ヒューマンインタフェース 58－5
（1995. 1. 19）

経路方式によるアルゴリズムアニメーション

河合 謙　徳田 雄洋

東京工業大学 工学部 情報工学科

アルゴリズムアニメーションには，アニメーション部分の作成に非常にコストがかかるという問題点がある．この問題点を回避する方法として経路起動法を提案する．経路起動法では，変化の経路と変化の型を組にしたものをメソッドとして図形イメージに定義しておき，これをアルゴリズムのプログラムから発生するイベントに応じて起動してアニメーションを実現する．経路起動法は，従来の方法と比べてアニメーション定義におけるコード記述が非常に少ないという特徴を持っている．本論文では，アルゴリズムアニメーションの性質に基づいて経路起動法を提案し，設計原理を説明する．さらに，記述例としてバブルソートのアニメーションの例を示す．

An Algorithm Animation Based on Path-Activation Method

Ken KAWAI　Takehiro TOKUDA

Department of Computer Science, Tokyo Institute of Technology

One serious problem of algorithm animations is that it costs too much to create an algorithm animation. To solve this problem, we propose a new method which we call Path-Activation method. In this method, algorithms are animated by activating methods defined in images. As a result, it requires less amount of code to define animations. A method consists of a motion type and a path. Methods are selected according to events of an algorithm process. We explain our design principles of Path-Activation method based on characteristics of algorithm animations and an example of a bubble sort animation.

1 はじめに

アルゴリズムアニメーションとは, プログラムの実行過程のデータや操作, 意味などを図形のアニメーションとして表示することである. アルゴリズムアニメーションは, プログラムの実行の様子を視覚的に表現できるので, プログラムの内部状態を直観的にとらえやすい. そのため, プログラムの理解, プログラムの性能評価といった用途に利用される.

アルゴリズムアニメーションを作る際, 多くの場合はアルゴリズムを記述したプログラムが既に存在し, このコードにアニメーションのコードをつけ足すことによってアルゴリズムアニメーションを作成する. この時, アルゴリズムのコードとアニメーションのコードはうまく同期をとるように作らなくてはならない. この点が, アルゴリズムアニメーションと一般のアニメーションの違いであり, アルゴリズムアニメーションの作成に非常にコストがかかってしまう, という大きな問題を引き起こす.

従来のアルゴリズムアニメーション作成システムとして, Balsa[1], Balsa-II[2], Zeus[3], Tango[4] などのシステムがある. これらのシステムでアルゴリズムアニメーションを作るためには, すべてをプログラムとして記述しなくてはならない. このことは, アルゴリズムアニメーションの作成者にとって大きな負担となっている.

また, アニメーションの部分の記述には, 図形イメージの定義, 図形の動作, アルゴリズム中の変数などの状態とアニメーション上の図形との対応関係, およびそれらの制御を記述しなければならない. 従来のシステムでは, これらを 1 つのプログラムとしてまとめて記述する方法でアニメーション定義をしていた. この点もまた, アルゴリズムアニメーションの作成を困難にしている原因である.

本論文では, アルゴリズムアニメーション作成の方法論として経路起動法を提案する. 経路起動法では, アニメーション定義を各機能ごとに分離して記述でき, プログラムによるコード記述を最小限におさえ, かつ非常に簡潔に記述できる方法である.

以下, アルゴリズムアニメーションの一般的性質を検討し, 経路起動法の設計の過程を述べる. そして, 経路起動法を提案し, 経路起動法によるアルゴリズムアニメーションの記述法を述べ, 最後に記述例としてバブルソートの例を示す.

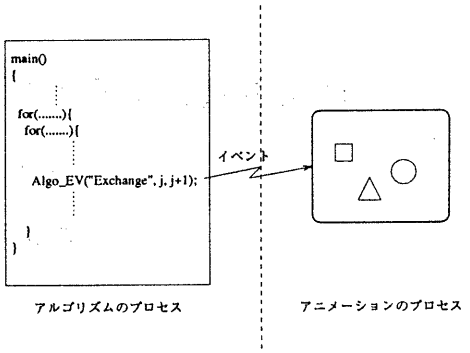


図 1: アルゴリズムとアニメーションとの間のインタフェース

2 アルゴリズムアニメーションの性質

アルゴリズムアニメーションは, 以下のように一般のアニメーションとは異なったいくつかの性質を持っている. 経路起動法は, これらの性質を利用してアニメーション定義のコストを少なくする方法である.

ここでは, アルゴリズムアニメーションの性質について整理し, その性質から経路起動法の設計への過程を述べる.

2.1 アルゴリズムのプログラムとアニメーションとのインタフェース

アルゴリズムアニメーションでは, アルゴリズムのプログラムが既に存在する場合が多い. 存在する場合もしない場合も, アルゴリズムの定義とアニメーションの定義は分離されていることが望ましい. また, アルゴリズム中のすべての変数を常に監視していなければならないわけではなく, 適当に情報を選択してアニメーションとして表現する必要がある. 情報量が極端に多いアルゴリズムアニメーションは, かえってわかりにくくなってしまうためである [6].

したがって, 経路起動法におけるアルゴリズム部分とアニメーション部分との間のインタフェースは, 図 1 のように, アルゴリズムのプログラムにイベン

ト発生命令のみを挿入し, イベントによってアルゴリズムのプロセスからアニメーションのプロセスへ情報を渡す機構とする. 例えば, ソートのアルゴリズムの場合, 配列中の 2 つの数値を交換する動作のように, そのアルゴリズムにおける特徴のある場所にイベント発生命令を挿入する.

2.2 図形

アルゴリズムアニメーションの画面上に登場する図形の種類は比較的少ない. すなわち, 同じ振舞いをする図形が数多く登場する. バブルソートなどのソートのアルゴリズムアニメーションを作る場合, 例えば数値の大きさに比例した高さを持つ四角形としてソートされる数を画面上に表現し, それらが交換される様子をアニメーションする例が考えられる. この場合の「ソートされる数値を表現した四角形」は, すべての四角形が似た動きをする. もう一つの例として, 2 分探索木に数値を追加, 削除する様子をアニメーションする例を考えると, 追加や削除される数値は木の節点の部分に置き, それらの間に木の枝をはった表現が考えられる. 数値の追加, 削除に応じて数値を木の枝に沿って移動させ, それにともなって木の枝を追加したり削除したりする. この場合の「追加・削除される数値を表す木の節点」や「木の枝」は, それぞれ同じような振舞いをする図形が画面上に数多く登場する.

これらの「四角形」や「木の枝」, 「木の節点」といったものは, 同じ動作をするものをいくつも複製して実現できる. そして多くの場合, これらの図形は複雑な動きをするわけではなく, 数種類の単純な動きだけができれば充分である. 図形が複雑な動きをするようなアルゴリズムアニメーションは, かえってアルゴリズムがわかりにくいものとなってしまう.

以上のことを踏まえ, 経路起動法では図形イメージに動作を組み合わせたデータ構造を導入する. 図形の動作は経路遷移法 [5] に基づき, 「図形の動作経路」と「図形動作の種類」の組で定義する. 経路遷移法は, アニメーションを時系列で並んだ絵の列として捉えずに, 図 2 のように, 「図形イメージ」・「動作の種類」・「動作経路」の 3 つ組によりアニメーションを記述する方法である.

経路起動法では, 経路遷移法を次のように拡張する. 経路遷移法では, 図形イメージ・動作経路は独立なものとして定義する. しかし, アルゴリズムアニメーションでは, 動作経路と図形イメージは対応していることが多い. また, 前述のようにアルゴリズムアニメーションには,

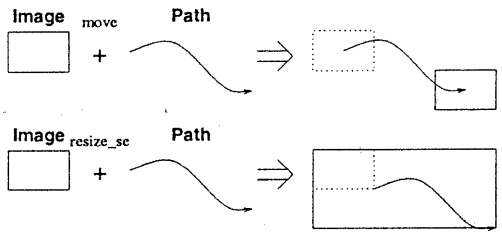


図 2: 経路遷移法

- 同じ図形が多く複製される
- それぞれの図形は, 単純な動作ができればよい

という性質がある. そのため, 経路起動法ではアニメーションの記述単位として, 図形イメージに動作経路を定義したデータ構造を導入する. これを "キャスト" とよぶ. 図形イメージ・動作の種類・動作経路を独立に扱わず, 動作の種類・動作経路の組を図形イメージに対して定義することで, それらを有機的に結合させた.

このキャストを必要なだけ複製して画面上に配置する. このキャストの複製である画面上の図形を "キャストのインスタンス" と呼ぶ.

図形イメージに対してあらかじめ何種類かの動作を定義しておき, 図形にメッセージを送ると, 図形はそのメッセージに対応した動作をする. ひとつひとつの動作は単純だが, 組み合わせることにより複雑な動作をさせる.

2.3 全体のコントロール

アニメーションにおけるローカルな定義は, 前節で述べたように図形定義の部分で行うことができる. ここでは, アニメーションの定義のグローバルな部分について考える.

アニメーション部分のグローバルな処理には, 以下のような処理がある.

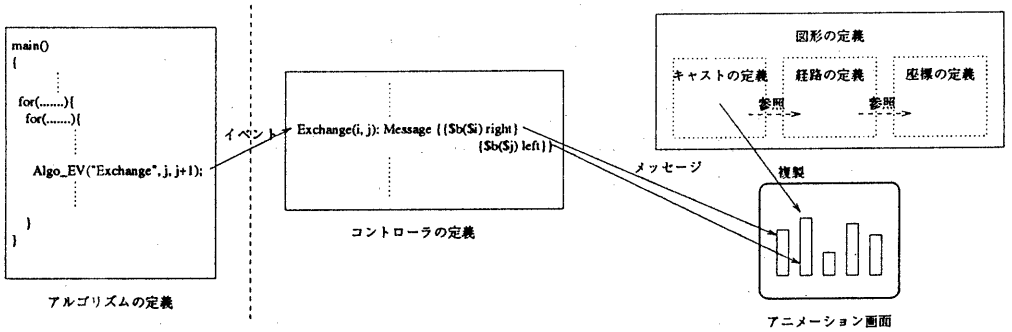


図 3: システムの概観

1. 画面上の図形を生成, 消滅させる.
   必要に応じて, キャストのインスタンスを画面上に生成させたり, それらを消滅させたりする.

2. アルゴリズム側の変数と画面上の図形との対応を保持する.
   アルゴリズム中の変数などの「状態」とアニメーション上の「図形」は, 1 対 1 対応をしている. それらの間の対応関係を保持する機構が必要である.

3. 画面上の図形へメッセージを送出する.
   キャストのインスタンスは, メッセージを受けとることにより動作をする. また, アルゴリズムのプロセスからはイベントが発生する. アルゴリズムのプロセスからのイベントを解釈して, キャストのインスタンスへメッセージを送出する機構が必要である.

経路起動法では, これらの処理はコントローラと呼ばれる部分で行う.

3 経路起動法

経路起動法では, 次の 3 種類の定義をシステムに与えてアルゴリズムアニメーションを定義する.

- アルゴリズムの定義
- 図形の定義
- アニメーションのグローバルな定義

これらの定義間の関係は図 3 のようになる.

アルゴリズムのプログラムに挿入されたイベント発生命令からイベントが発生すると, コントローラがそのイベントを解釈し, コントローラの定義にしたがってアクションを起こす. そのアクションがキャストのインスタンスに対するメッセージの場合, メッセージを受けとったキャストのインスタンスは, そのメッセージに対応した動作をする.

以下に各定義の概略を示す.

3.1 アルゴリズムの定義

多くの場合, 既存のアルゴリズムのプログラムをそのまま利用する. アルゴリズムのプログラムがない場合は, アルゴリズムのプログラムを作成しなくてはならないが, その際アニメーション部分をほとんど意識しなくてもよい. このアルゴリズムのプログラムに, イベント発生命令を挿入する. それ以外に手を加える必要はない.

3.2 図形の定義

図形定義は, アニメーション定義のうちのローカルな部分に相当する. 座標・経路・キャストの定義がある.

3.2.1 座標

"座標" は, アニメーション画面上の位置である. 座標として, 以下のものが扱える.

- 固定座標

- 任意の数値の組
- 画面上の任意の図形の座標

固定座標は, 対話的なツールにより画面上の一点を指定することで定義する. $\{x\ y\}$ というリストによって固定座標を指定することもできる. また, ひとつの座標と次節で説明する経路との和でも座標を定義できる. 座標 *loc* と, 経路 *path* との和は, *path* の始点の位置を *loc* とした場合の *path* の終点の座標を示す.

3.2.2 経路

"経路" は図形が動作する時の道すじで, 画面上の座標の列である. 始点と終点が定まっていれば, 閉路や重なりがあってもよい.

図形が経路に沿って動作するのに要する時間を "経路の長さ" と呼ぶ. 例えば, ある経路の長さを 10 とすると, 図形がその経路に沿って変化するのに 10 単位時間を要する.

経路に引数を与えることにより, 経路の図形的, 時間的な長さなどの属性を変化させることができる.

経路は図形が動作する時の道すじとして使われる他に, 前述のように座標を指定する用途にも用いられる.

3.2.3 キャスト

"キャスト" は動作が定義された図形イメージである. アルゴリズムアニメーションにおける画面上の図形は, このキャストのインスタンスである. 直線・多角形・円弧・文字列などが, 図形イメージとして扱える.

描かれたそのままの図形イメージを, キャストのインスタンスとして画面上に複製する配置の方法の他に, 配置時に引数を指定し, その引数の大きさによって図形イメージの高さや幅を変化させることができる. ただし, この方法はキャストのインスタンスの配置時にのみ有効で, 配置後に図形を変化させるためには, メッセージにより図形に動作をさせる方法を使わなければならない.

図形に定義された動作を "メソッド" と呼ぶ. メソッドは, 次のようにメソッド名・動作の種類・経路の組で定義する. 例えば,

right: move p1

と, あるキャストに定義した時, このキャストは 'right' というメッセージを受けると 'p1' という名前の経路に沿って移動することを意味する.

移動の他に, 図形イメージを変形させる動作, 色・塗りつぶしなどを変化させる動作, 可視化・不可視化させる動作などがある.

3.3 グローバルな定義

アニメーション処理のうちグローバルなものは, 2.3 節で示したように 3 種類のものがある. このグローバルな処理は, コントローラで実行される.

アルゴリズムのプログラムで発生したイベントは, コントローラで命令の列に変換される. コントローラにおける命令を "アクション" とよぶ. コントローラに与える定義は, 次のような形式で定義する.

$$event(i,\ j,\ ...)\colon\ action_1;\ action_2\ ...$$

コロンの左側に書かれたイベント名 *event* にマッチするイベントが発生すると, コロンの右側に書かれたアクションの列 ($action_1$, $action_2$, ...) が順番に実行される. イベントに引数が付加されている場合は, 変数 i, j, ... に代入される.

アクションには, 図形配置命令, 変数への値の代入命令, メッセージ発生命令などを書く.

図形配置命令は, 次のような形式である.

Put *cast loc arg*

これは, '*cast*' という名前のキャストのインスタンスを, '*loc*' の位置に配置することを表す. '*arg*' は, キャストに渡す引数のリストで, この値によってキャストのインスタンスの高さや幅などの初期値を指定する.

コントローラでは, 変数として連想配列や線形リストが扱える. コントローラにおける変数は, 主にアルゴリズム中の状態と, それの視覚的表現であるアニメーション画面上の図形との対応関係を保持するために用いられる. 図形配置命令を実行すると, 返り値として図形の識別番号を返す. この識別番号を変数に保存しておき, 図形にメッセージを送る時にはこの番号で送り先の図形を指定する.

キャストにメッセージを送るには,

Message $\{\{id_1\ message_2\}\ \{id_2\ message_2\}...\}$

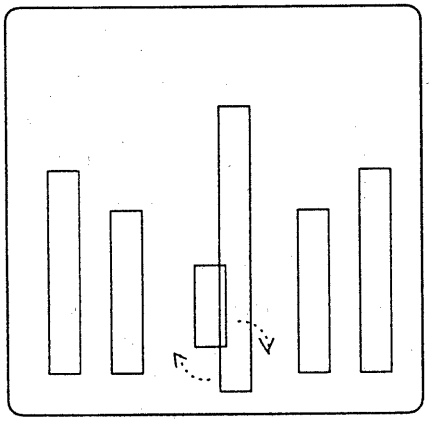


図 4: バブルソートの例

```
int data[50], max, i, j, tmp
main()
{
    scanf("%d", &max);
    Algo_EV("Init", max);
    for(i = 0; i < max; i++){
        scanf("%d", &data[i]);
        Algo_EV("Input", i, data[i]);
    }
    for(i = max - 2; i >= 0; i--)
        for(j = 0; j <= i; j++)
            if(data[j] > data[j+1]){
                tmp = data[j];
                data[j] = data[j+1];
                data[j+1] = tmp;
                Algo_EV("Exchange", j, j+1);
            }
}
```

図 5: バブルソートのアルゴリズム

のようにアクションを記述する. 図形の識別番号と送るメッセージ名のリストを Message 命令に与える. この組を複数 Message 命令に与えると, それらのメッセージは同時に各図形に送られる.

メッセージに引数を与えると, その引数はそのままメソッド内の経路に渡される. 前述したように, 経路はこの引数を使って高さや幅, 終点の位置などを変化させることができる.

4 経路起動法による記述例

経路起動法によるアルゴリズムアニメーションの記述例として, バブルソートの例を示す. ここでは, 図 4 のようにソートされる数値に比例した高さを持つ四角形が画面上に並び, アルゴリズムのプロセスで 2 つの数値が交換されると, アニメーション画面では図の矢印のように四角形が動いて交換されるアルゴリズムアニメーションを記述する.

試作システムにおけるアニメーション定義の様子を図 8に示す.

4.1 アルゴリズムの定義

アルゴリズムのプログラムは, C 言語で記述されたものを利用する. 図 5のリスト中のイタリック体の部分に, Init, Input, Exchange という名前のイベントを発生する命令を挿入した.

4.2 キャストの定義

このアルゴリズムアニメーションに現れる図形は, ソートされる数に対応した四角形の 1 種類だけである.

この四角形をキャストとして定義すると, 図 6(a) のようになる. 四角形の幅は固定であるが, ソートされる数値の大きさによって四角形の高さを変えなければならないので, 四角形の高さは 'arg0' という指定をしている. この指定により, 図形配置命令で与えられる引数リストのいちばん最初の要素が四角形の高さとして適用される.

このキャストには, 'right', 'left' と名付けられたメソッドが定義されている. right, left というメッセージをこのキャストのインスタンスが受けとると, それぞれ次節で定義する経路 'p1', 'p2' に沿って移動する.

4.3 経路の定義

バブルソートでは, 必ずとなりの数値同士で交換が起こる. この例では図 6(b) のように, 数値を表す四角形がひとつ右に動く経路 (p1) とひとつ左に動く経路 (p2) の 2 種類を用意する. 必ずとなり同士なので, 経路のスケーリングは必要でなく, 固定長で良い.

このバブルソートのアルゴリズムアニメーションの例では, 最初に四角形を等間隔に画面上に配置しなければならない. この配置の時に, 四角形と四角形の間隔を経路を利用して指定する. この経路の名前を 'p3' とする (図 6(c)). 経路の x 座標のスケーリングを 'arg0' と指定しているので, p3 に渡される引数リストのいちばん最初の要素で p3 は x 方向にスケーリングされる.

4.4 座標の定義

この例では, キャストのインスタンスを画面上に配置する時に利用する座標が, ひとつだけ必要である. この座標を 'loc1' とする.

loc1 は, 画面上の適当な場所をマウスでクリックすることにより定義する.

4.5 コントローラの定義

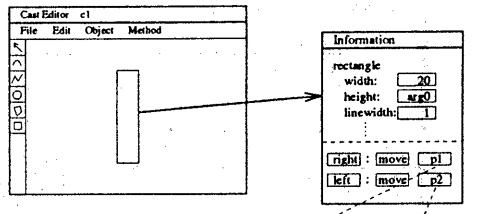
コントローラの定義では, 図 7のようにアルゴリズムのプログラムから発生する 3 種類のイベントについてアクションを定義する.

Input イベントが発生するたびに図形配置命令が実行されて, 画面上にキャスト c1 のインスタンスが配置される. このとき, 経路 p3 には引数として変数 i の値が渡され, p3 の長さは x 方向に i 倍される. そして, c1 のインスタンスは loc1 と p3 の和の位置に配置される. また, キャスト c1 には, 変数 j の値が引数として与えられる. 図形配置命令 Put の返り値は, 配置されたキャストのインスタンスの識別番号である. これは, a という配列に保存される.

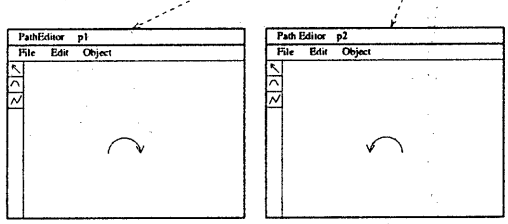
Exchange イベントが発生すると, Message 命令によって i 番目の四角形と j 番目の四角形にそれぞれ right, left というメッセージが同時に送られる. Exchange イベントのアクションでは, アルゴリズムの変数と画面上の Cast のインスタンスとの対応関係を保つため, 配列 a の i 番目の要素と j 番目の要素を交換することも行われる.

5 おわりに

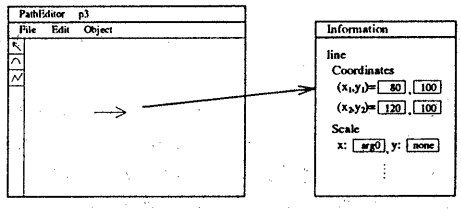
本論文では, アルゴリズムアニメーション作成の方法論として経路起動法を提案した. 経路起動法は, アルゴリズムの定義とアニメーションの定義が



(a) キャスト c1 の定義



(b) 経路 p1, p2 の定義



(c) 経路 p3 の定義

図 6: 図形の定義

```
Init{i}:  set m $i
Input{i,j}:
    set a($i) [Put c1 {loc1 {p3 $i}} {$j}]
Exchange{i,j}:
    Message {{$a($i) right {}}
                    {$a($j) left {}}};
    set tmp $a($i);
    set a(i) $a($j);
    set a(j) $tmp;
```

図 7: コントローラの定義

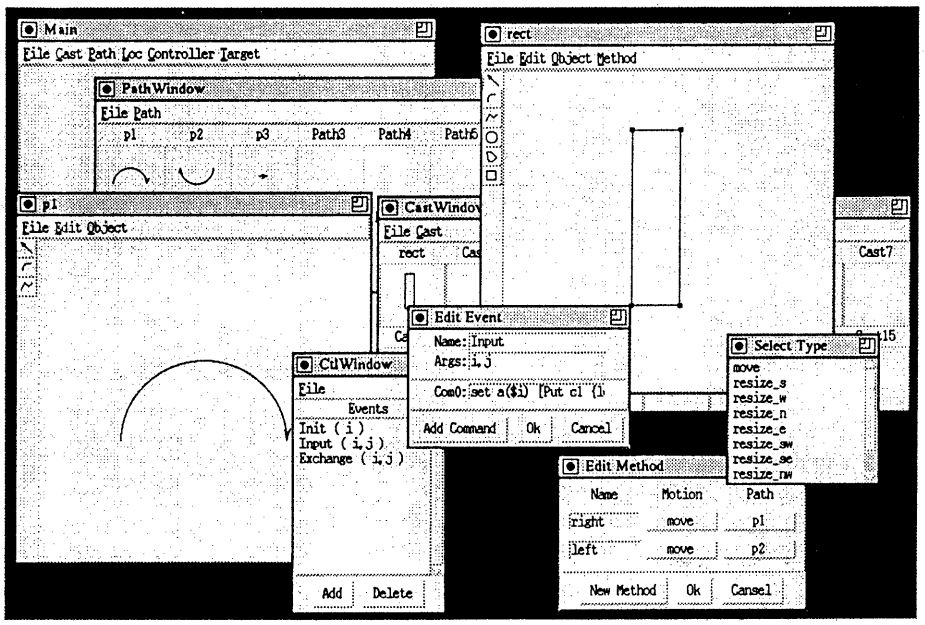


図 8: 試作システムにおけるアニメーション定義の様子

分離されていて, さらに, アニメーションの定義はローカルな定義とグローバルな定義に分離されているという特徴を持っている. 従来の方法は, アニメーション定義をすべてコード記述により行っていたのに対し, 経路起動法はアニメーションのローカルな定義ではコード記述は不要で, グローバルな部分も非常に簡潔なコードで記述できる方法である.

経路起動法では, アルゴリズムのプロセスからアニメーションのプロセスへの 1 方向のイベントしか仮定していない. 理解しやすいアルゴリズムアニメーションを作るためには, 初期値の入力等のインタラクションが不可欠である. 今後の課題として, 経路起動法に双方向のイベント機構を導入することを考えている.

参考文献

[1] Marc H. Brown and R. Sedgewick, Techniques for Algorithm Animation, *IEEE Software*, Vol.2, No.1, January 1985, pp.28-39.

[2] Marc H. Brown, Exploring Algorithms Using Balsa-II, *Computer*,Vol.21, No.5, May 1988,pp.14-36.

[3] Marc H. Brown and John Hershberger, Color and Sound in Algorithm Animation, *Computer*, Vol.25, No.12, December 1992, pp.52-63.

[4] John T. Stasko, Tango: A Framework and System for Algorithm Animation, *Computer*, Vol.23, No.9, September 1990, pp.27-39.

[5] John T. Stasko, The Path-Transition Paradigm: A Practical Methodology for Adding Animation to Program Interfaces

[6] John T. Stasko, Albert Badre, Clayton Lewis, Do Algorithm Animations Assist Learning? An Empirical Study and Analysis, *Proceedings of the ACM INTERCHI '93*, pp.61-66.