後に自動レイアウトを行って来た。しかし、グラフ作成中において必ずしも三次元化のメリット、特にレイアウトの自由度拡大というメリットを十分に活かしているとは言えない。

我々は、図形の作成中でもレイアウトの自由度拡大というメリットを享受するためには三次元オブジェクトの操作手法に自動グラフレイアウト機能を結びつける工夫が必要であると考えた。本論文では、三次元上において自動グラフレイアウトを伴う三次元オブジェクト操作手法とプロトタイプのシステム実装について述べる。

3.1 これまでのグラフ操作と自動レイアウト

これまでのグラフの作成方法、例えば ViewPP[17] の場合まずユーザはノード生成メニューから生成したい種類のノードを選択することによってノードを生成する。ノードの表面にある引数をクリックし、別の引数へドラッグ&ドロップすることによってエッジを生成し、結線することができる。グラフの美的基準を満たすように整形する自動レイアウト機能 [27] は、グラフの理解や記憶を容易にするために必要であるのでシステムは作成されたグラフを自動レイアウトし、再配置することによってグラフの可読性を向上させた。

しかし、三次元空間に配置できる情報量は二次元空間の場合と比べて多い。情報量の多いグラフをユーザが容易に理解するにはグラフの適切なレイアウトが重要になる。これまでの作成方法において、自動グラフレイアウトするまで三次元上で作成中のグラフのレイアウトは必ずしも適切であるとは言えない。

例えば、図 3.1にある複雑なレイアウトのグラフはノードの重なりやエッジの交差

が幾つかある。現在の視点から重なって見えるノードや交差して見えるエッジについては視点の移動によって解決できる。しかし、実際に重なっていたり交差しているノードやエッジに対しては視点の移動だけでは解決できない。適切でないレイアウトのままグラフの作成を続けることはユーザに混乱を招くことになる。

そこで、我々はこれまで別々に行われていたグラフの操作手法と自動レイアウト機能を統合させることで、一定の可読性を保ちながらグラフを作成することができる操作手法を提案する。

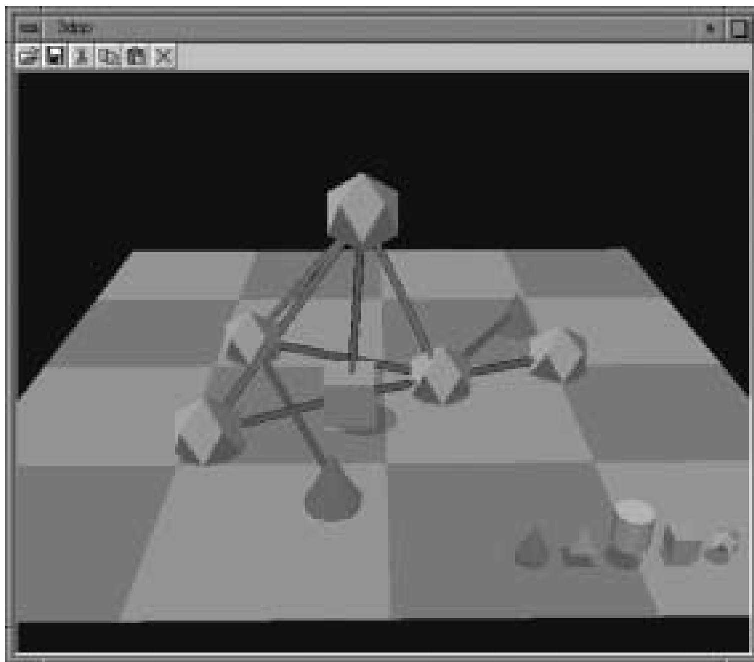

図 3.1: 複雑なレイアウト

3.2 三次元での自動レイアウト

二次元空間での無向グラフのレイアウトに有効な Eades のスプリング・モデル [28] がある。このモデルは 2 種類のバネが使われる。すなわち隣接するノード間をつなぐバネと隣接しないノード間に斥力だけを与えるバネである。隣接するノード間に働く力 f_s は

$$f_s = c_1 \log(d/c_2)$$

により与えられる。ここで d はノード間の距離、 c_1 と c_2 は定数とする。 $d > c_2$ のとき f_s は引力、 $d < c_2$ のとき斥力、 $d = c_2$ のときノード間に力は働かない。また、隣

接しないノード間に働く力 f_r は

$$f_r = c_3/d^2$$

により与えられる。ここで c_3 は定数とする。このような力 f_s と f_r をできるだけ緩和するように各ノードを (c_4 × そのノードに働く力) ずつ移動させることにより、すべての隣接するノード間の距離を c_2 の近傍に収束させる。ここで c_4 は定数である。二次元空間でスプリング・モデルを適用した時は、 X 方向 Y 方向に対してノード間距離を

$$d = \sqrt{x^2 + y^2}$$

としてノード間に働く力を計算している。ノード間距離は三次元の場合には、単に次元を一つ追加し X 方向 Y 方向 Z 方向に対してノード間距離を

$$d = \sqrt{x^2 + y^2 + z^2}$$

とすることで計算することができる。単にこの部分を変更するだけで二次元のスプリング・モデルは、簡単に三次元に拡張することが可能である。本研究では、我々は三次元スプリング・モデルを適用し、三次元空間でグラフを自動レイアウトできるようにした。

3.3 直接操作手法

ユーザが容易に二次元物体を操作するためには、直接操作手法 [24] が適用されたユーザインタフェースが非常に有効である。直接操作手法とは、扱おうとしている対象と操作が視覚化され、可逆的で逐次的な操作や複雑なコマンドの構文を直接的な操作に置き換えた手法である。例えば、自動車の運転を用いて説明する。運転手はフロントガラスを通して景色を直接見ることができる。車が左に曲るには運転手は左にハンドルを切れば直接的に車を操作することができる。応答は素早く、それに伴う景色の変化は曲り具合いを調整するフィードバックとして働く。これに対してコマンド入力では「左へ３０度」と入力し、さらに新しい景色を見るためのコマンドを入力する。運転手はハンドルを使った直接操作手法によって自然で直感的な操作をすることができる。また、直接操作手法について学習も使用も容易で、長い間操作を覚えていられるというメリットもある。しかし、単に直接操作手法を適用するだけでは三次元空間の操作系に適用するだけでは十分ではない。

3.4 拡張直接操作手法

三次元物体を直接操作するための手法、拡張直接操作手法の有効性が光延らや神谷によって示されている [25, 26]。拡張直接操作手法とは、直接操作手法に付加情報を用いた手法と拡張ドラッグ＆ドロップ手法を組み合わせ、操作を拡張した手法である。

付加情報を用いた直接操作手法とは、付加情報をオブジェクトと同時に表示することで効率的に直接操作を行なうことができる手法である。オブジェクト以外の付加情報として人間の感覚の基準となる「地面」とそれに写るオブジェクトの「影」を画面に対して表示する。付加情報を利用することで、ユーザの空間把握を助ける。

また、ドロップしようとしているオブジェクトを、ドロップ可能な位置にドラッグできているか、ユーザが判断することは難しい。例えば、図 3.2でドラッグ中のオブジェクトは、一見その下のオブジェクトにドロップ可能なように見えるが、実際には重なっておらず、ドロップできない。これは、ディスプレイという二次元のデバイスを用いているために、ユーザに対して奥行き方向の情報が不足していることから起こる問題である。この問題に対応して、拡張ドラッグ＆ドロップでは図 3.2の上の画像のようにディスプレイの画面上で二つのオブジェクトが重なって見える場合には手前のオブジェクトから奥のオブジェクトへドロップできるようにした。スクリーン上のマウスカーソルの位置をユーザの視線とする。マウスカーソルの位置に存在するオブジェクトをドロップ対象とする。ユーザは、実際には三次元のオブジェクトを一般的な二次元ドラッグ操作と同様の操作感で、容易にオブジェクトのドラッグ＆ドロップを行うことができる。

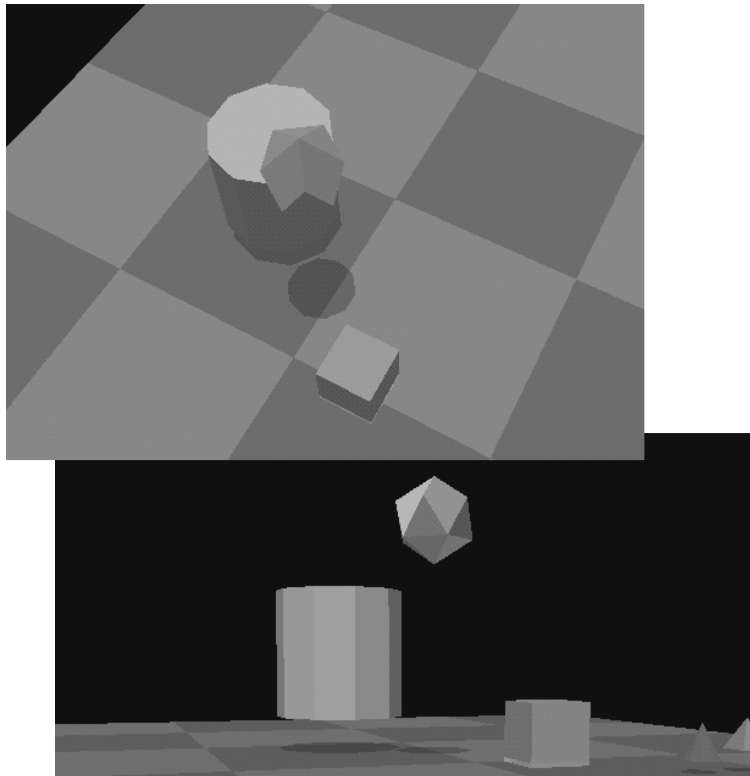

図 3.2: 視点の見え方による違い

また、サーフェス表現だけでは画面表示に表現力が足りないので、半透明表現、ワイヤフレーム表現を導入している。これらの表現を手前のオブジェクトや操作中のオブジェクトに用いることで、これらのオブジェクトが他のオブジェクトを隠蔽してしまい、ユーザの理解の負担となることを減らした。また、半透明表現、ワイヤフレーム表現の導入によって画面表示のバリエーションが広がり、例えば、内部に包含されたオブジェクトの表現等も可能になった。

我々は拡張直接操作手法に三次元スプリング・モデルの自動レイアウト機能を統合させることによってユーザがグラフを作成中であっても一定の可読性を維持することができる手法を実装した。

3.5 操作手法と自動レイアウトとの統合の問題点

3.5.1 ユーザの意図しないレイアウト

作成中は、エッジで直接つながっていないような複数のノードや直接関係ないサブグラフが混在する。プログラムはグラフとして充分ではない。そのまま三次元スプリ

ングモデルで自動レイアウト処理を行なった場合、ノードやサブグラフ同士が離れた位置に配置されプログラムの可読性を悪くするレイアウトになってしまうことがある。図 3.3は、一つのグラフがのエッジの削除によって二つのサブグラフに分けられた場面を表している。互いのサブグラフは矢印の方向へ自動レイアウトされる。これは、ユーザに混乱を招き認知負荷を増加させてしまうことになる。

3.5.2 グラフ内のノード移動の難しさ

ユーザがグラフ内のノードを特定の位置に配置したい時、事前に自動レイアウトされたグラフはユーザの操作によってレイアウトが崩れてしまう。システムは崩れたグラフに対して自動レイアウトをする。しかし、ユーザが特定の位置に配置したノードも自動レイアウトによって他のノードと供に再配置されてしまうために、ユーザがノードを配置したい位置に移動することを難しくしてしまう。図 3.4では、ユーザがノード A を矢印 1 の方向に移動しノード B の位置に配置しようとしている。しかし、自動レイアウトによってノードは他の繋がっているノードから斥力を受けるので矢印 2 の方向へ移動し元のノード A に近い位置に配置されてしまう。

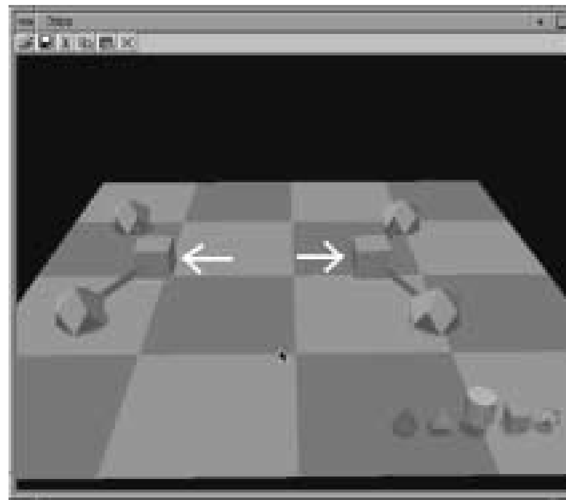

図 3.3: ユーザの意図しないレイアウト

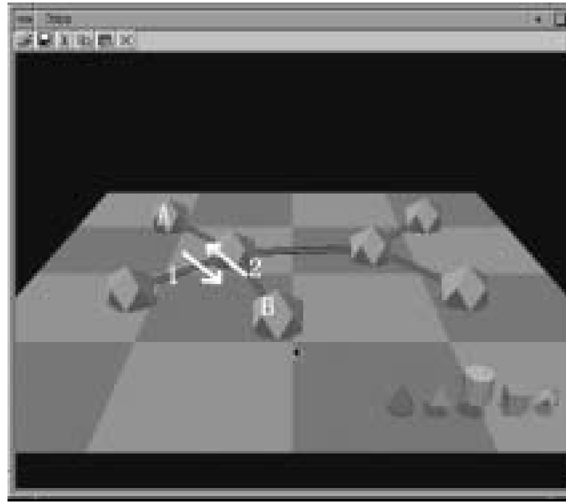

図 3.4: グラフ内ノード移動の難しさ

3.6　操作手法と自動レイアウトとの統合の改善

我々はグラフ作成時のレイアウトの改善策を二つ提案し、実際に三次元グラフエディタに組込んだ。

3.6.1　レイアウトの工夫

直接関係ないサブグラフや関係のないノード同士については敢えてレイアウトをしない。関係のあるサブグラフ内やノード同士では別々に自動レイアウトを行い、各々のグラフの可読性は維持される。また、元々関係のなかったサブグラフやノードがエッジの生成によって関係付けられると一つのグラフとして自動レイアウトが行われ、エッジの削除によって別々の関係のないグラフとされ、各々のグラフ内で自動レイアウトをする。

3.6.2　グラフ内ノード移動への対応

ユーザが移動したグラフ内のノードに追随するように他のノードを自動レイアウトする。ユーザは、グラフの可読性を維持しつつ、ユーザの配置したい場所にノードを移動することが可能になる。また、ユーザはグラフ内のノードを移動するだけでエッジで繋がれたグラフ全体を移動することができるというメリットを得ることになる。

3.7 提案した手法を用いたグラフ作成例

我々が提案したレイアウト方法を用いてグラフが作成され、自動レイアウトされていく様子を複数のグラフを連結した APPEND プログラム作成例を用いて説明する。APPEND プログラムは複数の木のデータを順番に一本のリストデータにするプログラムである (図 3.6)。我々は三次元アイコンを図 3.5に示すように定義し、図 3.6の APPEND プログラムを元に三次元空間にグラフを作成する。

操作１：データノードとコンスノードの生成

三次元アイコンを用いることによってノードを作成する (図 3.7)。ここでは、球の形をした三次元アイコンをマウスの右ボタンでドラッグすることによって３Ｄアイコンと同じ位置に少し大きな球 (ノード) が生成される。ユーザはドラッグしたまま、適当な位置にノードを配置する。ここでは、データノード (球) とコンスノード (立方体) を生成する。

操作２：データノードの結線とレイアウト

はじめに繋ぎたいノードをマウスの中ボタンでドラッグする。ドラッグ中のマウスカーソルをエッジで繋ぎたいノードへ重ねる。マウスカーソルを重ねたらドロップし、エッジが生成される。ノードが隣接しているのでバネ力が働き、自動レイアウトが行われる (図 3.8)。同様な操作で四つのデータノードをコンスノードに結線する。三つの木のサブグラフを作成するが、各々のサブグラフ内ではノードが自動レイアウトされる。しかし、サブグラフ同士では自動レイアウトをしない。

操作３：サブグラフ同士の結線

ユーザは操作２で作成された三つの木グラフから二つの木グラフを作成する。二つのデータノードを持つ木グラフのコンスノードをクリックし、一つのデータノードを持つ木グラフのコンスノードでドラッグ＆ドロップすることによって二つのグラフを結線する。二つのグラフを結線する際、各々のグラフが結線を行い難い位置に配置されてしまったので一方のグラフのノードを引っ張って操作しやすい位置に再配置する。結線された二つのグラフは一つのグラフであるとシステムが判断して自動レイアウトをする (図 3.9)。

操作４： APPEND プログラムの完成

まず、データノードやコンスノードの場合と同様にゴールノードと出力ノードを三次元アイコンから生成する。操作3で作成されたグラフをゴールノードに結線する。さらに、ゴールノードに出力ノードを結線するとすぐに自動レイアウトされ、APPEND プログラムが作成される (図 3.10)。

ユーザは一定の可読性を維持しながら、APPEND のプログラムを作成することができる。

図 3.5: 三次元アイコン

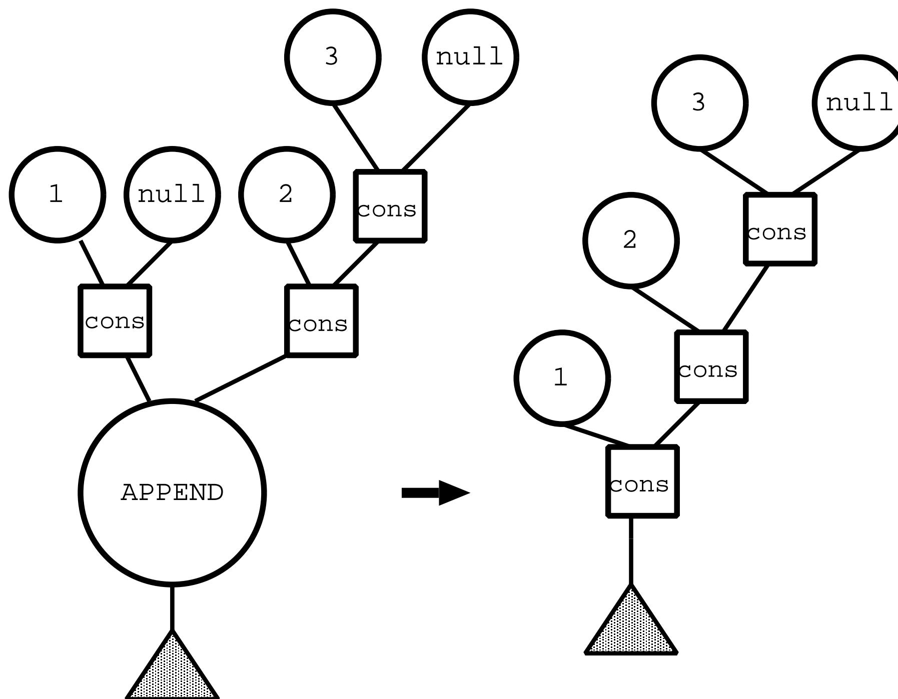

図 3.6: APPEND プログラム

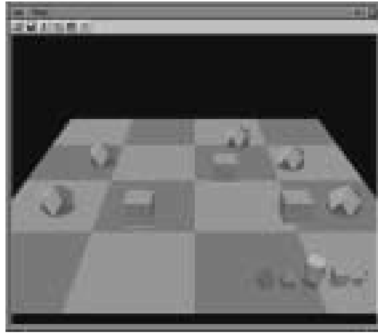

図 3.7: ノード生成

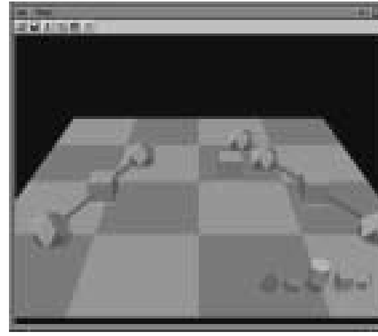

図 3.8: サブグラフ作成

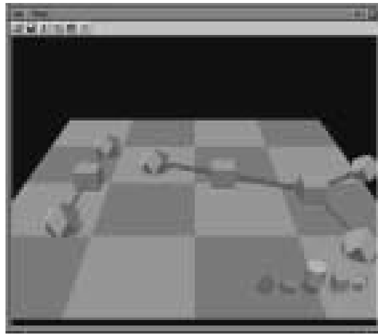

図 3.9: サブグラフの結線

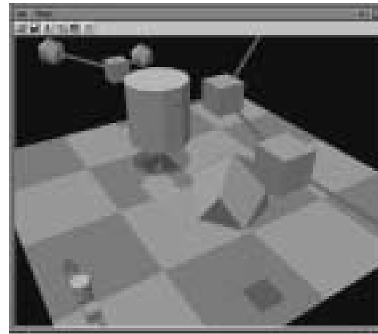

図 3.10: APPEND 作成

3.8　三次元グラフエディタの実装

三次元グラフエディタは、 GUI ソフトウェアツールキットを使用し、 Unix(IRIX6.3) 上で実現した。使用言語は C++ である。実行画面は図 3.11である。

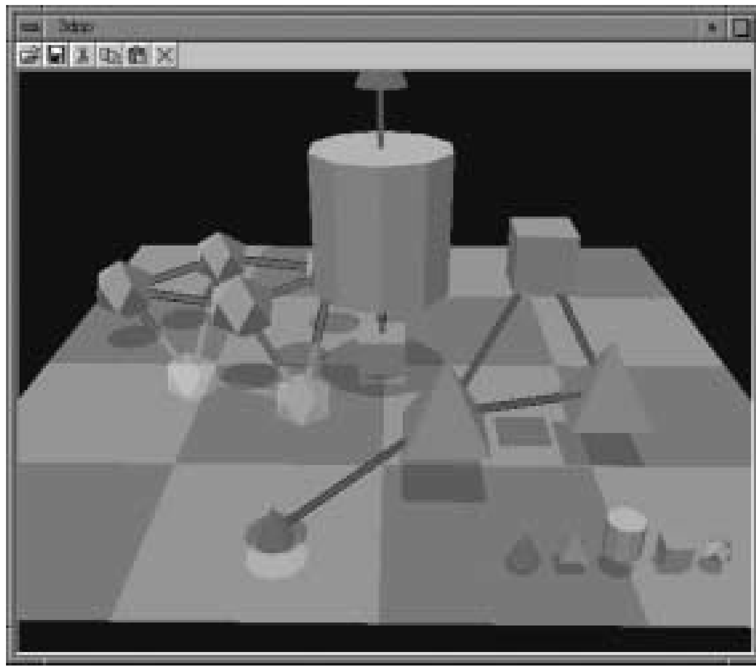

図 3.11: 実行画面

実装されている機能は、視点の移動、ノード生成、ノードの移動、エッジ生成と削除、自動レイアウトである。すべての操作を、マウス操作のみで行うことができる。

画面には、ノードとエッジの他に仮想的な地面とそれにうつるノードの影を描いている。これらはユーザがノードの位置を三次元空間で把握することを助ける。また、ユーザに自由に視点の変更をさせることによってノードの位置関係の把握を容易にしている。

視点移動は、オブジェクト以外の背景や地面をドラッグすることで行う。マウスの x-y 軸移動を、空間の x-y 軸 回転量に対応させている。この際、マウスの移動方向と空間の回転方向を一致させ、ユーザにとって直感的で容易な操作を可能にした。

ノード生成は、三次元アイコンという地面上に予め用意された小さなオブジェクトをドラッグすることで行われる。ユーザがマウスカーソルを三次元アイコンの上に重ねると三次元アイコンは点滅し、どの種類のノードを選択したか確認できる。ドラッグによって生成されたノードはフレーム表示され、そのまま移動操作をすることができる。

ノードの移動は、拡張直接操作手法によって二次元ドラッグ＆ドロップと同様な操作ができる。ユーザは、操作したいノードをドラッグする。ドラッグ中はフレーム表示されるので、どのノードを操作しているかユーザは確認することができる。ノード

はユーザのマウスの移動と一致して移動する。配置したい場所でドロップすることによってユーザはノードを配置することができる。

エッジの生成は、始点となるノードをドラッグ、マウスカーソルを移動し終点となるノードでドロップすることで行われる (図 3.13)。また、我々はスクリーン上のマウスカーソルの位置をユーザの視線とし、その位置に存在するノードを結線の対象とする工夫をした。図 3.12では、ノード 1、ノード 2 に対してユーザは奥行きを気にする事なくマウスカーソルを重ねるだけでドロップし、結線できる。

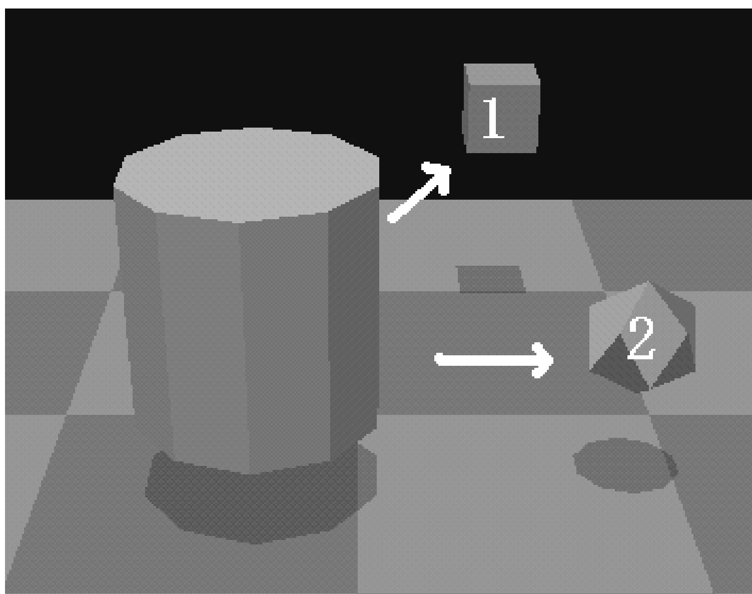

図 3.12: 結線可能なノード

ユーザが始点となるノードをドラッグするとノードはフレーム表示される。ドラッグ中のノードが、背後のノードやエッジを覆い隠さないためにフレーム表示をする。また、フレーム表示によりユーザはどのノードをエッジの始点として選択したか確認できる。また、終点となるノード上にマウスカーソルを重ねるとノードが点滅し、どのノードをエッジの終点として選択しているのか確認することができる (図 3.14)。エッジの削除は、既に生成されたエッジの一方のノードをドラッグし、もう一方のノードをリリースすることによって行われる。また、エッジの生成操作はノードの移動とは割当てたマウスボタンのを変えているので、別々に操作可能としている。

自動レイアウトについては、次のような実装をしている。システム内部ではノード同士での結線の有無を監視している。結線が存在するとシステムは Eades のスプリングモデル [28] から各ノードのバネ力と斥力を算出し、二つの力の合力からノードのレイアウトを行う (図 3.15)。レイアウトの工夫として、結線または関係あるノードやグ

ラフを一つのグループを見なしてノードの自動レイアウトを行う。グラフ内ノードの移動については、ドラッグしたノードにはバネ力と斥力を与えず他のノードにはバネ力と斥力を与えることによってドラッグしたノードを中心に自動レイアウトが行われる。また、グラフ内のノードを移動の途中に自動レイアウトをしてしまうと、移動中にも関わらずドラッグしたノードもレイアウトされてしまい思うように操作ができないので、ノードの移動中はグラフをレイアウトしない。ユーザが同じサブグラフ内のノードをクリックすることで位置を固定できるようにした。ノードの位置が固定されるので自動レイアウトとは関係なくノード配置することができる。クリックし固定されたノードは半透明表示される。

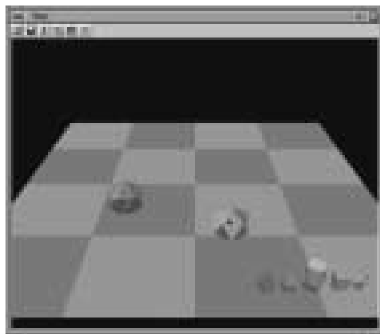

図 3.13: ノードをドラッグ

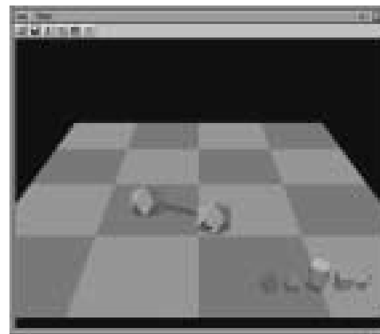

図 3.14: エッジの生成

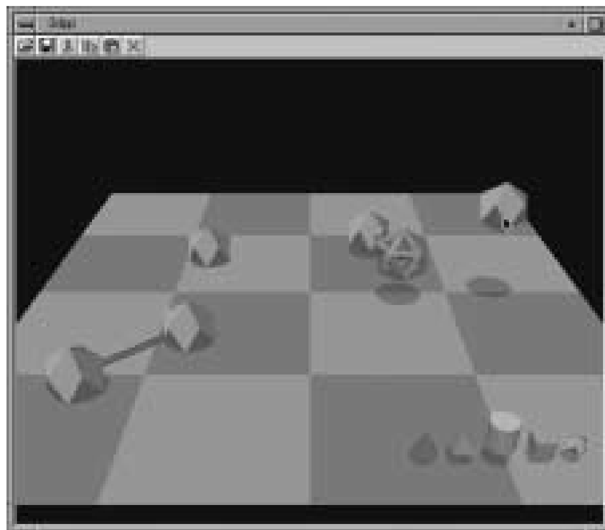

図 3.15: グラフ作成中の自動レイアウト

第 4 章

三次元グラフエディタの評価実験

我々が提案した手法の有効性を明らかにすることを目的として評価実験を行った。拡張直接操作手法と三次元スプリング・モデルを統合したシステムと統合していないシステム、を使い被験者の協力のもとで我々は実験を行った。また、統合したシステムは工夫したレイアウト機能、工夫したノードの移動機能を持っている。それぞれの機能について稼働しているシステムと稼働してないシステムを実装し、我々は評価実験を行った。

4.1 実験内容

実験環境： ハードウェアの構成は前章と同様、コンピュータディスプレイ、マウスを使用して表示、操作を行う。自動レイアウトのアルゴリズムについては同一であるが、統合していないシステムでは自動レイアウトボタンを画面に設定し押下することで自動レイアウトを可能にしている。

被験者： 本研究室の学生 7 人を被験者として実験を実施した。全員マウスの使用には熟知しているが、三次元物体の操作や視点の移動について慣れている被験者と慣れていない被験者がいた。

方法： タスクは三つある。最初のタスクは二次元ビジュアルプログラムで表現されたサンプルプログラム (図 4.1右) を参考にして、統合したシステムと統合していないシステム使って三次元グラフを作成することである。我々は、両システムを用いて三次元グラフの作成時間を計測する。

ユーザは、図 4.1の左側にある APPEND プログラムを右側にあるサンプルプログラムに編集する。サンプルプログラムは、二次元で表すと図にあるようにエッ

ジが交差してしまう。しかし、三次元空間ではレイアウトの自由度が拡大するのでエッジの交差を減らすことができる。

二番目のタスクは、 APPEND プログラム (図 3.6左) を参考にして工夫したレイアウト機能が稼働したシステムと稼働してないシステムを使い三次元グラフを作成することである。我々は、三次元グラフの作成時間を計測する。

三番目のタスクでは、図 4.2にあるように一つのサブグラフを地面の角に用意する。被験者は、サブグラフを操作しエリア A からエリア B まで移動させる。我々は工夫したノード移動機能が稼働したシステムと稼働してないシステム用いた場合の操作時間を計測する。

計測開始のタイミングは、被験者がマウスで最初にドラッグすると同時に時間の計測を開始する。グラフ作成の実験では、被験者がグラフを作成を終了しレイアウトが完了した時点で計測を終了とする。ノードの移動実験では、被験者がサブグラフをエリア A でドラッグすると同時に計測を開始する。サブグラフの 2 つのノードが両方ともエリア B 上に移動した時点で計測を終了とする。

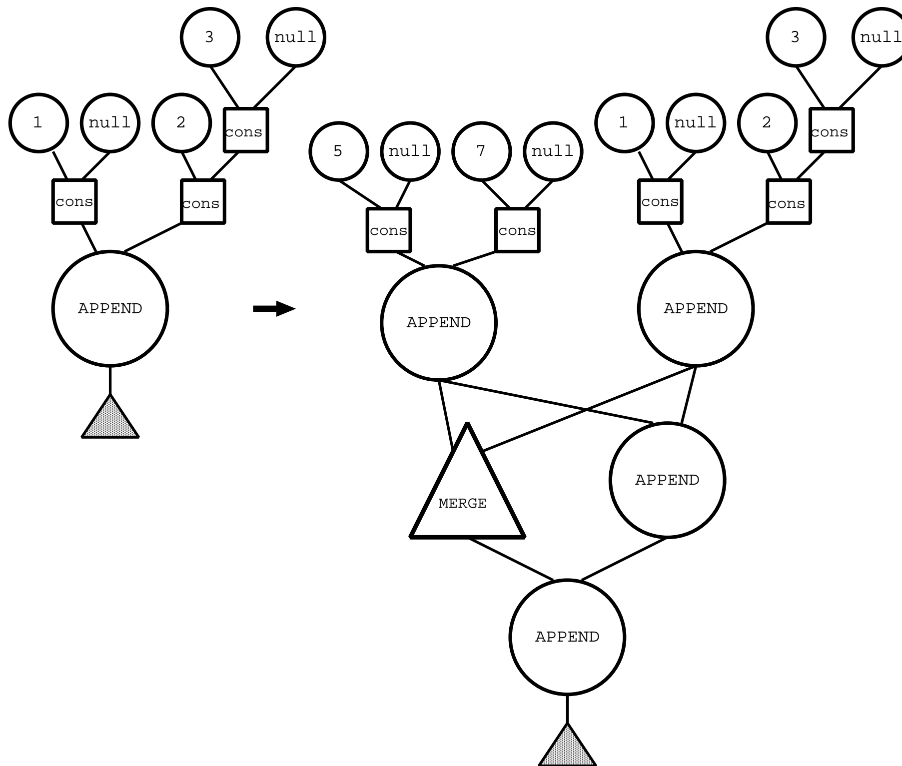

図 4.1: サンプルプログラム

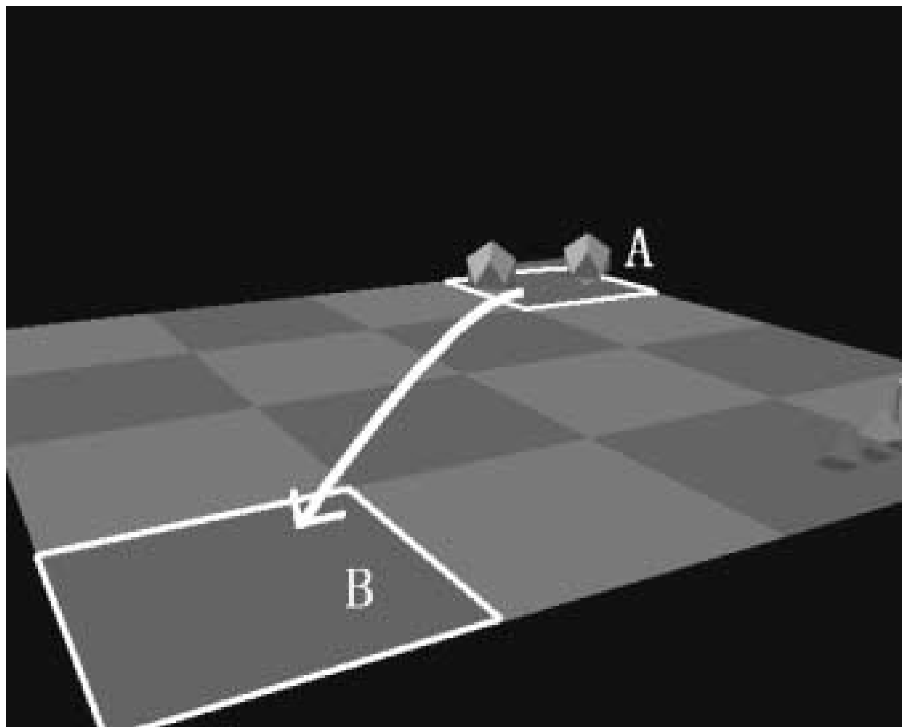

図 4.2: 工夫したノード移動機能の評価

実験条件： 最終的に作成されたグラフは、指定通りのグラフの作成が完了しているだけでなくノードやエッジが重なっていたりエッジが交差していないことを操作終了の条件とする。また、自動レイアウト機能による配置の結果は初期配置に依存するので被験者が作成したグラフのレイアウトが最終的に指定のグラフのレイアウトと異なっていてもよいこととする。

4.1.1 実験結果

統合したシステムの評価実験の結果を図 4.3に示す。どの被験者に対しても、拡張直接操作手法と三次元スプリング・モデルを統合したシステムの方が短い操作時間で三次元グラフを作成している。また、各被験者の計測時間の差にばらつきが生じている。各被験者の中で比較的三次元グラフエディタに慣れている者は A,B,C,E である。二つのシステムの計測時間に大きな差はない。三次元グラフエディタに慣れていない者は、 D、 F、 G で計測時間に大きな差がある。

工夫した自動レイアウト機能の実験結果を図 4.5に示す。工夫したレイアウト機能が稼働しているシステムを使う方が被験者は早くグラフを作成することができることがわかる。

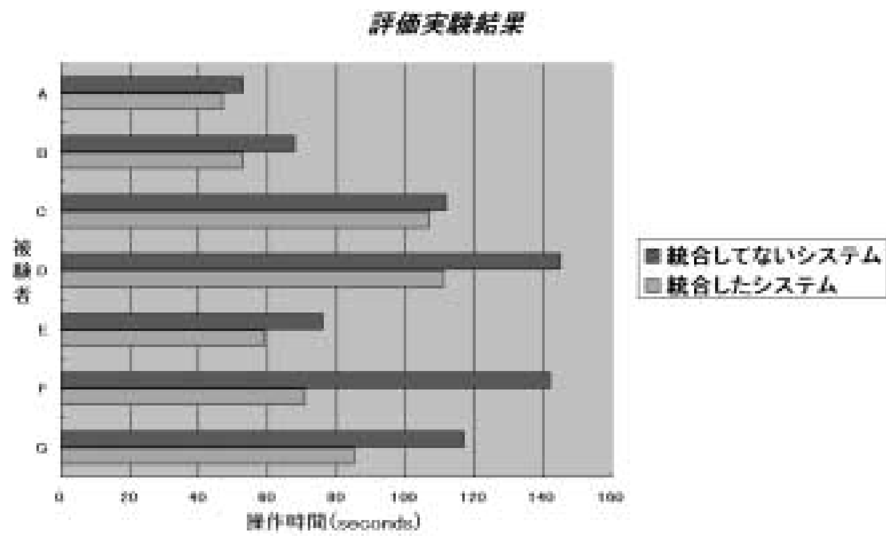

図 4.3: 評価結果

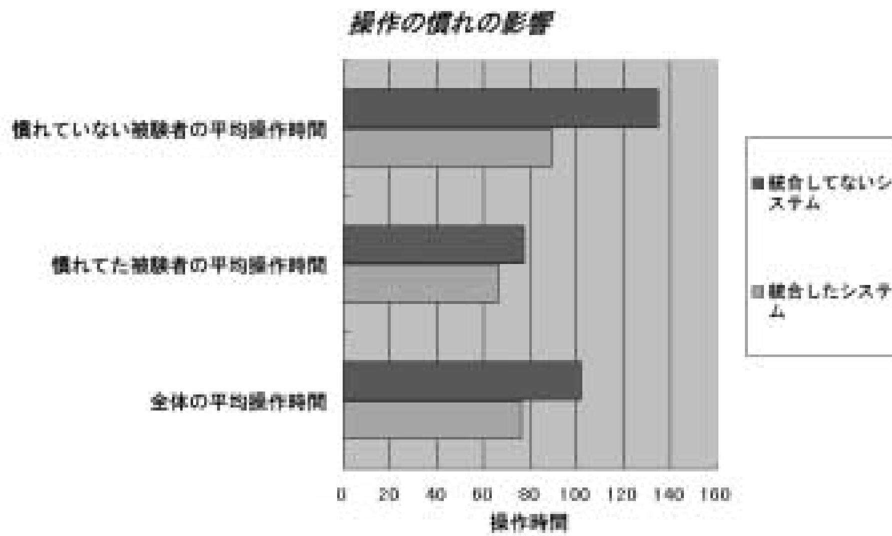

図 4.4: 操作慣れによる操作時間への影響

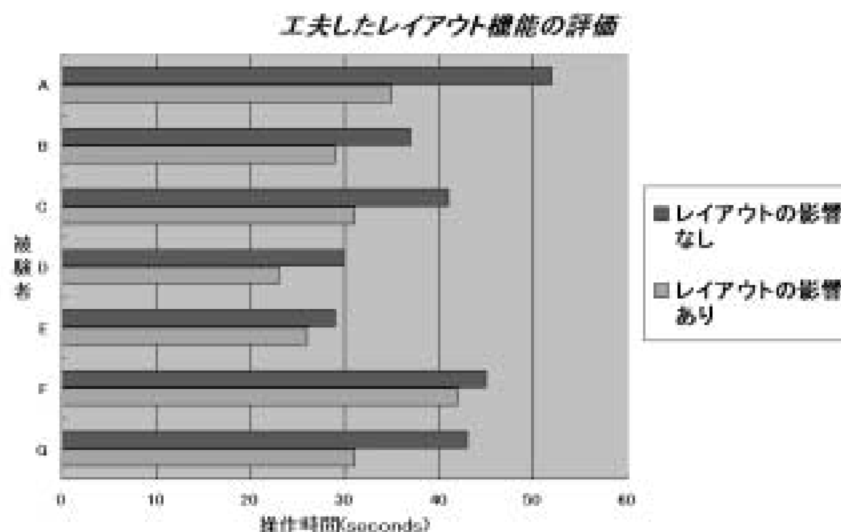

図 4.5: 工夫したレイアウト機能の実験結果

工夫したノード移動機能の実験結果を 4.6に示す。ユーザは、工夫したノード移動機能を使うことによって、サブグラフをより短い時間で移動することができる。

4.1.2 実験結果に関する考察

4.3の実験結果から、拡張直接操作手法と三次元スプリング・モデルを統合したシステムの方が統合していないシステムよりも三次元グラフを容易に操作できていることがわかる。各システムの測定時間の平均については統合したシステムで 76.1 秒、統合していないシステムでは、 101.8 秒である。この結果から平均時間で約 34 %の操作時間が短縮されていることがわかる。しかし、図 4.3から被験者によって計測時間の差にばらつきが生じていることがわかる。操作に慣れている被験者 A,B,C,E では、平均時間で約 16 %の操作時間が短縮されている。操作に慣れていない被験者 D、 F、 G では、平均時間で約 51 %の操作時間が短縮されている。操作に慣れていない者の方が大きく時間が短縮されている。これは、本システムが操作に特に慣れていない者に対して有効に働いていることを示している。

また、工夫したレイアウト機能の実験結果より機能が稼働しているシステムでは平均操作時間が 31.0 秒、機能が稼働してない 39.5 秒である。この結果から平均操作時間が約 27 %短縮されていることがわかる。

工夫したノード移動機能の実験結果から機能が稼働したシステムでは平均操作時間が 5.0 秒、機能が稼働していないシステムでは 8.4 秒である。この結果から平均操作

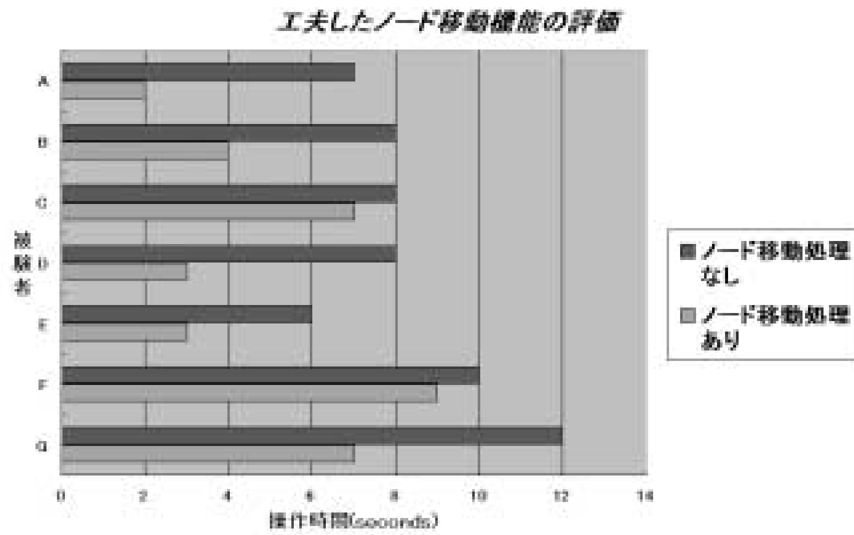

図 4.6: 工夫したノードの移動機能の実験結果

時間で約 68.5 %の時間が短縮されている。

工夫したレイアウト機能と工夫したノード移動機能のどちらも単独で使用しても有効に働いていることがわかる。これらの機能は、統合されたシステムでも有効に働き操作時間の短縮に貢献していると言える。

第 5 章

三次元グラフの効率的な入力法

我々は第 3章で拡張直接操作手法と三次元スプリング・モデルを統合した手法を提案し三次元ビジュアルプログラムの編集環境の構築にあたった。これまで我々が 3D-PP で考えていた入力方法はノードをエッジでつなぐという単純なステップの繰り返しから成り立ち、全ての入力の場面において必ずしも効率的であるとは言えない。我々は、三次元化によるメリットを享受しつつ効率的に図形を入力するには操作に工夫が必要であると考えた。本章では、より直感的で効率的な入力法について述べる。

5.1　効率的な入力

コンピュータにはキーボードなどの入力装置がある。しかし、ユーザが容易に二次元物体を操作するためには、直接操作手法 [24] が適用されたユーザインタフェースが非常に有効である。直接操作手法は二次元の操作系に親和性があり、ポインティングデバイスが有効な入力装置として適用されている。本節では、ポインティングデバイスで用いられるクリックとドラッグ＆ドロップをどのように操作すれば効率的に入力することができるかという点について述べる。

クリックは直接マウスカーソルでスクリーンに表示された点や面を選択する操作であり、ドラッグ＆ドロップ [19] はさらに連続的な移動や範囲指定をする操作である。ドラッグ＆ドロップは連続的に操作を必要とする場面、例えばエッジを引くなどの操作に有効である。クリックは連続的に操作を必要としない場面、例えば、ノードの生成という操作には有効である。クリックをエッジの生成に用いた場合、一つのエッジに対して始点と終点をクリックする必要がある。クリック数はエッジの数の二倍の回数で入力することになりドラッグ＆ドロップと比べて効率的な操作とは言えない。

グラフで表現されたビジュアルプログラムに用いられているノードとエッジは表裏

一体の関係にある。ノードを入力すれば、エッジが必要になり、逆の場合もある。しかし、グラフを次々と作成する場面でクリックとドラッグ＆ドロップでノードとエッジを別々に入力するのは必ずしも効率のいい方法とは言えない。

我々は、クリックする操作をドラッグ＆ドロップでも同時に操作できるようにすれば、クリックする操作量が減りこれまでの入力法に比べて効率的に入力できると考えた。

5.2　これまでの入力法

これまで我々が適用していた入力法では、まずユーザは３Ｄアイコンと呼ぶ小さな基本図形をクリックすることによってノードを生成する。ノードの表面にある引数をドラッグし、別の引数へドロップすることによってエッジを生成し、結線することができる。この方法では、ノードの数だけクリックが必要となり、エッジの数だけドラッグ＆ドロップが必要となり、ユーザが実用レベルの VPS を記述する際入力が煩雑になってしまう。

そこで、より少いクリック数と少いドラッグ＆ドロップ数でノードを生成し結線する手法について考えた。

5.3　入力操作の問題点

5.3.1　入力を妨げるレイアウト

拡張直接操作手法と三次元スプリング・モデルの統合によって操作と同時に自動レイアウトが行われる。しかし、本手法はかえって効率的な入力を妨げる場合がある。例えば、同じサブグラフ内のノードに対して結線したい場合隣接していないノード同士であれば自動レイアウトによってお互いに斥力が働く。お互いに斥力によって離れてしまうため、ユーザが結線することを難しくしてしまう。図 5.1で、ユーザはノード A とノード B を結線しようとしている。しかし、自動レイアウトによって矢印の方向に斥力がノード A、ノード B に働く。二つのノード位置は斥力によって離れて結線を難しくする。

5.3.2　マウスによる三次元移動の難しさ

三次元上で、オブジェクトを任意の位置までドラッグしていくのが難しい。これは、二次元デバイスであるマウスを使い三次元移動を行うために起こる問題である。マウ

スでは、一度の操作では二次元でしか移動できない。そのため、複数回の別平面での操作を行わなければならないのである。図 5.2では、 1 → 3 へ移動できるのが理想だが、実際には、地面と平行な平面上を 1 → 2 と移動し、次に地面と垂直な方向へ 2 → 3 という 2 段階の移動をすることになる。 1、 2、 3 を含む平面上を移動すれば 1 度の移動で済むが、この場合、この平面を指定することは、マウスによる直交する 2 平面を移動させることよりも難しいことが多い。

5.3.3　操作の繁雑さ

通常の三次元ＣＧでは、二次元の平面であるコンピュータのディスプレイに投影表示を行っている。マウスは二次元入力デバイスである。入出力装置が二次元であるのに対し、操作の対象となるノードやエッジが三次元で表示される。ユーザの視点から手前にあるオブジェクトによって奥にあるオブジェクトが隠れてしまうことがあるので頻繁に視点移動の操作が必要となり、視点移動の操作の分だけ操作が繁雑になる。

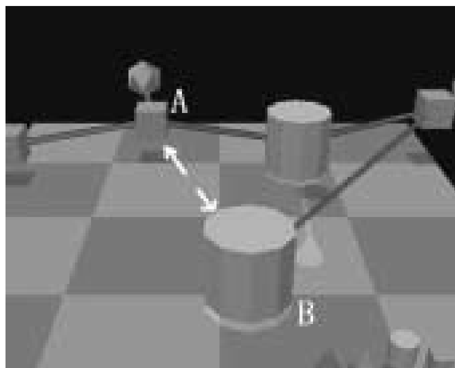

図 5.1:　入力を妨げるレイアウト

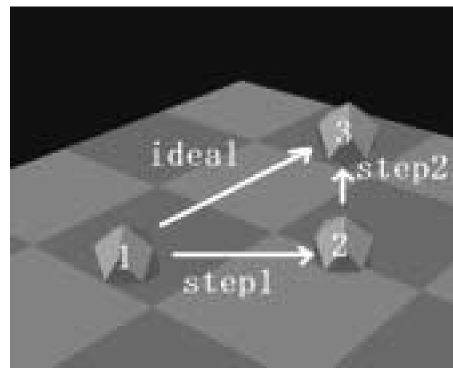

図 5.2:　操作の難しさ

5.4　入力操作の改善

5.4.1　レイアウトの一時停止操作

マウスカーソルがノードと重なっている時は、ユーザはノードを操作していることが多い。また、マウスカーソルがノードと重なっていない時は、ユーザが視点の移動操作、エッジ生成のためにドラッグをしていることが多い。

我々は、エッジの生成中にノードの自動的レイアウトを行わない方が入力操作を容易にすると考えた。そこで我々は、マウスカーソルが地面や空間にある場合は自動レ

イアウトを一時停止するようにした。例えば、ユーザが、ノード同士を結線する時二つのどちらか一方をドラッグする。ユーザはもう一方のノードを指定するためにマウスカーソルを最初のノードから離し移動する。マウスカーソルがもう一方のノードに重なるまでの間、自動レイアウトを一時停止する。一時停止するので、ユーザがエッジで結線することを容易にする。

5.4.2 操作平面を用いた三次元空間の移動

オブジェクト移動時、操作平面が視線と平行に近いような状況では、どのくらい移動したか分りづらいことがある。また、操作中にオブジェクトが視界外や描画範囲外に簡単に出てしまうこともあり、操作が困難になってしまい操作対象のオブジェクトを見失いやすい。

オブジェクト移動時の操作平面として、視線に垂直な仮想的な平面を導入した。ドラッグ操作をこの平面上で行う場合、オブジェクトの奥行き方向の移動はなくなる。このため、ユーザの視点からは、一般的な二次元ドラッグ操作と同様の操作感で、容易にオブジェクトの移動を行うことができる。この移動方法では、オブェクトが視界から消えることもなく一操作で画面全体に移動することができる。

5.4.3 少ないクリック数と少ないドラッグ＆ドロップ数

少いクリック数と少いドラッグ＆ドロップ数で入力することによってユーザは効率的に入力することができる。今回は特に効率的な入力法が重要と考え、最も有効であると考えられる手法を提案する。

5.5 新入力法

新入力方法を「複数の操作を一度にまとめた操作型」と「再利用型」という２つのアプローチを元に提案する。

5.5.1 複数の操作を一度にまとめた操作型

従来、我々は複数の命令を一命令に置換えたマクロ命令を用いることがある。この手法から VPS でノードの生成やエッジの生成を一度の操作で入力する手法を考え、以下の４つの手法にまとめた。

細胞分裂型入力法

我々は、細胞が分裂することによって数を増やす、という方法をノードとエッジの生成に適用することを考えた。図 5.3では細胞分裂型入力法を用いてノードとエッジの生成を行っている。最初にユーザはノード A の右端を摘む。マウスカーソルをそのまま右へ移動する。ノード A はくびれながら伸びる。くびれの一部はそのままエッジになる。ノード A から細胞分裂するように同じ種類のノード B がエッジの右端に生成される。マウスボタンをドロップすると一連の入力操作は一旦完了する。ユーザは、同じ操作を繰り返すことによって連続的にノードとエッジを生成することができる。

この入力法では同じ種類のノードの生成と同時にエッジの生成も行われる。ドラッグ＆ドロップ数はノードの数だけ必要になるがクリック数は従来より少く済むがという点で従来の入力方法よりも効率的であると言える。

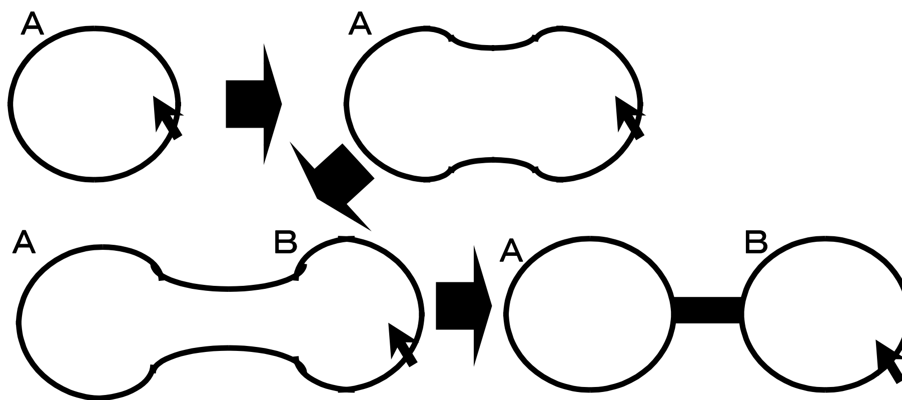

図 5.3: 細胞分裂型入力法

一筆書き型入力法

一筆書きとは、紙面から筆を一度も離さないでなぞり終える書き方である。我々は、一筆書き用いてをエッジを効率的に入力する一筆書き型入力法を考えた。図 5.4で一筆書き型入力法を用いたエッジが効率的な入力法について説明する。図 5.4左ではユーザは既にノード A、ノード B、ノード C を生成している。ユーザはノード A とノード B をつなげ、さらにノード B とノード C もつなげたい。ユーザはマウスカーソルをドラッグしてノード A からノード B、ノード B からノード C へとなぞる。図 5.4真中ではなぞったマウスカーソルの軌跡が実線で示されている。ユーザがなぞり終わり、ドロップするとシステムはマウスカーソルの軌跡を元にノード A、ノード B 間とノード

B、ノード C 間にエッジを生成する。図 5.4右ではノード A、ノード B、ノード C が結線されている。ユーザが１回ドラッグ＆ドロップするだけで２つのエッジを生成することができる。

ユーザは、ノードを生成する時にノードの数だけクリックしなければならない。この点は従来の入力法と変わらない。しかし、エッジの結線はドラッグしたマウスの軌跡と交わったノードに対して行われる。ドラッグ＆ドロップの数は、ノードの数よりも少いという点で従来の方法と比較して効率的である。また、二次元ペイントツールのように手軽に結線できるというメリットもある。

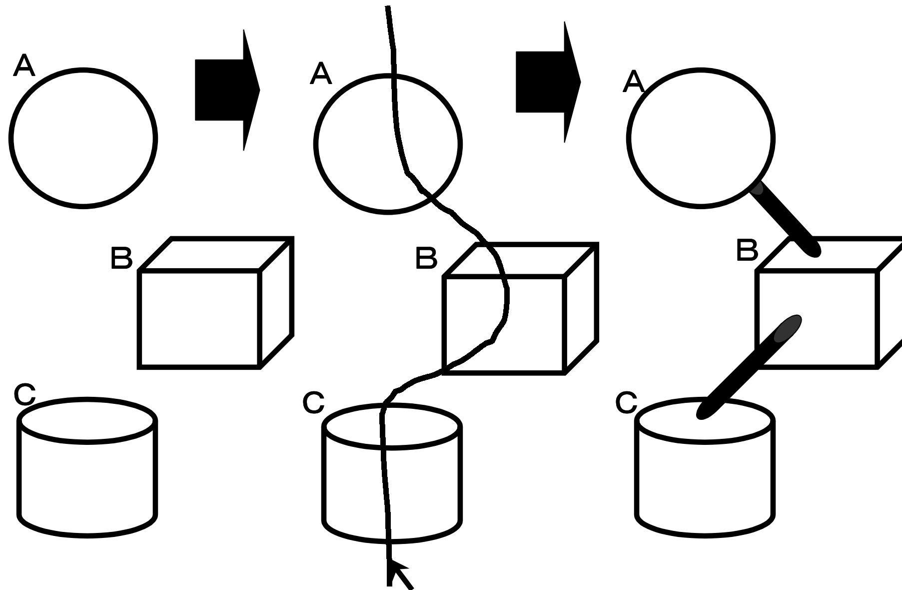

図 5.4: 一筆書き型入力法

重ね合わせ型入力法

我々は、拡張直接操作手法の中で拡張ドラッグ＆ドロップ手法を適用している。拡張ドラッグ＆ドロップではユーザの視線から見たオブジェクトの重ね合わせを利用している。我々はユーザの視線から見たオブジェクトの重ね合わせを利用したエッジの効率的な入力法として適用することを考えた。

図 5.5では、重ね合わせ型入力法を用いてエッジを効率的に生成している。ユーザの視線からノード A、ノード B が重なっていないように配置している。ユーザは手前に配置してあるノード A をドラッグして右に移動し、奥にあるノード B に重ねる。システムはユーザの視線から見てノード A とノード B が重なっていると検知する。ノー

ドＡとノードＢの間にはエッジが生成される。ユーザは、ノードＢ以外でも重ねるだけで連続的にエッジを生成することができる。

この方法は従来の入力法と比べてドラッグ＆ドロップ数はノードの数だけで済む。また、重ねれば結線してしまうので奥行を気にせず気軽に入力できるというメリットもある。

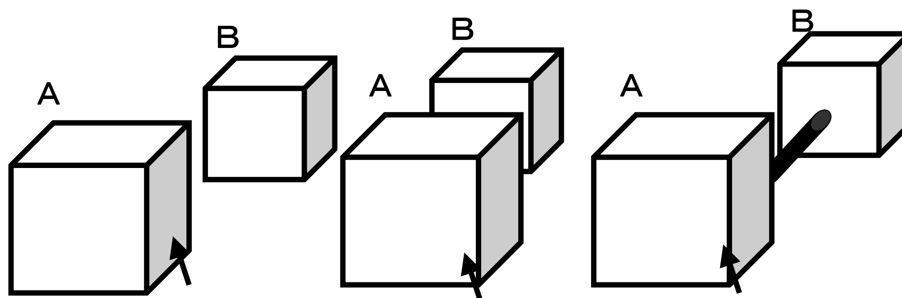

図 5.5: 重ね合わせ型入力法

新一筆書き型入力法

早野が dish シェル [29] 上で複数のアイコンを一度に操作する方法として「串刺し」を提案した。つまんだアイコンの列を一度に操作したいアイコンの上に重ると摘んだアイコンの列に追加される。我々はこの手法をプログラムの入力法に適用することを考えた。この手法を一筆書き型入力法に適用することでノードとエッジの生成を同時に行うことができる。

図 5.6で新一筆書き型入力法を用いた効率的なノードとエッジの生成について説明する。ユーザは最初に、生成したいノードと同じ種類の３Ｄアイコンをドラッグして生成する。図 5.6左でユーザは３Ｄアイコン A、 B、 C を順番にドラッグしている。３Ｄアイコン A からノード A'、３Ｄアイコン B からノード B' がエッジで結線された状態で生成される。ノード B' とエッジで結線された状態でノード C' が３Ｄアイコン C から生成される。ユーザがドロップすることで入力処理が終了する。ユーザは１回のドラッグ＆ドロップでノードを３つ、エッジを２つ生成することができる。

ユーザが次々とマウスカーソルとノードを重ね合わせることによってグラフが生成される。また、ノードの「移動」操作とノードやエッジの「生成」操作を区別せずに操作できるのでユーザは「移動」と「生成」の操作を切り替る必要がなくなる。

一筆書き型入力法では、ノードの生成する操作は従来と同じクリックによる操作で

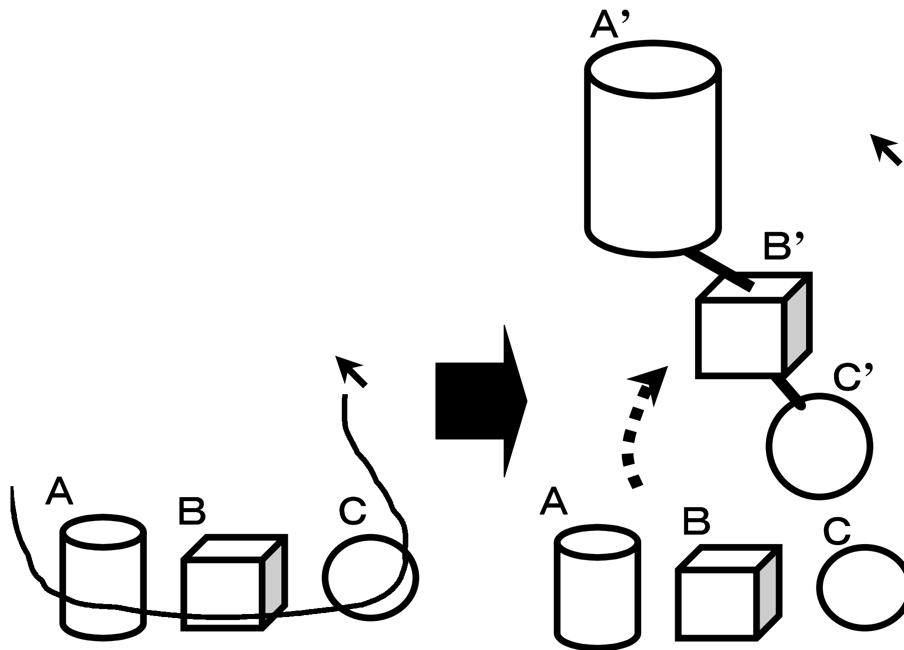

図 5.6: 新一筆書き型入力法

あったため効率的な入力法としては十分ではなかった。新一筆書き型入力法では、３Ｄアイコンを利用しノードをエッジの生成を同時に行なうのでドラッグ＆ドロップ数だけでなく少ないクリック数での入力が可能になった。この手法は、３Ｄアイコンからノードを生成する場面で有効な入力法である。

5.5.2　再利用型

一般的に定型の文章を頻繁に書く時、事前に用意されたテンプレートに文章を編集することで効率的に文章を書くことができる手法がある。VPS において今まで使用していたプログラムや頻繁に記述するようなプログラムを部品として再利用し、入力の手間を省く手法について以下の２つの手法にまとめた。

セミオーダー型入力法

ユーザがよく使うようなグラフのひな型を幾つか用意する。そのようなひな型を３Ｄアイコンとして表示する。ユーザが同じような構造を持つビジュアルプログラムを繰り返し入力する場面でひな型の３Ｄアイコンをクリック、生成する。ユーザは、ひな型を利用することによって最初からプログラミングする必要がなくなり効率的にプログラミングができる。

図 5.7を用いてセミオーダー型入力法について説明する。よく使うグラフをひな型として３Ｄアイコン C として用意する。ユーザは３Ｄアイコン A からノード A'、３Ｄアイコン C から２つのサブグラフ C' を生成する。ユーザは、ノード A' とサブグラフ C' を少し編集するだけでビジュアルプログラム D' を作成することができる。

この入力法は、予めよく使うプログラムが分かり、３Ｄアイコンとして用意されている場合に有効である。

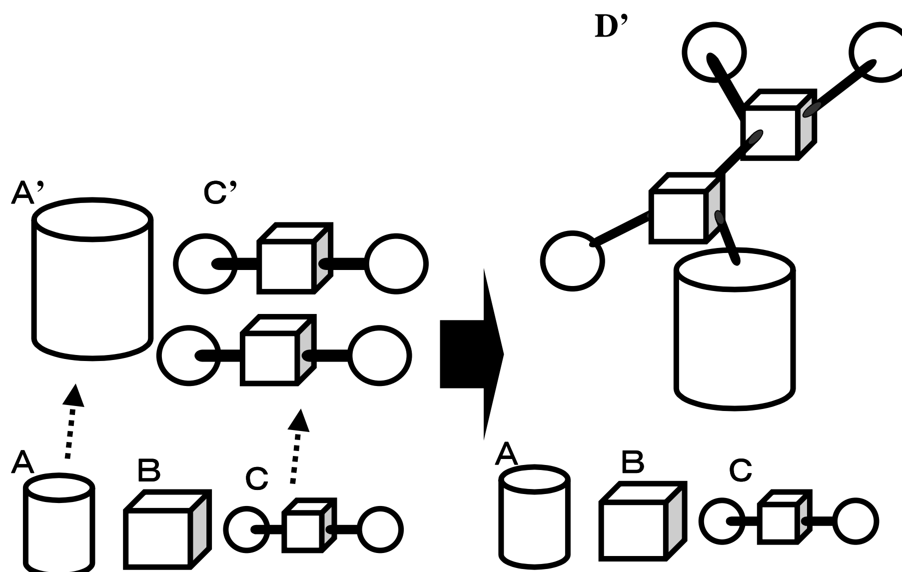

図 5.7: セミオーダー型入力法

テキスト利用型入力法

今までテキストで書いたプログラムをコピーしエディタの画面にペーストすることによってビジュアルプログラムをシステム側で自動生成する。プログラムの任意の部分を利用できるのでプログラムを全てロードして不要なビジュアルプログラムを生成しなくて済む。

図 5.8では、テキスト利用型入力法を用いてビジュアルプログラムを入力法している。ユーザは、これから入力するテキストで書かれたプログラムをコピーする。はじめ、図 5.8左では画面にはビジュアルプログラムが表示されていない。しかし、図 5.8右ではペーストされたテキストプログラムがビジュアルプログラムに変換され表示されている。既存のプログラムを利用しているのでゼロからプログラミングする必要がなくなり入力の手間が省ける。

この入力法は、予めテキストで書いたプログラムがある場合に有効である。

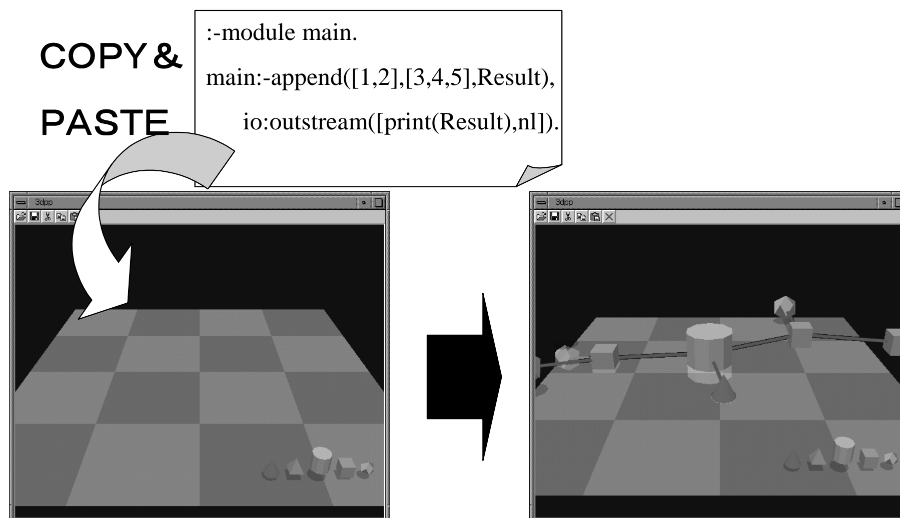

図 5.8: テキスト利用型入力法

5.6　各入力法のクリック数とドラッグ＆ドロップ数

我々は、各入力法のクリック数とドラッグ＆ドロップ数に注目しどの入力法がより効率的であるかについて述べる。

細胞分裂型入力法

ユーザは、この入力法によってノードの生成とエッジの生成を繰り返し行う。しかし、最初のノードの位置が自動レイアウトによって移動してしまうとユーザが引張ったノードの位置に最初のノードがついてきてしまう。ユーザは最初のノードの位置は固定する必要がある。連続的に入力する毎に固定操作しなければならない。固定操作が増えてしまうため、クリック数やドラッグ＆ドロップ数が減らない場合がある。例えば、ユーザはクリックによってノードの位置を固定できると仮定する。二つのノードを結線する操作において、従来の入力法ではノード生成でクリック数２回、エッジの生成でドラッグ＆ドロップ数１回である。細胞分裂型入力法では、ノードの生成と位置固定で**クリック数２回**、**ドラッグ＆ドロップ数１回**を必要とする。この場合効率的に入力しているとは言えない。

また、ユーザが異なった種類のノードを結線する場合、従来の入力法でノードの生成と結線を行うので、効率的にプログラムを入力することができない。細胞分裂型入力法は、同じ種類のノードを連続的に結線する場面で有効な入力法である。

一筆書き型入力法

二つのノードを生成し結線する場面では従来の入力法、一筆書き型入力法のどちらもクリック数とドラッグ＆ドロップ数は同じである。ノードの生成に**クリック数２回**、エッジの生成に**ドラッグ＆ドロップ数が１回**必要である。一筆書き型入力法のノードの生成については、従来の入力法と同じである。プログラムを入力する場面ではノードを生成することが多い。効率的に入力をするには、ノードの生成時にクリックする数を減らす必要がある。

重ね合わせ型入力法

重ね合わせ型入力法についても一つの場面でクリック数やドラッグ＆ドロップ数が減らないことがある。二つのノードを生成し一つのエッジを生成する場面では従来の入力法と重ね合わせ型入力法のどちらもクリック数とドラッグ＆ドロップ数は同じである。ノードの生成に**クリック数２回**、エッジの生成に**ドラッグ＆ドロップ数が１回**必要である。重ね合わせ型入力法についてもノードを生成する方法は従来の方法と同じである。連続して結線するノードが少ない場面では効率的であるとは言えない。

新一筆書き型入力法

新一筆書き型入力法では、ノードの生成とエッジの生成を３Ｄアイコンを用いて行う。連続して結線するノードが少ない場面でもクリック数やドラッグ＆ドロップ数が減る。二つのノードを生成し一つのエッジを生成する場面において従来の入力法では、ノードの生成にクリック数２回、エッジの生成にドラッグ＆ドロップ数が１回必要である。新一筆書き型入力法では、**ドラッグ＆ドロップ数が１回**で済む。

セミオーダー型入力法

ひな型として用意されたノードは生成するには**クリック数が１回**必要である。元になるプログラム (ひな型) そのままでは、記述が充分ではない。更にプログラムを入力し充分な記述にする必要がある。ユーザは効率のよい入力法を組み合わせることで操作を行う必要がある。組み合わせた入力法によってプログラム作成に必要なクリック数やドラッグ＆ドロップ数を大きく減らすことができる。

テキスト利用型入力法

通常、マウスカーソルを使ってテキストをコピー＆ペーストする時テキストをコピーしたい範囲でドラッグしペーストしたい場所をクリックする場合が多い。テキスト利用型入力法でも同じようにコピー＆ペーストすると**クリック数１回**、**ドラッグ＆ドロップ数１回**が必要である (但し、コピーの範囲指定でのドラッグ１回はドラッグ＆ドロップ数１回として数えた)。テキストからの入力を元にプログラムの内容を更に編集する場合があるので効率のよい入力法と組み合わせる必要がある。セミオーダー型入力法

と同様に、組み合わせた入力法によってプログラム作成に必要なクリック数やドラッグ＆ドロップ数を大きく減らすことができる。

N 個のノードと N−１個のエッジを入力する場合、各入力法で必要なクリック数とドラッグ＆ドロップ数を比較表 5.6にまとめた。

表 5.6：各入力法の比較

	クリック数	ドラッグ＆ドロップ数
従来の入力法	$\boldsymbol{N}$	$\boldsymbol{N-1}$
細胞分裂型入力法	**1**	$\boldsymbol{N-1}$
一筆書き型入力法	$\boldsymbol{N}$	**1**
重ね合わせ型入力法	$\boldsymbol{N}$	**1**
新一筆書き型入力法	**0**	**1**
セミオーダー型入力法	**1**	**0**
テキスト利用型入力法	**1**	**1**

5.7 複数の入力法を組み合わせた効率的な入力法

入力の場面毎に優れた入力法がある。ノードとエッジを繰り返し生成する場面において効率のよい入力方法としてはクリック数とドラッグ＆ドロップ数の一番少ない**新一筆書き型入力法**が効率的である。構造の似ているビジュアルプログラムを繰り返し作成するような場面では**セミオーダー型入力法**が効率的である。

また、既存のテキストで記述されたプログラムを利用してプログラミングする場面では、 **テキスト利用型入力法**で効率的な入力をすることができる。

そこで、新一筆書き型入力法とセミオーダー型入力法とテキスト利用型入力法を組み合せた効率的な入力法を提案する。

5.8 新一筆書き型入力法、セミオーダー型入力法とテキスト利用型入力法を組み合せた作成例

新一筆書き型入力法、セミオーダー型入力法とテキスト利用型入力法を組み合せてノードとエッジが作成されていく様子を dispatch(図 5.9) というプログラムの作成例を用いて説明する。 dispatch とは、ある数列を入力すると奇数と偶数の２つの数列を出力するプログラムである。

操作１：ノードの生成

３Ｄアイコンを用いることによってノードを作成する。(図 5.10) ここでは、円錐の形をした３Ｄアイコンをドラッグすることによって３Ｄアイコンと同じ位置に少し大きな球 (ノード) が生成される。この段階では、エッジによる結線はされていない。

新一筆書き型入力法

操作２：エッジとノードの同時生成。

ドラッグ中のマウスカーソルをエッジで繋ぎたい種類のノードへ重ねる。新しいノードがノードをエッジがつながれた状態で生成される。(図 5.11) 最後に結線したいノードでドロップをすると、ノードの作成作業とエッジの作成作業は一旦完了となる。

新一筆書き型入力法

操作３：ひな型ノードを使ったビジュアルプログラムの作成

ノードを作成する要領でひな型ノードを作成、エッジで結線する。ノードをダブルクリックによって拡大、内部を適当なプログラムに編集する。(図 5.12)
セミオーダー型入力法

操作４：テキストで記述された既存のプログラムを用いたビジュアルプログラムの作成

拡大されたノードの内部に既存のプログラムの好きな部分をコピーして、画面にペーストする。するとシステム側でピジュアルプログラムを自動的に生成する。内容を編集して目的のプログラムを作成する。

(図 5.13) テキスト利用型入力法

操作１、操作２、操作３、操作４を繰り返すことで、ビジュアルプログラムを作成する。

```
:-module main.
main :- dispatch:dispatch([], Odd, Even),
        io:outstream([print(Odd),nl,print(Even),nl]).
:-module dispatch .
dispatch([],Odd,Even):- Odd=[],Even=[].
dispatch([One|Rest],Odd,Even):- One mod 2=\=0 |
        Odd=[One|OddTail],dispatch(Rest,OddTail,Even).
dispatch([One|Rest],Odd,Even):- One mod 2=:=0 |
        Even=[One|EvenTail],dispatch(Rest,Odd,EvenTail).
```

図 5.9: プログラム

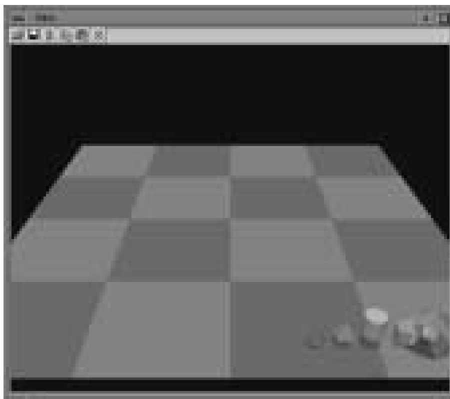

図 5.10: 新一筆書き型入力法１

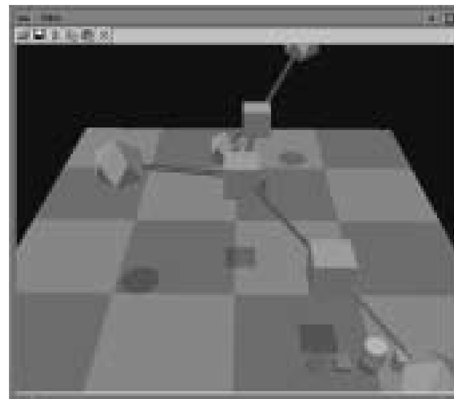

図 5.11: 新一筆書き型入力法２

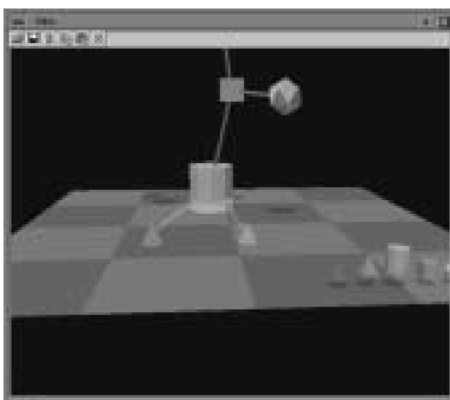

図 5.12: セミオーダー型入力法

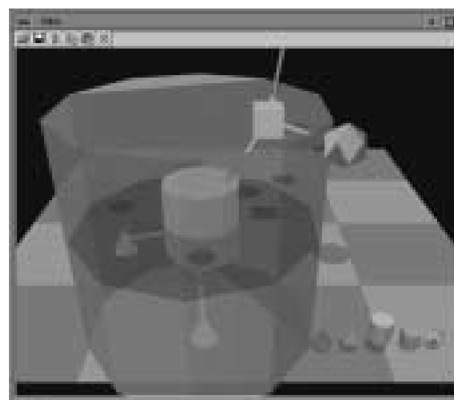

図 5.13: テキスト利用型入力法

第 6 章

関連研究

6.1　三次元グラフに関する研究

篠沢らは WWW 空間を三次元グラフィックを用いて視覚化し、ノードで表現されたページを「持ち上げ」操作によってページ間を結ぶエッジを辿り、情報調べることのできる「Natto View」 [30] を提案、開発した。ユーザは詳細な情報を調べるために「持ち上げ」操作をすることがある。視覚化された情報はレイアウトに依存しない三次元グラフであるので可読性を充分保持しているわけではない。また、「Natto View」は情報検索をするシステムであり、編集をすることはない。本研究では、システムは自動レイアウトによって三次元グラフの可読性を一定に保ち、同時に三次元グラフの編集をすることが出来るという点で異る。

舘村は、二次元空間にスプリングモデルによって配置された文献集合からドラッグなどの操作によって情報獲得を支援する「DocSpace」 [31] を提案している。ユーザがのノードを操作するとエッジで繋がれた他のノードも操作に追従して移動、レイアウトされるという点は本手法と似ているが、編集することはない。

大澤ら [32] は並列計算機や並列プログラムを三次元グラフで表現し、実行状態や統計量を力学系モデルを用いて配置し計算と通信のバランスをアニメーション表示する方法を提案している。三次元グラフを用いて、力学系モデルでレイアウトするという点では本手法と似ているが、同時に直接操作を行わないという点で異なる。

6.2　効率的な図形の入力法に関する研究

newPP[18] では二次元空間でビジュアルプログラムを作成する。ユーザが効率的に図形を入力する手法を試験的に導入した。ノードの生成と同時にエッジを生成するこ

とによってクリック数を減らし効率的な入力を実現している点は、我々のアプローチと似ている。我々は三次元空間でビジュアルプログラムを効率的に入力することを目標にしているという点で異なる。

安福ら [16] は曲線や直線をドローツールとして利用した効率的な入力法を提案している。一から三次元図形を効率的に入力するという点で有効性が示されている。我々は、用意された三次元図形を用いてビジュアルプログラムを作成する。三次元図形を生成し組み合わせることで効率的プログラムを作成する点で異なる。

第 7 章

まとめ

我々は、グラフを直接操作手法を用いて作成し、同時に三次元バネ・モデルを用いて自動レイアウトする手法を提案し、実装した。本システムの実装によって、グラフは作成中であっても直接操作手法と可読性の維持の両方を実現することができた。我々は本システムを三次元 VPS **“3D-PP”**へ適用する予定である。また、本システムを使った効率的な入力法を幾つか組み合わせた入力法を提案し作成例を用いて有効性を示すことができた。

謝辞

本研究を進めるにあたり終始親切に指導してくださいました田中二郎教授に心より感謝いたします。また田中研究室の皆さんからはゼミやディスカッション等から貴重な意見やコメントを頂きました。特に３Ｄ－ＰＰの研究グループのメンバーである、大芝崇さん、小川徹さん、中須正人さん、永田丈士さん、劉学軍さんとは本研究に対する有益なコメントを頂きました。システムの実装にあたりサポートして頂いた飯塚和久さん、神谷誠さんに感謝します。

参考文献

[1] K.M. Kahn, V.A. Saraswat: Complete visualizations of concurrent programs and their execution.In Proc. of 1990 IEEE Workshop on Vsual Languages, pp.7-15, 1990.

[2] Ken Kahn: ToonTalk(TM)-An Animated Programming Enveronment for Children, Journal of Visual Languages and Computing, June, 1996.

[3] 高橋 哲, 小池 英樹: 並列言語 Linda のプログラムの実行状態の視覚化, インタラクティブシステムとソフトウェア I: 日本ソフトウェア科学会 WISS'93, pp.9-16, 1993.

[4] 志築 文太郎, 豊田 正史, 高橋 伸, 柴山 悦哉: ビジュアル並列プログラミング言語 KLIEG: プロセスネットワークパターンを利用した再利用性の向上と実行表示の効率化, インタラクティブシステムとソフトウェア IV: 日本ソフトウェア科学会 WISS'96, pp.81-90, 1996.

[5] Christian Geiger, Wolfgang Mueller and Waldemar Rosenbach: SAM-An Animated 3D Programming Language, Proc.1998 IEEE Symposium on Visual Languages, pp.228-235, 1998.

[6] 田中二郎: 並列論理型言語ＧＨＣのビジュアル化の試み, インタラクティブシステムとソフトウェア I: 日本ソフトウェア科学会 WISS'93, pp.265-272, 1993.

[7] J. Tanaka: Visual Programming System for Parallel Logic Languages, Proceedings of the IASTED International Conference Parallel and Distributed Computer and Network(PDCN'97), pp.188-193, 1997.

[8] 宮城幸司, 大芝崇, 田中二郎: 三次元ビジュアル・プログラミング・システム 3D-PP, 日本ソフトウェア科学会第 15 回大会論文集, pp.125-128, 1998.

[9] T.Oshiba, J.Tanaka: “3D-PP”: Visual Programming System with Three-Dimensional Representation, Proceeding of International Symposium on Future Software Technology (ISFST ’99), Nanjing, China, October 27th to 29th, pp.61-66, 1999.

[10] T.Oshiba, J.Tanaka: “3D-PP”: Three-Dimensional Visual Programming System, Proceeding of 1999 IEEE Symposium on Visual Languages (VL ’99), Tokyo, Japan, 13th to 16th, pp.189-190, September, 1999.

[11] Michael Chen, S. Joy Mountford and Abigail Shellen: A Study in Interactive 3-D Rotation Using 2-D Control Devices,In ACM SIGRAPH Computer Graphics, pp.121-129, 1998.

[12] K. P. Herndon, R. C. Zeleznik, D. C. Robbins, D. B. Conner, S. S. Snibbe, and van Dam:Interactive Shadows, Proceedings of the ACM Symposium on User Interface Software and Technology, pp.1-6, 1992.

[13] Jackie Neider, Tom Davis, Mason Woo: OpenGL Programming Guide, 1995.

[14] 千葉則茂, 土井章男: 3次元 CG の基礎と応用, 1997.

[15] 暦本純一: InformationCube: 半透明表示を用いた3次元情報視覚化技法, インタラクティブシステムとソフトウェア I: 日本ソフトウェア科学会 WISS’93,pp.1-8.

[16] 安福 尚文, 佐賀 聡人: 空間描画動作同定に基づく立体プリミティブ入力インタフェース, インタラクション’99 論文集, pp.119-126, 1999.

[17] 南雲淳, 田中二郎: “viewPP”: グラフ構造とアニメーション表現に基づくプログラム実行の視覚化、日本ソフトウェア科学会第14回大会論文集, pp.17-20, 1997.

[18] 田中二郎, 後藤和貴, 馬場昭宏: ビジュアルプログラミングシステムにおける入力法の効率化, 日本ソフトウェア科学会第 12 会大会論文集,pp165-168,1995.

[19] Annette Wagner, Patrick Curran, Robert O’Brien: Drag Me,Drop Me,Treat Me Like an Object, Proceedings of The ACM Conference on Human Factors in Computing Systems(CHI’95), pp.525-530, 1995.

[20] D. A. Henderson and S. K. Card: Rooms:the use of multiple virtual workspace to reduce space contention in a window-based graphical user interface,ACM Transaction on Graphics, Vol.5, No3, pp211-243, july 1986.

[21] Walker II, J. Q: A Node-Posotioning Algorithm for General Tree,Software-Practice and Experience, 20-7, pp.685-705, 1990.

[22] Sugiyama K., S. Tagawa and M. Toda: Method for Visual Understanding of Hierarchical System Structures, IEEE Trans. on System, Man, and Cybernetics, SMC-11-2, pp.109-125, 1981.

[23] Batini C., M. Talamo and R. Tamassia: Computer Aided Layout of Entity-Relationship Diagrams, J. Systems and Software, pp.163-173, 1984.

[24] Ben Shneiderman: Direct Manipulation: A Step Beyond Programming Languages, IEEE Computer, vol.16, No8, pp.57-69, 1983.

[25] H. Mitsunobu, T. Oshiba and J. Tanaka: Claymore: Augmented Direct Manipulation of Three-Dimensional Objects, Proceedings of Asia Pacific Computer Human Interaction 1998 (APCHI'98), IEEE Computer Society Press, pp.210-216, July, 1998.

[26] 神谷 誠: 3次元ビジュアルプログラミングシステムにおける ドラッグ&ドロップ手法の拡張, 筑波大学システム工学学類卒業論文,1998.

[27] 杉山公造: グラフ自動描画法とその応用, 計測自動制御学会, 1993.

[28] Peter Eades: A heuristic for graph drawing, congressus numerantium, vol 42, pp.149-160, 1984.

[29] 早野浩生: dish: ドラッグ&ドロップでスクリプトの記述が可能なシェル, 日本ソフトウェア科学会第15回大会論文集, pp.169-172, 1998.

[30] H. Shinozawa, Y. Matsushita: WWW visualization giving meanings to interactive manipulations, HCI International'97, Augst, 1997.

[31] 舘村純一: DocSpace: 文献空間のインタラクティブ視覚化, インタラクティブシステムとソフトウェア IV: 日本ソフトウェア科学会 WISS'96, pp.11-19, 1996.

[32] 大澤範高，弓場敏嗣: 力学系モデルを利用した並列計算機動作状態のアニメーション，インタラクティブシステムとソフトウェア IV: 日本ソフトウェア科学会 WISS'96, pp.189-197, 1996.

付録 A

三次元グラフエディタの詳細

A.1　三次元グラフエディタの作成

我々は三次元グラフエディタを作成するにあたり、幾つかの工夫をしている。

A.1.1　ノードの生成と配置の工夫

我々の研究室では、三次元モデリングツール **Claymore** を開発している。**Claymore** は、手軽に三次元モデルを作成するシステムである。**Claymore** ではプリミティブ (基本立体) を使うことによって三次元モデルを手軽に作成することが可能である。我々は、同様にプリミティブを用いて三次元グラフを容易に作成できると考えた。

Claymore では、ユーザが３Ｄアイコンをクリックすることで立体モデルを生成している。しかし、３Ｄアイコンをクリックすると地面の中央に配置される。ユーザが立体モデルを任意の位置に配置したい場合、ユーザは立体モデルを地面中央の位置から再配置しなければならない。ほとんどの立体モデルは地面中央以外の位置に配置されることが多い。立体の生成と配置の位置が離れているので３Ｄアイコンの位置と地面中央との往復が多いと操作が繁雑になりやすい。三次元グラフでは、自動レイアウト処理を行うので初期配置が重要になる。三次元グラフエディタでは３Ｄアイコンとほぼ同じ位置にノードを配置し、３Ｄアイコンの位置と地面中央との往復を減らす工夫を行っている。

A.1.2　グラフのグループ化の工夫

三次元グラフエディタでは、各々のサブグラフが互いに影響を与えずに自動レイアウトする事を可能にしている。システムでは一つのサブグラフを一つのグループとみなして自動レイアウトを行っている。我々は、サブグラフをグループ化するにあたり

グループ ID という情報を用いている。ノードやエッジは順番に ID が割り当てられている。サブグラフ内にあるノードの ID で一番大きい ID をグループ ID として採用し、同じサブグラフ内の全てのノードが持っているグループ ID を揃える。システムは同じグループ ID を持っているノード同士で自動レイアウトを行うことができる。

三次元グラフエディタでは、グラフの編集と同時に自動レイアウトを行っているのでグラフの編集と同時にグループ ID の更新処理も必要になる。例えば、最初のノード A とノード B のグループ ID は自分の ID 1、2 と同じ ID にしてある。しかし、ノードを結線することでノード A とノード B は同じサブグラフのグループに属することになる。システムは二つのノードの ID の中で大きい方、2 をグループ ID として採用することになる。システムはノード A のグループ ID を 2 に更新した後、自動レイアウトの処理を行う。layout.cpp ではグループ ID を更新して、自動レイアウトの処理をしている。

A.1.3 三次元スプリング・モデルの実装の工夫

システムは、スプリング・モデルの適用しているのでノード間に働くバネ力と斥力を算出する。算出されたバネ力と斥力の情報は、各ノードに保存される。全てのノードに対してバネ力と斥力の情報を算出し、保存し終えてからレイアウトの処理を行う。我々が作成した三次元グラフエディタでは、直接操作に対してグラフのレイアウトが動的に変化する。我々は、レイアウトの変化をアニメーションによって表現している。ユーザは、グラフが自動レイアウトされる様子を容易に理解することができる。

A.2 layout.cpp ソース

三次元スプリング・モデルを用いて自動レイアウトするプログラム (layout.cpp) のソースコードを掲載する。

```
/******************************************************************************
** layout.cpp,v 1.0 1999/08/23 11:44:44
** author : Takashi Miyashita
** Implementation of Layout Function
** This algorithm is three-dimensional spring model.
******************************************************************************/

#include <stdio.h>
```

```
#include <stdlib.h>
#include <math.h>
#include <iostream.h>
#include "layout.h"

#define DC1 30.0     // スプリング定数
#define DC2 400.0     // スプリングの長さ
#define DC3 2000000.0     // 非隣接頂点間定数 (使ってない)

/*!
  Layout Class
*/
void GLBox::NodeLayout() {
  GUINodeCore* guinode;
  GUINodeCore* guinodenext;
  GUINodeCore* guiedgestart;
  GUINodeCore* guiedgeend;
  GUINodeCore* guinodefind;

  double d,dc,dk,f,fx,fy,fz,dx,dy,dz,d1x,d1y,d1z,d2x,d2y,d2z,
         fx1,fy1,fz1,fx2,fy2,fz2;
  int ID1,ID2,ID3,Flag;
  // マウスボタンが押されている間は自動レイアウトを適用しない。←ドラッ
  // グ中も自動レイアウトが適用されてしまうと、ノードが動いてしまって
  // 最初にノードをつまんだ位置からマウスカーソルがズレて行ってしまう！

  if (!(mouse_button_middle == FALSE &&
   mouse_button_right == FALSE
   &&  gui_pick != NULL)) {
    return;
  }

    for(guinode = gui_list_start->NodeNext; // 初期化ループ
```

```
guinode != NULL;
guinode = guinode->NodeNext){
      ID1 = guinode->EdgeStartID;
      ID2 = guinode->EdgeEndID;
      if( ID1 == 0 && ID2 == 0)
{
  // 初期化
  guinode->Fx = 0.0;
  guinode->Fy = 0.0;
  guinode->Fz = 0.0;
  // グループ ID 初期化
  // ノードであればグループ ID を入れる
  guinode->GrpID = guinode->ID;
}
    }

    for(guinode = gui_list_start->NodeNext; // グループ ID 再付加処理 (リ
スト先頭より開始)
guinode != NULL;
guinode = guinode->NodeNext){
      ID1 = guinode->EdgeStartID;
      ID2 = guinode->EdgeEndID;
      if(guinode->GrpID != 0){
// ノード ID であればグループ ID の値を入れる
ID3 = guinode->GrpID;
      }
      // エッジがあった場合の処理
      if( ID1 != 0 && ID2 != 0)
{
  for(guiedgestart = gui_list_start->NodeNext; //Edge 始点レイアウトノー
ド ID 検索
      guiedgestart->ID < ID1 ;
      guiedgestart = guiedgestart->NodeNext){
```

```
  }
  for(guiedgeend = gui_list_start->NodeNext; //Edge 終点レイアウトノード
ID 検索
      guiedgeend->ID < ID2 ;
      guiedgeend = guiedgeend->NodeNext){
  }
  // グループ ID 再付加
  // 既に継っているがグループ ID が異なる場合の再付加処理
  if(guiedgestart->GrpID != guiedgeend->GrpID)
    {
      if(guiedgestart->GrpID > guiedgeend->GrpID)
{
  guiedgeend->GrpID = guiedgestart->GrpID;
}
      if(guiedgestart->GrpID < guiedgeend->GrpID)
      {
  guiedgestart->GrpID = guiedgeend->GrpID;
}
    }
}
    }

    for(guinode = gui_list_end->NodePrev; // グループ ID 再付加処理 (リスト
後尾より開始)
guinode != gui_list_start;
guinode = guinode->NodePrev){
      ID1 = guinode->EdgeStartID;
      ID2 = guinode->EdgeEndID;
      if(guinode->GrpID != 0){
// ノード ID であればグループ ID の値を入れる
ID3 = guinode->GrpID;
      }
      // エッジがあった場合の処理
```

```
      if( ID1 != 0 && ID2 != 0)
{
  for(guiedgestart = gui_list_start->NodeNext; //Edge 始点レイアウトノー
ド ID 検索
      guiedgestart->ID < ID1 ;
      guiedgestart = guiedgestart->NodeNext){
  }
  for(guiedgeend = gui_list_start->NodeNext; //Edge 終点レイアウトノード
ID 検索
      guiedgeend->ID < ID2 ;
      guiedgeend = guiedgeend->NodeNext){
  }
  // グループ ID 再付加
  // 既に継っているがグループ ID が異なる場合の再付加処理
  if(guiedgestart->GrpID != guiedgeend->GrpID)
    {
      if(guiedgestart->GrpID > guiedgeend->GrpID)
{
  guiedgeend->GrpID = guiedgestart->GrpID;
}
      if(guiedgestart->GrpID < guiedgeend->GrpID)
      {
  guiedgestart->GrpID = guiedgeend->GrpID;
}
    }
}
    }
    for(guinode = gui_list_start->NodeNext; // 基本レイアウトノード
guinode != NULL;
guinode = guinode->NodeNext){
      ID1 = guinode->EdgeStartID;
      ID2 = guinode->EdgeEndID;
      if(guinode->GrpID != 0){
```

```
ID3 = guinode->GrpID;

      }

      if( ID1 != 0 && ID2 != 0)
{
  for(guiedgestart = gui_list_start->NodeNext; //Edge始点レイアウトノー
ド
      guiedgestart->ID < ID1 ;
      guiedgestart = guiedgestart->NodeNext){
  }
  d1x = guiedgestart->shape.vector.x;
  d1y = guiedgestart->shape.vector.y;
  d1z = guiedgestart->shape.vector.z;

  for(guiedgeend = gui_list_start->NodeNext; //Edge終点レイアウトノード
      guiedgeend->ID < ID2 ;
      guiedgeend = guiedgeend->NodeNext){
  }

  d2x = guiedgeend->shape.vector.x;
  d2y = guiedgeend->shape.vector.y;
  d2z = guiedgeend->shape.vector.z;

  //GrpIDの修正
  if(guiedgestart->GrpID != guiedgeend->GrpID){
    if(guiedgestart->GrpID > guiedgeend->GrpID){
    guiedgeend->GrpID = guiedgestart->GrpID;
    }
    else{
    guiedgestart->GrpID = guiedgeend->GrpID;
    }
  }
```

```
  dx = d1x-d2x;
  dy = d1y-d2y;
  dz = d1z-d2z;

  dk = sqrt((dx*dx)+(dy*dy)+(dz*dz));
  //重なっている場合
  if (dk ==0.0){
    dk = 1.0;
    dx =1;
  }
  //  バネの力を計算
  f = DC1*log(dk/DC2);

  fx1 = f*(dx/dk);
  fy1 = f*(dy/dk);
  fz1 = f*(dz/dk);

  // バネ力の代入
    if(gui_pick->ID!=guiedgestart->ID && (guiedgestart->surface))
      {
//    printf("TEST2\n");
guiedgestart->Fx -= fx1;
guiedgestart->Fy -= fy1;
guiedgestart->Fz -= fz1;
      }
    else{
  guiedgestart->Fx = 0.0;
  guiedgestart->Fy = 0.0;
  guiedgestart->Fz = 0.0;
}

    if(gui_pick->ID!=guiedgeend->ID && (guiedgeend->surface))
```

```
      {
guiedgeend->Fx += fx1;
guiedgeend->Fy += fy1;
guiedgeend->Fz += fz1;
      }
    else{
  guiedgeend->Fx = 0.0;
  guiedgeend->Fy = 0.0;
  guiedgeend->Fz = 0.0;
}
}
    }

    for(guinode = gui_list_start->NodeNext; //斥力計算
guinode != NULL;
guinode = guinode->NodeNext){
      if(guinode->GrpID == 0 ){
continue;
      }
      if( guinode->EdgeStartID == 0 && guinode->EdgeEndID == 0)
{
  for(guinodenext = guinode->NodeNext; //
      guinodenext != NULL;
      guinodenext = guinodenext->NodeNext){
    if(guinodenext->EdgeStartID == 0 && guinodenext->EdgeEndID == 0){
      if(guinode->GrpID == guinodenext->GrpID )
{
  Flag = 0;
  for(guinodefind = gui_list_start->NodeNext; //
      guinodefind != NULL;
      guinodefind = guinodefind->NodeNext){
    if( guinodefind->EdgeStartID != 0 && guinodefind->EdgeEndID != 0)
      {
```

```
if(  (guinodefind->EdgeStartID == guinode->ID &&
guinodefind->EdgeEndID == guinodenext->ID)
   ||(guinodefind->EdgeStartID == guinodenext->ID &&
guinodefind->EdgeEndID == guinode->ID)){
  Flag = 1;
  break;
}
      }
  }

  //つながってなかったら
  if(Flag == 0){
    d1x = guinode->shape.vector.x;
    d1y = guinode->shape.vector.y;
    d1z = guinode->shape.vector.z;

    d2x = guinodenext->shape.vector.x;
    d2y = guinodenext->shape.vector.y;
    d2z = guinodenext->shape.vector.z;

    dx = d1x-d2x;
    dy = d1y-d2y;
    dz = d1z-d2z;

    dk = sqrt((dx*dx)+(dy*dy)+(dz*dz));

    //重なっている場合
    if (dk ==0.0){
      dk = 1.0;
      dx =1;
    }
    //    斥力計算
    f = DC3/(dk*dk);
```

```
    fx2 = f*(dx/dk);
    fy2 = f*(dy/dk);
    fz2 = f*(dz/dk);

    //斥力の代入
      if(gui_pick->ID!=guinode->ID && (guinode->surface))
{
  guinode->Fx += fx2;
  guinode->Fy += fy2;
  guinode->Fz += fz2;
}
      else{
  guinode->Fx = 0.0;
  guinode->Fy = 0.0;
  guinode->Fz = 0.0;
      }
      if(gui_pick->ID!=guinodenext->ID && (guinodenext->surface))
{
  guinodenext->Fx -= fx2;
  guinodenext->Fy -= fy2;
  guinodenext->Fz -= fz2;
}
      else{
  guinodenext->Fx = 0.0;
  guinodenext->Fy = 0.0;
  guinodenext->Fz = 0.0;
      }
  }
}
    }
  }
}
    }
```

```
    for(guinode = gui_list_start->NodeNext; // レイアウト処理
guinode != NULL;
guinode = guinode->NodeNext){

      ID1 = guinode->EdgeStartID;
      ID2 = guinode->EdgeEndID;
      if( ID1 == 0 && ID2 == 0)
{
      guinode->shape.vector.x += LAYOUT_MOVE*(guinode->Fx);
      guinode->shape.vector.y += LAYOUT_MOVE*(guinode->Fy);
      guinode->shape.vector.z += LAYOUT_MOVE*(guinode->Fz);
}
    }
    updateGL();
}
```

A.3 glbox.cpp ソース (一部)

拡張ドラッグ＆ドロップによってノードを結線するプログラム (glbox.cpp の一部) のソースコードを掲載する。

```
/*********************************************************
 * void GLBox::mousePressEvent で
 * マウスの真中ボタンを押した時の処理
 *********************************************************/
  if(e->button() == MidButton){ // ■中ボタン

    int polynum = CalcTouchPolygon(gui_list_start->NodeNext, e->x(), e->y(),
   &gui_operate,
   gui_world);
    if((void *)gui_operate == NULL){
      past.x = e->x(); past.y = e->y();
    }
```

```
    else{
      // エッジ生成前処理
      NodeEdgeID = 0;// 初期化
      NodeEdgeID = gui_pick->ID;// エッジの始点となる ID を抽出
      moved = FALSE;
      start.x = e->x();start.y=e->y();
      stock.x = gui_operate->shape.vector.x;
      stock.y = gui_operate->shape.vector.y;
      gui_operate->DrawMode(FALSE, FALSE, TRUE);
      updateGL();
    }
    mouse_button_middle = TRUE;
  }
/*********************************************************
 * void GLBox::mouseReleaseEvent で
 * マウスの真中ボタンをリリースした時の処理
 *********************************************************/
    if (e->button() == MidButton) { //■中ボタン
      mouse_button_middle = FALSE;
      last.x=end.x; last.y=end.y;
      if (gui_operate != NULL && gui_pick != NULL ) { // 何かに触れてい
る
if(!(gui_operate->focus)){
  gui_operate->DrawMode(TRUE, FALSE, FALSE);
  printf("MOUSE_TEST\n");
}else {
  gui_operate->DrawMode(FALSE, TRUE, FALSE);
}
moved = FALSE;
// 二重結線防止処理
DoubleID = 0;
for(guinode = gui_list_start->NodeNext;
    guinode != NULL;
```

```
    guinode = guinode->NodeNext){
  if( guinode->EdgeStartID != 0 && guinode->EdgeEndID != 0)
    {
      if(  (guinode->EdgeStartID == NodeEdgeID &&
guinode->EdgeEndID == gui_pick->ID)
   ||(guinode->EdgeStartID == gui_pick->ID &&
guinode->EdgeEndID == NodeEdgeID)){
//結線削除処理
guinode->EdgeStartID = 0;
guinode->EdgeEndID = 0 ;
tmpguinode = guinode;
guinode = tmpguinode->NodePrev;
guinode->NodeNext = tmpguinode->NodeNext;
if(tmpguinode->NodeNext != NULL){
  guinode = tmpguinode->NodeNext;
  guinode->NodePrev = tmpguinode->NodePrev;
}
else{
  gui_list_end = guinode;
}
delete tmpguinode;
DoubleID = 1;
break;
      }
    }
}
if(DoubleID == 0 && NodeEdgeID != gui_pick->ID){
  guiedge = new GUINodeCore;
  gui_list_end->NodeNext = guiedge;
  guiedge->NodeNext = NULL;
  guiedge->NodePrev = gui_list_end;
  gui_list_end = guiedge;
  guiedge->EdgeStartID = NodeEdgeID;//エッジの始点となるIDを抽出
```

```
  guiedge->EdgeEndID = gui_pick->ID;// エッジの終点となる ID を抽出
  printf("Not!!DoubleID!!!\n");
}
updateGL();
//GrpID を付ける
if(gui_operate->ID > gui_pick->ID){
  gui_operate->GrpID = gui_operate->ID;
  gui_pick->GrpID = gui_operate->ID;
}
else{
  gui_operate->GrpID = gui_pick->ID;
  gui_pick->GrpID = gui_pick->ID;
}
gui_operate = NULL;
      }
      else{
if(gui_operate != NULL){
  gui_operate->DrawMode(TRUE, FALSE, FALSE);
    }
updateGL();
gui_operate = NULL;
      }
    }
```