並行制約プログラミングに基づいた ベクターアニメーション作成環境の開発

Design and Implementation of the Environment for Composing Vector Animation

石井 大輔
Daisuke ISHII

早稲田大学大学院理工学研究科情報科学専攻上田研究室
(〒 169–8555 東京都新宿区大久保 3–4–1 早稲田大学大久保キャンパス 61 号館 414
E-mail: dai@ueda.info.waseda.ac.jp)

ABSTRACT. In our project, we designed the framework for composing vector animation easily and efficiently. Based on the framework, we implemented the system called CCAVA. CCAVA models animation with the expression based on *Concurrent Constraint Programming*. With Concurrent Constraint Programming, computing is expressed as a set of *constraints* manipulated by *agents*. Applying this model to the domain of vector animation, we devised the expression with *PVG* and *animator*.

1 背景

今日、WWW のコンテンツやプレゼンテーションの資料として、**ベクターアニメーション** (Vector Animation、以下「VA」とする) を用いることは珍しくなくなり、今後もさらに普及することが予想される。そのため、一般ユーザが簡単な操作で使うことができる VA 作成環境が求められている。

既存の VA 作成環境には改良の余地がある。たとえばユーザが高度な VA を作成するためには、複雑な操作の VA 作成環境 (例: Flash) に習熟する必要がある。逆に簡単な操作の環境 (例: PowerPoint) では単純な VA しか作ることができない。また、これらの環境において細かい操作や繰り返し作業を強いられるなどの問題もある。

このような不便が生じる理由として、VA を表現するためのモデルが低水準なことが挙げられる。そのため、ユーザが VA を、基本的な操作を用いて一から作成していく、あるいはあらかじめ準備されたものを利用する、という選択肢しかなくなってしまう。また、階層的に VA を構築したり、作成した VA を部品化するための柔軟かつ直観的な方法が存在していない。

CCAVA では VA を、**並行制約プログラミング**という計算モデルに基づき、**PVG**(Parameterized Vector Animation) と**アニメータ**という新しい概念により表現した。これらの概念によって VA を柔軟かつ直観的に表現し、効率良く作成する。

2 成果の概要

本プロジェクトでは、おもに論理的な概念や関係を表すような VA を簡単に効率よく作成可能な環境 CCAVA（Concurrent Constraint As Vector Animation）を開発した。

3 設計

3.1 並行制約プログラミング

CCAVA では**並行制約プログラミング** (Concurrent Constraint Programming、CCP) に基づいて VA のモデル化を行った。特に VA の記述に関しては Hybrid Concurrent Constraint Programming[1] のモデルを採用した。

並行制約プログラミングは、**制約ストア**と**エージェント**という概念を用いた計算モデルを持つ。制約ストアに制約の集合が格納されており、制約ストア中の制約をエージェントが操作することで計算が行われるような枠組みである。

制約

CCAVA では並行制約プログラミングにおける制約にあたる概念として、PVG と**補助制約**を用いる。

PVG は一般的なベクター画像情報に**パラメタ**が付随したものである。パラメタの取りうる値を決めることにより、画像の属性を操作することが可能になる。パラメタ

として、ベクター画像の位置座標、変換行列、色、生成個数などを設定することができる。

補助制約は、作成される VA で明示されることはないが、補助的に用いられる情報である。たとえば、ある部品図形が移動する軌跡やある座標が取りうる範囲を示すためなどに用いる。

エージェント

CCAVA ではエージェントにあたる概念として、アニメータを用いる。

アニメータは制約にあたる PVG・補助制約の時間的な変化を表現する。アニメータが PVG 中のパラメタ値を逐次的に設定していくことにより、VA が実現される。

アニメータには他のアニメータを呼び出す操作も含まれ、複数のアニメータを組み合わせてアニメータを構築することができる。

インタプリタ

CCAVA の**合成処理系**は、PVG・補助制約 (=制約) とアニメータ (=エージェント) からなる並行制約言語のプログラムの解釈処理を行う。

3.2 操作環境

CCAVA では汎用的な PVG とアニメータを**アニメーションライブラリ**としてユーザに提供する。ユーザはアニメーションライブラリから目的にあった PVG とアニメータを読み込み、それらを組み合わせ、VA を作成することができる。

また熟練ユーザは、既存形式のベクター画像にパラメタを付加する作業 (PVG) や、簡単なプログラミング作業 (アニメータ) を行うことにより、アニメーションライブラリを拡張することができる。

ユーザが一連の作業を行った後、CCAVA は最終的に既存アニメーション形式のデータを生成し、ファイルへ出力する。

4 実装

CCAVA は Java によって実装した (JDK 1.3.1 において動作確認)。また JAXP[2]、Batik[3]、Rhino[4] などの既存ライブラリを利用している。

CCAVA はおもに GUI 環境、アニメーションライブラリ、制約システム、合成処理系という要素から構成される。以下、これらの要素について説明する。

4.1 GUI 環境

GUI 環境はユーザが対話的に VA 合成処理に必要な情報を入力するための環境である。

GUI 環境はおもに**ライブラリビュー**と**キャンバス**から構成される (図 1)。ライブラリビューはアニメーションライブラリの一覧を表示し、ユーザはここから必要なライブラリの選択・読み込みを行う。キャンバスは入力された情報を表示する。またユーザはキャンバス上において、補助制約の入力と入力情報の編集を行うことができる。

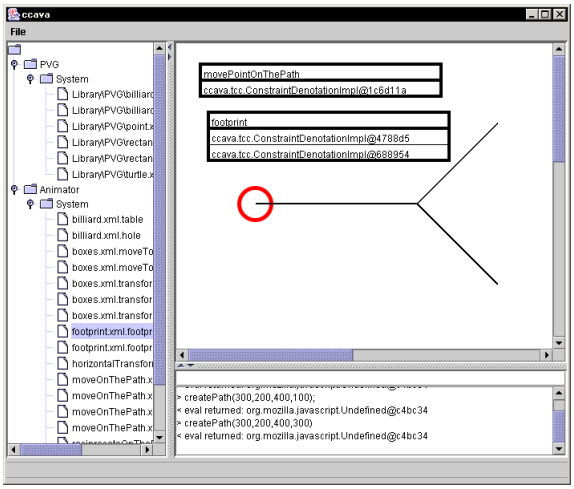

図 1: GUI 環境のスクリーンショット

4.2 アニメーションライブラリ

アニメーションライブラリは PVG とアニメータという 2 種類のデータからなる。

アニメーションライブラリのデータは XML 形式となっている。PVG は SVG[5] の要素が埋め込まれた独自のボキャブラリ、アニメータは独自のボキャブラリにより記述される。

4.3 制約システム

CCAVA の制約システムは PVG や補助制約などの制約、生成された制約の集合を管理する制約ストア、制約への操作を表す組み込み述語からなる。

PVG (および PVG インスタンス、パラメタ値) や補助制約が制約にあたる。制約は PVG 型、浮動小数点型、ベクトル型、グラフィック要素型 (パスや色など) といった型を持っている。CCAVA 内部において制約は、`Constraint` インタフェースを実装し、各型ごとに実装されたオブジェ

クトとして表現される。

制約ストアは生成された制約の集合を表し、その管理を行う。制約の追加・削除をはじめ、ある制約が満たされているかどうかを判定するマッチング処理などを行う。

組み込み述語は制約に対する操作を表し、実際の処理も行う。CCAVA で用意された組み込み述語には、単一化演算子、関係演算子、微分係数、加減乗除算などがある。

4.4 合成処理系

合成処理系は、並行制約言語のプログラムである PVG・補助制約とアニメータを処理し、その結果を既存アニメーション形式のファイルへ出力する。

合成処理

合成処理はおもに次のモジュール群によってなされる (図 2)。

1. AnimationModel

AnimationModel は PVG とアニメータによって構成される VA のデータ構造を統括する役割を担うモジュールである。ユーザが読み込んだアニメーションライブラリ、設定した補助制約などを GUI 環境の裏で管理する。また合成処理中の PVG インスタンスの表を管理する。

2. Context

Context は合成処理中に解釈処理されるアニメータ群が共有する情報を管理する。処理されるアニメータの表、処理中のアニメータへの参照、制約ストアへの参照などを持つ。

3. MainAgent

MainAgent は合成処理されるアニメータ群を統括するアニメータで、合成処理の起点となる。

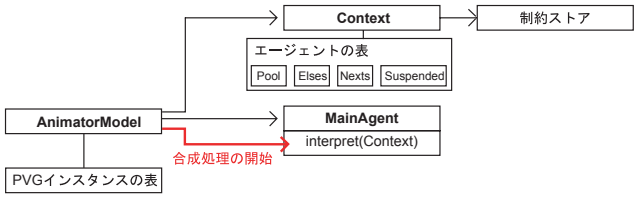

図 2: 合成処理系の構成

PVG マニピュレータ

PVG マニピュレータは合成処理におけるパラメタ操作に関する情報を管理する。パラメタへの操作がなされると PVG マニピュレータへイベントオブジェクトが送られる。

現在の CCAVA では後述のジェネレータがこのインタフェースを実装しており、処理後に結果を出力する。

ジェネレータ

ジェネレータは、既存形式のアニメーションデータをファイルへ出力するモジュールである。現在の CCAVA では SVG + ECMA Script 形式をサポートする `EcmaScriptGenerator` が実装されている。

5 VA の例

ここでは CCAVA を利用して効率よく作成できるような VA の例題を紹介する。

5.1 パス上を移動する PVG

PVG がパス上を移動していくような VA である (図 3)。パスの分岐箇所ではそれぞれの分岐する枝に対し、新しい PVG が生成され、さらに進んでいく。

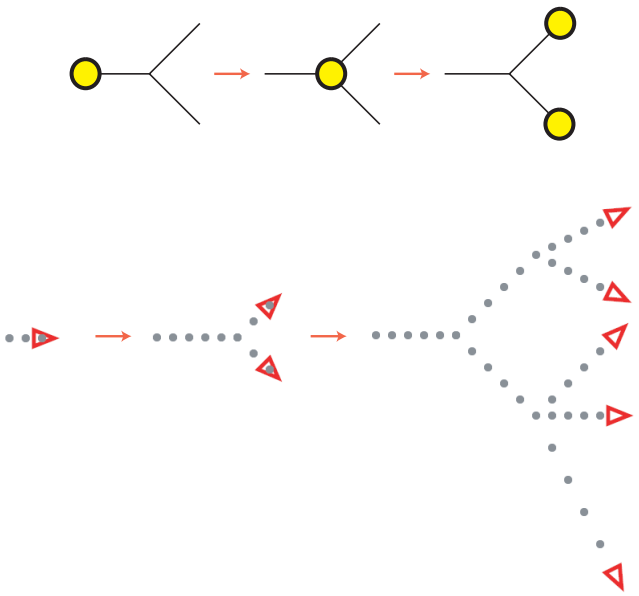

図 3: パス上を移動する PVG

この例を作成するために次の PVG・アニメータを使用する。

- PVG

  turtle パス上を移動する PVG。moveOnThePath アニメータに対しては PVG 自身 (base パラメタ) を渡す。

- アニメータ

  moveOnThePath パス上を PVG を移動させる。パスの端点の判定、PVG の速度の調整などを行う。

  branch パスの分岐点での処理。それぞれの分岐パスに対し、新しい PVG Instance を生成し、moveOnThePath を呼び出す。

  turn パスの分岐点で PVG の方向転換をする。

  footprint PVG の軌跡を残す。

moveOnThePath アニメータは速度の x 変位を一定のまま、y 変位のみをパスに合せて調整する。moveOnThePath の代わりに、たとえば速度ベクトルの長さを一定に保ちながら調整する moveOnThePath2 を使用することで、異なった動作をする VA を作成することができる。

5.2 任意個の PVG の配置を整える

垂直に並んでいる任意個の PVG が水平位置へ移動するような VA である (図 4、図 5)。この例では図 4 を基本動作とし、部品図形の個数が増えた場合は基本動作が複数回適用される (図 5)。

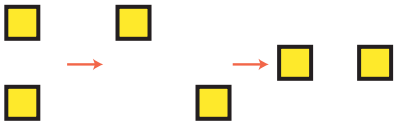

図 4: 任意個の PVG の配置を整える (基本となる動作)

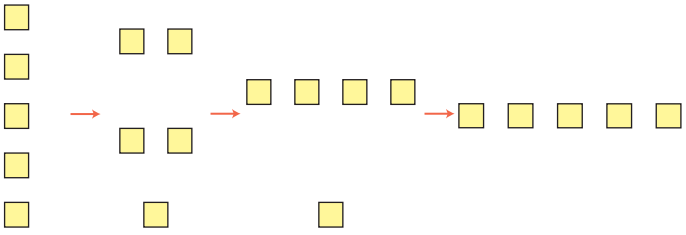

図 5: 任意個の PVG の配置を整える

この例を作成するために次の PVG・アニメータを使用する。

- PVG

  rectangle 部品図形となる PVG。

- アニメータ

  transform 2 個の PVG を引数とし、基本動作を施す。動作終了後、2 個の PVG を合体させる。

transform アニメータは、同時に適用可能な全ての箇所に対して適用される。これは**パラメトリックアスク**という仕組みによってなされる。

6 まとめと今後の展望

本プロジェクトでは、VA を簡単に効率良く作成可能するための環境 CCAVA を開発した。

今回のプロジェクトでは、合成処理系や GUI 環境の実装に手間取り、アニメーションライブラリの整備が十分に行えなかった。有用な PVG 並びにアニメータの作成を行い、ライブラリの整備を進めることが必要である。

現在の CCAVA は数値演算を自前の組み込み関数で処理しているが、今後既存の数値演算処理系を利用し、より複雑な数値関係式による例題を扱うことを考えている。

またユーザがより使いやすくなるようアニメーションライブラリの仕組みを改良していく予定である。たとえば現在のアニメーションライブラリは、PVG・アニメータを一元的な集合として管理しているが、プリミティブなものから複雑・高度なものへと階層的に管理することが有用である。

参考文献

[1] V. Gupta, R. Jagadeesan, V. A. Saraswat, D. G. Bobrow: Programming in Hybrid Constraint Languages. Hybrid Systems II, LNCS 999, Springer Verlag, 1995.

[2] Sun Microsystems, Java API for XML Parsing (JAXP), `http://java.sun.com/xml/jaxp`.

[3] The Apache XML Project, Batik SVG Toolkit, `http://xml.apache.org/batik`.

[4] The Mozilla Organization, Rhino, `http://www.mozilla.org/rhino`.

[5] W3C, Scalable Vector Graphics (SVG), `http://www.w3.org/TR/SVG`.