7P−2　X Window上での実時間アニメーション法

梶本 雅人　松浦 敏雄　谷口 健一

大阪大学 基礎工学部 情報工学科

1 まえがき

近年, X Windowを備えたUNIXワークステーションが急速に普及してきており, これらを用いたプレゼンテーションシステムやユーザインタフェースとして, 計算機の画面上でのアニメーションへの要求が高まってきている. しかし, 従来この種のアニメーションは, 描画速度の問題等から, 比較的単純なものを対象とし, 実時間性は重視されず, 前後関係のみがわかれば良い場合に用いられてきた. しかし, 動画像や音声といった他のメディアとの同期の必要性から, 実時間性も重要になってきている. ところが, 刻々の表示画像の計算時間や画面表示の所要時間がボトルネックになってスムーズな実時間アニメーションが行なえない場合がある. 本研究では, X Window上でのアニメーションに対して, 実時間性を損なわず, かつ, できるだけスムーズなアニメーションを行う方法を提案している.

2 アニメーションのモデルと問題点

2.1 アニメーションのモデル

本研究では, アニメーションに登場する物(*Cast*)の動きを記述したシナリオに基づくアニメーションシステムを対象としている. シナリオでは, 以下の点を記述する.

- [*Cast*の形状] 各*Cast*の形状は, 固定した1つのimage(bitデータの集まり), または, 時間とともに変化する形状（例えば, 人が歩いている様子など）を表現するための複数のimageによって定義される.
- [*Cast*の動作] シナリオでは, 指定した時間内での, それぞれの*Cast*の2次元平面上での移動・拡大縮小・回転等の動作を記述でき, 平面上で重なりあった*Cast*に対して, その遠近関係を指定できる.
- [実行順序の制御] *if*文や*for*文により, シナリオの実行順序を制御できる. 条件部には, 変数の値や*Cast*の位置などを用いた論理式を指定できる.
- [実行時の入力] 実行時のマウス入力やキー入力によって, 実行順序や*Cast*の配置等を変えることが出来る.

アニメーションは, 静止画（フレームと呼ぶ）を短時間に連続して表示することで実現される. 対象とするアニメーションシステムでは, 1秒間の描画フレーム数(FPS)を指定し, それを満たしつつ, できるだけシナリオで記述した通りの動きを画面に表示する.

A Real Time Animation Method on X Window System

Masato KAJIMOTO, Toshio MATSUURA, Kenichi TANIGUCHI

Dept. of Information and Computer Sciences, Osaka Univ.

2.2 アニメーションの問題点

*Cast*の拡大縮小・回転等の動作を行なう場合, そのimageの計算に時間がかかる. また, 複数の*Cast*が重なりあった付近を他の*Cast*が通過するような場合, 通過前後の画面を生成するには, 重なりあった*Cast*を順に何枚も描画しなければならず, フレームの描画に時間がかかる. いずれの場合も$1/FPS$秒以内に次のフレームを表示できない場合が起こる. このような場合に, いかにしてスムーズなアニメーションを行うかが問題となる.

3 提案するアニメーション法

3.1 リハーサルの利用

本研究で提案するアニメーションの方法は, リハーサルを行なうことにより, 要求された時間($1/FPS$秒)内に描画できない部分を見つけだし, 実行時に高速に描画するために必要な情報(リハーサル情報と呼ぶ)を保存し, 実行時にはそれを利用してスムーズなアニメーションを実現するものである(図1参照).

しかし, 大きな*Cast*が動く場合など, リハーサルを行なっても$1/FPS$秒内に表示できない場合もある. このような場合は, スムーズさを犠牲にして, FPSを減らして描画する.

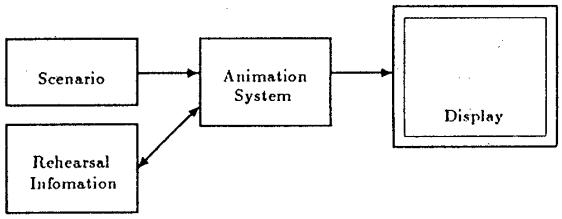

図1. アニメーションシステムの構成図

3.2 リハーサルおよび実行時の描画方針

リハーサルの方針としては, リハーサル中にもユーザからの入力を受けつけるので, なるべく違和感を与えないために, できるだけ実時間で描画を行ないながら(単純描画法(3.3)を用いる), リハーサル情報を収集する. ただし, 変形を伴う*Cast*などでは, 本来の形状のイメージを用意してない場合もあり, また, 描画に時間がかかりFPSが要求値より下回ることもある.

実行時の描画方針は, あるフレームから次のフレームを作る方法に, 単純描画法(3.3)または差分描画法(3.5)のいずれかを用いる. 差分描画法は多くのメモリを必要とするのでなるべく単純描画法を用いる. 可能な限り$1/FPS$秒間隔でフレームを描画する. この間隔で描画できない部分のみFPSを減らして描画する.

3.3 単純描画法

この描画法は，リハーサル時も実行時にも用いられ，どんな場合にも，以下の1～3の方法で次のフレームを描く．拡大縮小や回転によって形が変わる *Cast*(*RC*と呼ぶ) については，それらのimageをその場で計算せず，実行時であっても*RC*の描画に必要なimageがない場合，近似のimageで描画する．従って，本来の大きさや回転角で描画されないこともある．

1. *RC*と，形は変わらないが移動する *Cast*(*MC*と呼ぶ) を消去するため，*RC*，*MC*が占めていた領域を背景のimageで描く．さらに，元のフレーム上でこれらと重なりあったすべての *Cast* をmarkする．

2. シナリオの記述とその時の時刻から各 *Cast* の次のフレームでの位置・大きさを計算し，*MC*，*RC*，および，次のフレーム上でこれらの上に重なる *Cast* にmarkする．

3. markされた *Cast* を重なりあった順に画面上に描く．ただし，*RC*については，変形後のimageが既に保存されているならばそれを用いるが，そうでない時は，保存してあるimageの中から最も近い大きさ (回転角) のimageを用いて描く．リハーサルを行なっていない場合は，通常元のimageしかないのでそれを用いる．リハーサル時には，さらに，本来必要なimageをあとで計算するために，imageの大きさ (または回転角) を一時データとして記憶する ("RC-memo" と呼ぶ)．

3.4 リハーサルの方法

リハーサルは，リハーサルを行いたいシナリオの区間をIとし，$T = 1/FPS$として，以下の手続きを実行する．

1. [どの描画法を用いるかを決める手続き] シナリオの区間Iと時間間隔Tが与えられたとき，区間Iについて，単純描画法で次々とフレームを描画する．時間Tで描画できる区間 *FO* と出来ない区間 *FN* に分類する．他のプロセスの負荷による影響を取り除くため，区間 *FN* について，もう一度同じ操作を試み，いずれも時間Tで描画できない区間のみ *FN* とし，それ以外の区間を *FO* とする．区間 *FO* については，実行時にも間隔Tで単純描画法により描画する．

2. [*RC*のimageを計算する手続き] 単純描画法を実行中に保存した "RC-memo"(3.3) をもとに，*RC*の描画に必要なサイズのimageを作り保存する (RC-imgと呼ぶ)．

3. [差分情報を求め保存する手続き] 区間 *FN* の各フレームについて，前のフレームとの差分のイメージ (前のフレームからこのフレームを作り出すのに必要な画面上の領域を長方形のimageの集合で表したもの) を計算・保存する (これを差分情報と呼ぶ)．このとき，*RC*の描画に必要なサイズのimageがない場合，それを作り保存する．また，差分情報が実行時に再利用できるかどうかを素早く判断するために，各 *Cast* の大きさ，位置，重なりの順を基にhashingを行なって得た値を差分情報を用いて区間 *FN* で差分描画法で描画しながら，それに要する時間を測定する．時間Tで描画できる区間については実行時にも描画間隔Tで差分描画法により描画する．時間Tで描画できない区間については，その区間をIとし，T=T+ΔTとして，1.からの手続きを繰り返す (ΔTの目安として，例えば，1フレームの描画に要する不足時間の最大値の50%)．

3.5 実行時の描画方法

リハーサルを行なっていても，条件文や実行時の入力等により，各 *Cast* の大きさ，位置等が異なる場合があるので，リハーサル情報を利用できないことがある。実行時に，差分情報を利用して指定された (あるいは修正された) 描画間隔で利用できるのは，実行時の諸条件がリハーサルのときと一致したときに限る．

実行時の描画方法は，基本的には，指定された描画間隔にタイマ割り込みをセットし，割り込まれた時点での時刻をチェックし，その時刻に近い描画間隔のフレームを生成する．このとき，差分情報が保存されており，かつ，それが利用できるときには，差分情報を用いて描画する (これを差分描画法と呼ぶ)．差分情報が使えない時は，単純描画法によって描画する．描画中のタイマ割込みは禁止である．

実行状況がリハーサルと異なる場合は，単純描画を行なうのでフレームの描画時間が，タイマの割込み間隔を越える可能性がある．このときは，フレームの表示が終った時点で，次の割り込みがかかる．本方式では，速さを重視しているので，割込みのかかった時点にもっとも近い描画間隔でのフレームを生成する．従って，途中のフレームの表示を飛ばす．

一方リハーサル情報を使っている場合でも他のプロセスの負荷などによって，指定した描画間隔で描画が終らない場合がある．このとき，動きに正確に追従するために，途中のフレームの表示を飛ばして，その間の差分情報を連続して適応し，次に表示すべきフレームを生成し表示する．

4 あとがき

本稿では，スムーズな実時間アニメーションを実現するために，リハーサルによる描画方法を提案した．

本稿で述べた以外に，スムーズなアニメーションを阻害する要因として，他のプロセスの負荷などがある．これについては，アニメーションのプロセスの優先度を上げるか，または，リアルタイムOSを用いることである程度対応できるものと考えている．

今後は，本稿で述べたアニメーションの方法がどの程度有効かを調べていく．

参考文献

[1] C. W. Reynolds. "Computer Animation with Scripts and Actors," *Computer Graphics*, Vol.16, No.3, July, 1982, pp.289-296.

[2] M. H. Brown and R. Sedgewick. "Techniques for Algorithm Animation," *IEEE Software*, Jan., 1985, pp.28-39.