3Dコンテンツのコラボレーティブオーサリングフレームワーク

A framework for collaborative 3D content authoring

川合 史朗
Shiro KAWAI
E-mail: shiro @ acm.org

ABSTRACT. To produce higher quality 3D Computer Graphics (CG) content, it is necessary for many staff to work collaboratively, sharing large amount of data. Tracking the development of those data with inter-dependency has become a more and more challenging problem, because the existing content authoring software aims at supporting individual work-flow, but not collaborative work-flow. With the traditional authoring tools, the production has to be planned well and broken down precisely before a creative work to start. It wouldn't work nicely, since a creative process inevitably requires trial-and-error and change of plans[1][2].

In this project, we implemented a framework to build authoring software tools that can share a single large 3D scene database (repository), allowing each artist to retrieve, edit, and update a part of the scene individually and simultaneously. The scene repository tracks the update history, dependencies in one part from the other part, and various meta-data including licenses the creator attributed to the parts. The scene repository can be placed on the Net, and we provide a way to name elements in the scene globally using URI.

This architecture enables a new style of production, which everybody can participate in center of a creative process, instead of working in a tiny piece without having global view of the process. It also opens a possibility for small productions and individual artists to produce 3D contents collaboratively over the Net.

1 背景及び目的

コンピュータグラフィクス (CG) アニメーションやビデオゲームの制作時には、大量の相互に関連しあう 3D データを多くのスタッフが並行して作成・編集する必要がある。しかし、既存のオーサリングツールは個人で完結するワークフローを前提としており、そのような状況に対応できない [1][2]。

本プロジェクトは、複数のクライアントがひとつの大きな 3D シーンデータベースの必要な部分だけを並行して参照・変更できるようなフレームワーク・ミドルウェアを製作することを目的とする。

本提案のアーキテクチャの採用により、コンテンツを多人数で「寄ってたかって」制作することが可能になる。スタッフは変更箇所を反映したプレイバックを短いターンアラウンドタイムで見ることができ、全体の中での自分の担当箇所のバランスを自分で判断できる。

ディレクターやプロデューサーは常に現在の最新のプレイバックを目にすることで進行状況を把握できる。

これによって CG プロダクションの生産性を大きく向上させる効果が期待できる。さらに、独立アーティストや小規模プロダクションがネットワーク上で協調して 3D CG コンテンツを制作するという新たな創作形態を可能にする。

2 概念

まず、本ソフトウェア設計上の主要な概念と全体の構成、そして利用イメージに関して述べる。

2.1 主要な概念

ノード (node) 3D CG コンテンツを構成する基本要素。ひとつの 3D CG コンテンツは、ノードが方向を持つポインタで接続された有向グラフとして表現される。

具体的な node としてはオブジェクトの形状を保持する shape、オブジェクトの質感のパラメータを保持する material、質感をもとに shape を適切にレンダリングする shader、オブジェクトの位置情報を制御する transform 等がある。

データ管理上必要なのは、特定の名前付け方法によってノードとそれを指示する名前間に対応づけが可能であること、及びノード間の依存関係を知ることができることであり、具体的なノードの動作はアプリケーションの要請に応じて定義可能である。

シーン (scene) ひとつの画像としてレンダリングすることが可能なノードのグラフ。最低限、カメラ、ライト、及びレンダリングされるべき「もの」が必要である。

従来のオーサリングツールは、このシーンを単位としてファイルへのセーブ・ロードを行っているため、シーンを構成する要素を並行に制作することが困難であった。一方、本システムでは、シーンは画像を表示する側が必要に応じて、次に述べるパートを集めて組み立てるものと考える。

パート (part) 本システムが新たに提案する 3D CG コンテンツの管理単位となるもの。シーンは一つまたは複数のパートから構成される。パートの主要な役割は 2 つある。

第一の役割は、コンテンツ制作の単位となることで

ある。本システムではパート毎にオーサリンルツールへのロード/編集/セーブが可能であり、セーブの度にパート単位で履歴が管理される。シーンは複数のパートで構成可能なため、ひとつのシーンをパートに分割して並行作業することが可能になる。
第二の役割は、ノードの名前空間となることである。3D コンテンツデータの管理上、ノードを名前で一意に指定することが必要になるが、従来のファイルベースのオーサリングツールでは名前空間が存在しないか、アドホックな対応をしていたため、ツールが勝手にノード名をリネームしてしまうなど混乱が多かった。パートで名前空間を区切ることにより、制作単位と名前の管理単位が一致し、管理が容易になる。また、コンテンツのデータが大きくなって来た場合でも、パートの指定とノードの名前でノードが一意に決定できる。

リポジトリ (repository)　制作された 3D CD コンテンツのデータを保持しているデータベース。データの出し入れ (チェックイン/チェックアウト) はパート毎に行われ、またチェックインの履歴は記録される。また、パート間の依存関係をメタデータとして記録しておき、必要に応じて依存しているパート群を同時にチェックアウトすることもできる。

2.2　全体の構成

本システムの基本的なアーキテクチャを図 1 に示す。

3D コンテンツの制作時には、各制作者がオーサリングツールを用いて編集に必要なパートをリポジトリからチェックアウトし、修正を加えてチェックインするという過程を繰り返す。必要に応じて、全体の進行を見るためにまとまったシーンをリアルタイムプレイバック用にチェックアウトして専用のプレビューツールで見ることができる。さらに、ハイクオリティの画像を得るためにチェックアウトしたノードのグラフをオフラインレンダリングツールに渡すことも可能であろう。

編集時に必要な情報とプレイバック時に必要な情報は必ずしも同じではない。例えばある形状を procedural に変形して別の形状を作成したような場合、編集のためには元の形状ノードと変形手続きノードが必要であるが、プレイバック時には最終的な形状のみがあれば良い。チェックアウト時に、リポジトリ内のノードグラフから出力に必要とされるノードグラフへの翻訳手続きを書いてやることにより、ひとつの 3D データを様々な用途へと利用することができるようになる。

2.3　利用イメージ

本ソフトウェアの利用形態として、大きく分けて二つの方法が考えられる。

CG プロダクション内でのワークフローの改善に使う場合は、トップダウン方式が適しているであろう。まず制作すべきシナリオによって各シーンをどのように構成すべきかというブレークダウンが行われ、それに沿って必要なパートのプロトタイプ (位置を決めるためのボックスのみ、等) がまずチェックインされる。その後、パート毎に担当のアーティストが詳細化してゆく。

一方、本ソフトウェアによって可能になる新しい制作形態がある。いわばボトムアップとも呼べる形態である。制作のアイディアを持つユーザーがネット上の開かれた共有リポジトリにアイディアの断片や部分的なシーンをどんどんチェックインしてゆく。それを見て興味を持った別のユーザが、既存のパートを組み合わせたり、新たなパートを付け加えたりしてコンテンツを広げて行く。この制作形態は、ソフトウェア開発におけるオープンソースのアプローチをコンテンツ制作に適用しているとも言えるだろう。

オープンソース方式は、ソフトウェア開発においては良質のソフトウェアを生み出すという点で有効であることが実証されているが、コンテンツ制作においてもその形態が有効であるかどうかは実際に試みてみないと分からない。そのためにも、このような新しい協調制作のフレームワークを実装することは必要な一歩である。

3　開発内容

3.1　開発方針・戦略

前章で述べたシステム全てを実用レベルまで実装するには、開発人員、期間ともに圧倒的に不足していることは明らかである。したがって、目標を絞って開発を進める必要がある。

そこで本開発案件では、2.3 節で述べた利用イメージのうちの後者、すなわち公開されたリポジトリ上でのオープンソース的コンテンツ制作を支援するプロトタイプシステムの実装を目標とする。

フレームワークの中心となる、ノードグラフとパートの管理部分はライブラリとして実装し、将来様々な目的に適用できるようにする。そして、クライアントとしてシーンのエディタとプレイバックツール、サーバとしてインターネットでアクセス可能なリポジトリをそれぞれプロトタイプレベルで作成し、一般に公開する。ソフトウェア開発そのものもオープンソースで行い、興味を持った開発者が参加できるようにする。

従って、未踏ソフトウェア創造事業としての開発案件はライブラリ及びクライアント/サーバーのプロトタイプ製作までとするが、それらは始まりに過ぎない。公開したシーンリポジトリへのフィードバック等を通じて、継続的にオープンソースでの開発を行ってゆく予定である。

3.2　設計

本ソフトウェアの実装言語として、Scheme を主として使用する。処理系には開発者 (川合) がオープンソースで開発している Gauche を利用する[*1]。

Scheme を選択したのは、抽象度の高い言語の方が開発効率が良いこと [3]、Lisp 系言語はグラフを扱うのが得意であること、そして開発者自身が全てを把握している処理系を使うことで、処理系に起因するトラブルを速やかに解決できること、による。

前節で述べた方針から、開発すべきソフトウェアモジュールを次のようにブレークダウンした。

- 支援ライブラリ：本システムの中核部分を実装するための基礎になるライブラリである。
  - 3D ベクタ・行列・4 元数演算ライブラリ
  - GUI ツールキットの Scheme バインディング
- 基本ライブラリ：本システムの中核である、ノードグラフ、パート、シーン、リポジトリを実装するライブラリである。
- ユーザインタフェースライブラリ：基本ライブラリの要素をユーザがグラフィカルなインタフェースで操作するためのライブラリである。主としてクライアントアプリケーションの実装に使用される。
- クライアントアプリケーション：プロトタイプとして、3D シーンのエディタ、ブラウザを提供する。
- サーバリポジトリ：公開サーバリポジトリを運営するためのソフトウェア。
- ノードグラフトランスレータ：リポジトリに蓄積されたコンテンツを別形態で利用するためのソフト

[*1] http://www.shiro.dreamhost.com/scheme/gauche

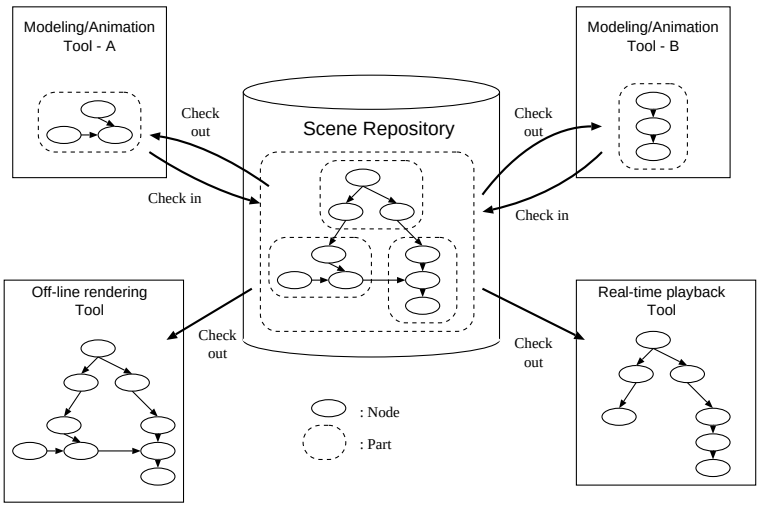

図 1: 本システムの基本的なアーキテクチャ

ウェア。サンプルとして、VRML でシーン情報を出力するプログラムを提供する。

また、本ソフトウェアの製作には以下の外部ソフトウェアを使用する。(*印のあるものは川合自身が開発し公開しているソフトウェアである)。

- OpenGL : 標準グラフィクスライブラリ。
- Gtk+2.0 : GUI ウィジェットライブラリ[*2]。
- GtkGLExt : Gtk 上で OpenGL を使うためのライブラリ[*3]。
- *Gauche : Scheme 処理系[*4]。
- *Gauche-gl : Gauche から OpenGL を使うためのライブラリ[*5]。
- *WiLiKi : テキスト主体の協調 Web 制作環境。シーンリポジトリの web サイトに補助的に使用する[*6]。

3.3 実装

最終的に、表 1 に示す Scheme ライブラリと表 2 に示す実行可能アプリケーションを作成した。

以下に、作成した各モジュールに関する詳細を述べる。

3.3.1 gl.math3d

Gauche には n 次元の配列を扱う汎用的なモジュールはあるが、3D シーンのレンダリングに使うには性能面で不利なため、3 次元の homogeneous coordinate 演算に特化したベクタ・配列の計算を扱うモジュールを作成した。

4 要素のカラムベクタ (`<vector4f>`、`<pointr4f>` クラス)、4x4 の配列 (`<matrix4f>`クラス)、4 元数 (`<quatf>`クラス)、ベクタ配列 (`<vector4f-array>`、`<pointr4f-array>`クラス) のデータ型間の代表的な演算、及び 3D CG において頻繁に必要とされる演算 (トランスフォームと配列の相互変換や合成・分解、回転の 4 元数表現と配列表現の変換等) を C で書き、Scheme から使えるようにした。また、データのバイナリ表現を OpenGL で使われるものと合わせてあり、データ変換無しで OpenGL API へオブジェクトを渡せるようにした。

このモジュールは汎用的であり、かつ OpenGL と親和性が高いため、既に川合がリリースしていた Gauche-gl パッケージに統合した。統合後、Gauche-gl-0.2 から Gauche-gl-0.2.2 まで 3 回のリリースを行っている。

3.3.2 Gauche-gtk

クライアント側の GUI を構築するために、GUI ツールキット Gtk+の機能を Scheme から呼ぶための拡張モジュール Gauche-gtk を作成した。

Gtk+は、特にバージョン 2.0 以降、極めて膨大な API を提供している。バージョン 1.3 までは言語バインディング生成のためのフォーマルな API 記述が提供されていたのだが、バージョン 2.0 ではまだそれも提供されていない。バインディングを全て手で書くのは非現実的なため、Gtk+のヘッダファイルをパーズしてバインディングを自動生成するスクリプトを作成して使用した。C と Scheme でのオブジェクトの扱いが異なるため、いくつかの API は完全な自動生成が出来ない。その分に関しては差分を記述することで対応した。また、Gtk のオブジェクトシステムと Gauche のオブジェクトシステムを統合し、Gtk オブジェクト (ウィジェット) のサブクラスを Scheme で定義できるようになっている。

未踏プロジェクト期間内に Gauche-gtk-0.1, 0.2, 0.2.1, 0.2.2, 0.2.3, 0.2.4, 0.3 と 7 回のリリースを行った。

3.3.3 banyan

本プロジェクトの中心となるライブラリである。ノード、パート、リポジトリ、シーン等の主要概念を実装している。ほぼ全て Scheme で書いたが、表示系の速度の必要な部分だけは C で実装している。

`<node>` (メタクラス `<node-meta>`) ノードを構成する基本クラス。メタオブジェクトプロトコルを用いて、C で実装した部分が Scheme からシームレスにアクセスできるようになっている。また、ノードの実装に関わらずノードの名前や依存関係等のメタ情報を扱える機能をベースクラスに実装している。`<node>`を直接継承するクラスとして、レンダリング可能な (すなわち、目に見える) ノードを表現する`<renderable-node>`と、シーンの状態が変化した場合に再計算を必要とするノードを表現する`<updatable-node>`が定義されている。具体的なノードは、これらのいずれか、もしくは両方を継承して定義される。現在までに定義されているノードを表 3 に示す。

[*2] http://www.gtk.org/
[*3] http://gtkglext.sourceforge.net/
[*4] http://www.shiro.dreamhost.com/scheme/gauche/
[*5] http://www.shiro.dreamhost.com/scheme/gauche/
[*6] http://wiliki.sourceforge.net/

表 1: 実装した Scheme ライブラリ

パッケージ	モジュール名	説明
Gauche-gl	gl.math3d	3D ベクタ・行列 演算 ライブラリ。Gauche-gl の配付パッケージに統合。主として C で実装。
Gauche-gtk	gtk, gtkgl	GUI ツールキットの Scheme バインディング。主として C で実装。単独のパッケージとして配付。
Banyan	banyan	基本ライブラリ。ノード、リポジトリ、パート、シーン等、基本クラスの定義に加え、3D コンテンツを構成、表示するために通常必要なノードの定義を含む。
Banyan	banyan.gui	ユーザインタフェースライブラリ。Gtk 及び下に述べる GLIM を使って、ノードを操作する GUI 部品を提供する。
Banyan	banyan.export	ノードグラフトランスレータ。シーンリポジトリのデータを他の形式で利用するためのトランスレータのフレームワーク、及び VRML への変換。
Banyan	banyan.server	サーバーユーティリティライブラリ。サーバー側の実装に必要なユーティリティ。
Banyan	glim	汎用のユーザインタフェース抽象化ライブラリ。Gtk 等、個々の GUI ツールキットによらずアプリケーションを書くためのフレームワーク。
Banyan	x3d	汎用 X3D/VRML ライブラリ。VRML へのトランスレータで使用。

表 2: 実装したアプリケーション

アプリケーション名	説明
banyan	クライアントアプリケーション。起動時オプションにより、3D シーンエディタまたはブラウザとして動作する。
banyan2vrml	補助スクリプト。Banyan のシーングラフを VRML 形式へと変換する。グラフトランスレーションのサンプルでもある。
repository	サーバーアプリケーション。CGI スクリプトとして動作し、シーンリポジトリに対する HTML でのメタデータ閲覧と application/x-banyan コンテントタイプのチェックイン/アウトのハンドリングを行う。

`<part>` パートを管理するクラス。管理下にあるノードのリストを保持し、チェックイン/チェックアウト時のノードのシリアライゼーションとデシリアライゼーションを行う。また、パート間の依存関係等のメタ情報も管理している (表 4)。
各パートには、ファイルシステムの階層的な名前付けに似た方法で階層的な管理名が与えられる。管理名はリポジトリ内でユニークでなければならない。ネットワーク上に公開されたリポジトリはグローバルにユニークな URI を持つため、管理名と合わせればパートをグローバルにユニークな URI で指定することができる。
パートの制作者は、パートのメタデータとしてライセンスを付加することが出来る。ライセンスの指定方法は Creative Commons[*7] で提供されるフレームワークに沿っており、コンテンツの利用に関して制作者の考えを明示することが出来る。`<cc:licence>`クラスを使えば、パートのメタデータから Creative Commons の定義する形式での RDF 記述が生成可能である。

`<repository>` リポジトリを表現するクラス。現在のところ、ローカルファイルシステムを使用するものと http プロトコルによって通信するネットワーク上のリポジトリの 2 種類のクラスがサポートされている。リポジトリは URI によって指定される。ネット上に公開したリポジトリであれば、その URI を使うことでリポジトリ内のパートがユニークに指定できる。

`<scene>` シーンを実現するクラス。表示可能なノードのリストと再計算が必要なノードのリストを管理し、必要に応じて表示及び再計算のジェネリックファンクションを起動する。また、ユーザインタラクションに必要なセレクションの管理も行う。

[*7] http://creativecommons.org/

3.3.4 banyan.gui

`banyan` モジュールで実装されているノードグラフを実際に表示し、グラフィカルなインタフェースを通じて操作するために必要なクラスを集めたモジュール。後述する banyan クライアントの機能のほとんどはこのモジュール中でライブラリとして実装されている。

3.3.5 banyan.export

`banyan` モジュールで実装されたノードグラフを他のツールでも利用できる形に変換するクラスを集めたモジュール。現在のところ、VRML への変換をサポートしている。

3.3.6 banyan.server

サーバー側スクリプトを書くのに有用なライブラリ。CVS バックエンドの操作等が含まれる。

3.3.7 glim

`banyan.gui` モジュールを書く過程で、Gtk の API を直接記述するのは煩雑で重複も多く、開発効率と保守性の低下を招いていることに気づいた。Gtk は豊富な機能を持つが、多くの場合、その使い方はパターン化しており、抽象化ライブラリを書くことで開発の効率化と保守性の向上が見込まれる。そこで Gauche-gtk の上に Scheme で GLIM という抽象化レイヤを実装した。

ユーザインタフェースの抽象化に関しては実装例も多いが、ここでは仕様が十分に検討され実績もある Common Lisp Interface Manager (CLIM) の API を参考にすることにした。CLIM はユーザインタフェースのあらゆ

表 3: 実装されているノード。**R/U**: 'R' は renderable-node を、 'U' は updatable-node を示す。

クラス名	R/U	説明
形状に関するノード		
<shape>	R	形状に関するノードの抽象ベースクラス
<polygon-shape>	R	ポリゴンによる形状
<locator-shape>	R	Locator(編集時のマーク) に使われる形状
<connector-shape>	R	Connector(特殊な編集オブジェクト) に使われる形状
<parametric-shape>	R/U	パラメータから計算により生成される形状
<lsystem-shape>	R/U	L-System によって生成される形状
質感に関するノード		
<shader>	R	質感生成に関するノードの抽象ベースクラス
<gl-shader>	R	標準の OpenGL パイプラインによる質感生成を行うノード
<const-shader>	R	単一色でレンダリングするためのシェーダノード (主として編集時に使われる)
<material>	R	質感のパラメータを保持するノードの抽象ベースクラス
<gl-material>	R	標準の OpenGL シェーダで使われるパラメータ
<light>	U	光源に関するノードの抽象ベースクラス
<gl-light>	U	標準的な OpenGL のライティングモデルを実現するクラス
<light-group>	R	同時に使用される光源のグループを管理するクラス
位置に関するノード		
<transform>	U	階層的なトランスフォームを実現する抽象ベースクラス
<trs-transform>	U	移動、回転、拡大の組合せによる標準的なトランスフォーム
その他のノード		
<camera>	R	レンダリングの視点を指定するクラス
<instance>	R	transform, shape, shader をまとめるクラス。シーン内で「目に見える」ものは全て instance である。
<child-connector>	U	パートをまたがる接続を管理する特殊なノード。通常のトランスフォームの親子関係では子供が親へのポインタを保持しているので、パートとしては子供を含むパートが親のパートに依存することになる。しかし、親を含むパートが子供を含むパートに依存する方が管理しやすい場合も多い。コネクタノードはそのような依存管理を実現する。
<parent-connector>	R	コネクタノードの親側。

表 4: <part> の管理するメタデータ

名称	機能
name	パートの管理名。管理名はリポジトリの URI の後に付加してパートを一意に指定する URI を得るのに使われる。
imports	このパートが依存するパートのリスト
exports	このパート内のノードで、外部からの参照を許すノードのリスト。(デフォルトでは全てのノードの参照が許される)。
revision	現在のリビジョン番号
title	より人間に分かりやすい形のパートの名称。
description	パートの説明。
modified-by	パートを最後に変更したユーザ。
modification-time	パートが最後に変更された時刻。
created-by	パートを最初に制作したユーザ。
creation-time	パートが制作された時刻。
contributed-by	パート制作に協力したユーザのリスト。
derived-from	このパートが別の素材から作られた場合に、その素材の URI を示す。
cc-license	Creative Commons で定義されているライセンスを指定するシンボル。

る面をカバーしようとする巨大な仕様であるが、本件に必要なのは典型的な UI 部品 (ウィジェット) の抽象化が主体なので、以下の 3 点に絞って実装した。CLIM との完全互換を目指すものではないので、Scheme 風に API を変えた部分もある。

- Application Frame: GUI では「窓」に相当する、ユーザインタフェース中のセッションの単位。
- Pane: GUI 部品を抽象化したもの。
- Menu: メニューを抽象化したもの。

現在の実装は Gtk と強く結び付いているが、API 自体は GUI ツールキットには非依存である。将来は GUI だけではなく Web インタフェース等のバックエンドも付加して、独立した Gauche パッケージとして配付する予定である。

3.3.8 x3d

VRML へのエキスポート機能を実現するために実装したモジュール。X3D/VRML で定義されるシーングラフ構造を実現するクラス群が定義されている。VRML へのエキスポートは、Banyan のシーングラフを VRML のシーングラフへと変換した後に、VRML のグラフの write メソッドを呼ぶという方法で実現されている。

このモジュールも独立性が高いので、将来は別の Gauche パッケージとして配付する予定である。

3.3.9 banyan (client)

本システムのクライアントプロトタイプとして実装したプログラムである。起動時オプションにより、シーンの編集が可能なエディタ、あるいはシーンを見るだけのブラウザとして動作する。図 2 に動作中のクライアントプログラムのスクリーンショットを示す。

現在のエディタクライアントはプロトタイプレベルであり、以下に示す基本的な機能のみ実装されている。

- 定義済みのポリゴン形状の作成 (cube, cylinder, sphere, cone, plane) 及び、定義済みの procedural な形状の生成 (L-system)
- 特殊オブジェクトの作成 (locator, connector)
- オブジェクトのトランスフォームの変更 (translation, rotation, scale, transform freeze)
- トランスフォームの階層の変更 (parent, unparent)
- 属性エディタによる属性の編集
- ノードグラフの変更
- パート情報の編集
- リポジトリの選択とチェックイン・チェックアウト
- VRML へのエクスポート

3.3.10 banyan2vrml

このプログラムは banyan ライブラリを用いてノードグラフを VRML の形式へと変換する。オフラインでのグラフ変換のデモである。

3.3.11 repository (server)

サーバ側のアプリケーションは、実装と管理のしやすさを考えて、CGI を利用して作成した。クライアントのリクエストは httpd へと送られ、そこから CGI を経由して repository スクリプトへと渡される。repository はリプライを http のフォーマットに載せて httpd に返し、httpd からクライアントへと送り返される。この方法は多数のトランザクションを処理する場合はオーバヘッドが大きく不利であるが、本システムのようにひとつひとつのデータが大きいもののトランザクション量はそう多くならないことが期待される場合には、次のような利点がある。

1) http は比較的簡単なプロトコルであり実装が容易である。クライアントアプリケーションも書きやすい。
2) 少なくとも開発の初期段階で、頻繁にアップデートが行われる状況では常時起動しておくサーバより CGI の方が遥かに管理が容易である。また、開発が進んでソフトウエアが成熟した段階になってから常時起動式に移行するのも難しくない。
3) 同一プログラムで、通常の Web ブラウザからのアクセスと専用クライアントからのアクセスを処理できる。

第 3 の点に関しては、repository スクリプト内でリクエストの `Accept` ヘッダの MIME タイプを見て、それが `text/http` である場合は Body にメタデータを表現する HTML を、`application/x-banyan` の場合はノードグラフをシリアライズしたデータを返すことで実現している。従って、通常の Web ブラウザでパートの URI にアクセスすればメタデータをブラウズすることが出来る (図 3)。

シーンリポジトリのデータは Unix ファイルシステム上に展開されており、バックエンドには CVS を用いて履歴管理を行っている。

リポジトリ内でのノードグラフ変換の例としては、サーバ側での VRML への変換をサポートした。Web ブラウザでパートの URI にアクセスして「Export to VRML」のリンクをクリックすれば、VRML に変換されたシーンを web ブラウザのプラグインで見ることが出来る。

また、実験的にチェックインされたパートに関する意見交換を行える機能を追加してみた。この機能はテキストベースのコラボレーションツールとして一般化している Wiki を使って実現されている (実装には川合が作成、公開している Wiki エンジンである WiLiKi を用いている)。

3.4 テストと評価

本ソフトウェアの基本機能に関しては、サーバ側の実装が固まり始めた 2002 年 12 月後半より開発者の LAN 環境でサーバとクライアントを別マシンで継続的に運用し、テストを行って来た。フレームワークが動作すること、及びコンセプトを示すためのプロトタイプを作成するという点は達成出来たと考える。

未踏事業の契約はここで終了となるが、それは決して本ソフトウェアの開発の終了ではない。むしろ今回までの開発結果を公開し初期ユーザーを獲得するところから、次の段階の開発が始まると言える。オープンソース開発プロジェクトが軌道に乗るためには、ある程度まとまった出発点となるソフトウェアが必要である。本ソフトウェアは未踏事業開始時に全くのゼロから出発したが、胎生期を終えて、広く公開して開発を進められる段階に達したと考えている。

4 開発成果の特徴

製作過程のマテリアルを公開サーバに置き、多人数で協調して製作を進めるという手法は、ソフトウェア製作では既に常識となっており、テキストを主体としたコンテンツ制作でも Wiki システム等によりその有効性が示されつつある。従って、それが 3D CG コンテンツ分野でも有効であると考えるのは自然である。しかし、より複雑な 3D CG コンテンツに関しては制作過程にあるデータを共有、交換する仕組みが今まで無かった。

本ソフトウェアは 3D CG コンテンツ制作の分野で協同制作のアプローチを試すことができる基盤を実際に提供するという点で極めて有意義であると考えられる。実際に触れるものがあることで、ユーザからのフィードバックが得られ、そのような協同制作の形態を発展させ

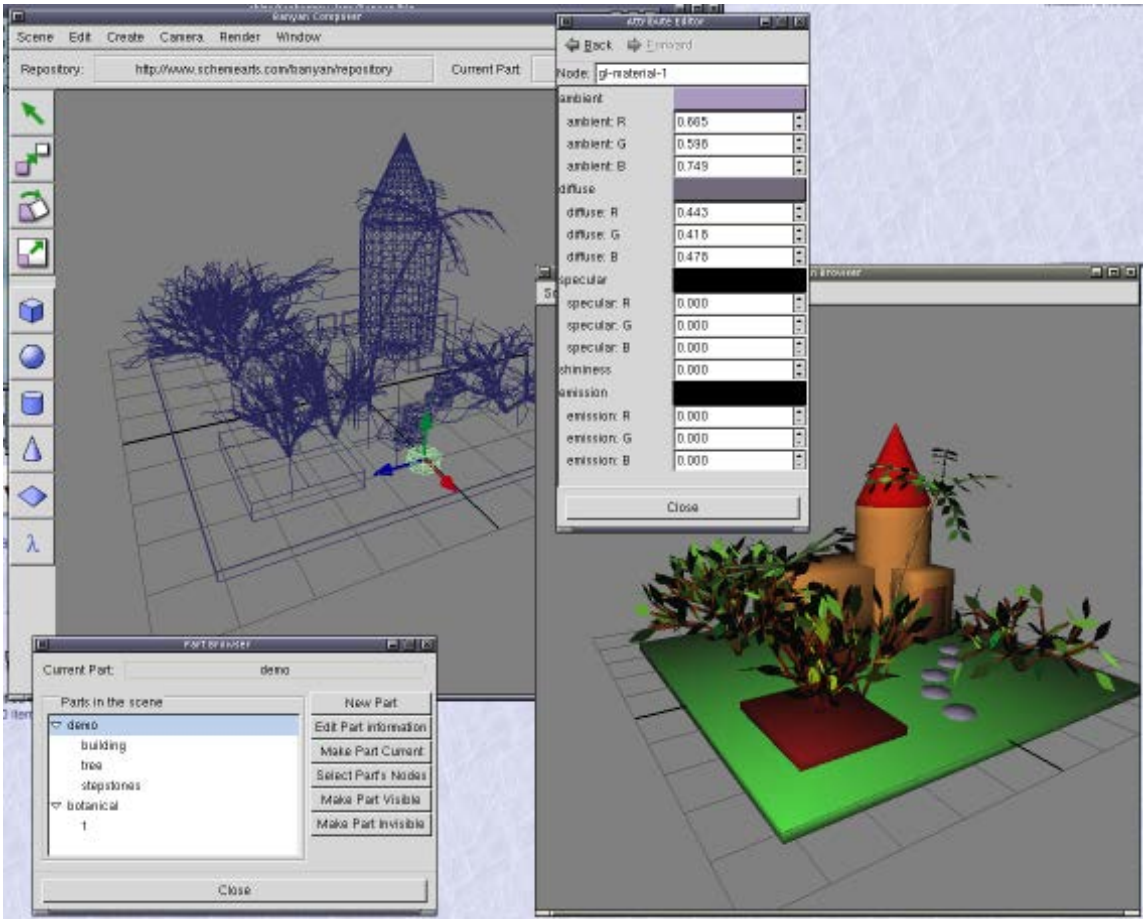

図 2: クライアントプログラムのスクリーンショット

てゆけるであろう。

また、ネット上に制作物を公開する場合に、コンテンツの二次利用をいかに制限するか点が議論になることが多い。しかし、制作者全てが必ずしもコンテンツの再利用を望んでいないわけではない。むしろ、無償で公開して改良・再配付を許すオープンソースソフトウェアの文化と同様に、コンテンツ制作者も著作権を保留した上で、コンテンツを共有して二次利用を推奨しようという考えが広まりつつある。Creative Commons はその考え方に基づき、いくつかのライセンス形態を定義している。本システムではメタ データに Creative Commons で定義するライセンスを持たせることで、積極的にオープンコンテンツ制作を推奨してゆく基盤ともなるであろう。

5 今後の課題、展望

まず、プロトタイプと共有リポジトリを公開することで、ユーザーを獲得することが最初のステップとなる。初期ユーザーからのフィードバックによってソフトウェアは改善してゆけるであろう。

クライアントとしてのシーンエディタが実用段階に達するにはまだ多くの時間と開発が必要である。それとは別に、既存のオーサリングシステムのプラグインを開発することで、より広いユーザーベースが獲得できるであろう。具体的にどのようなシステムのプラグインを開発するかはユーザーの要望による。開発者としては全てオープンソースのシステムとして実装できるものを望んでいるが、実用的には商用システムへのプラグインも必要となるであろう。

サーバの機能としては、実用段階に至るまでに以下の点を解決すべきであると考えている。これらは今後共有リポジトリサイトを運用してゆく過程で開発を進めて行く。

- データ転送の最適化。3D データは大きくなりがちなので、まずはデータを圧縮して転送する機能をつける。
- ノードグラフの差分検出。二つのグラフを比較して変化分を抽出する。これによって、(1) クライアントがチェックインの際に変更部分だけを送る最適化が可能となり、また (2) 変更が衝突した場合にある程度の自動マージが可能になる。

また、提案書にオプショナルな機能として含めてあったが今回実現できなかった機能として、ノードグラフをコンパイルするというアイディアがある。プレイバックの最適化やプロプライエタリなコンテンツの配信に応用できる技術であるので、その点に関しても開発を進めてゆく予定である。

6 実施計画書内容との相違点

実施計画書では、クライアントの機能としてモデリング機能の他にアニメーション機能を実装することとしていた。しかし、クライアント部分に関して豊富な機能を求めるには開発資源が足らず、中途半端に実装するよりコンセプトを示す方が良いと考え、PM の了承も得てアニメーション機能は割愛した。そのかわり、本ライブラリを拡張して procedural modeling を実装できるという

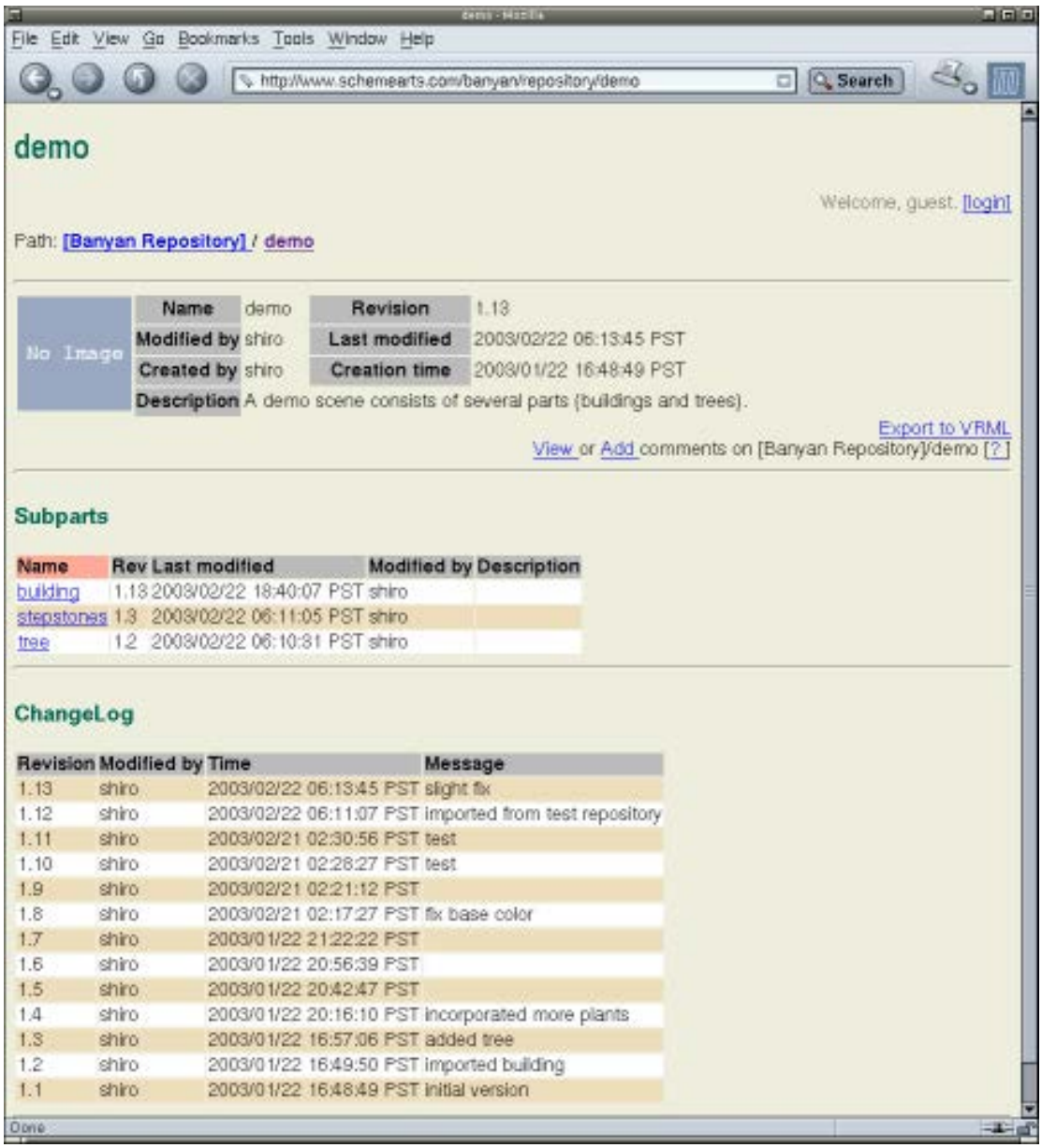

図 3: サーバーに Web ブラウザでアクセスした場合の画面

サンプルとして L-System の実装を行なった。また、実施計画書には具体的に示していなかったが、サーバ側でのデータの別形態でのチェックアウトの例として VRML への変換を実装した。

7 参加企業及び機関

本開発では、日本エンジェルス・インベストメント株式会社にプロジェクト実施管理組織となって頂いた。

参考文献

[1] Shiro Kawai. Shooting A Moving Target— An Experience In Developing A Production Tracking Database, *in Proceedings of Japan Lisp User Group Meeting (JLUGM) 2000, Tokyo, Japan, May 2000.*

[2] Shiro Kawai. Tracking Assets in the Production of 'Final Fantasy : The Spirits Within', *in Proceedings of Game Developers Conference 2002, San Jose, CA, March 2002.*

[3] Shiro Kawai, Gluing Things Together - Scheme in the Real-time CG Content Production, *presented at International Lisp Conference 2002, San Francisco, CA, October 2002.*