Модульный подход к построению визуализаторов для компьютерных моделей сложных систем

С.В. Иванов, А.А. Безгодов

НИИ Наукоемких компьютерных технологий Санкт-Петербургского государственного университета информационных технологий, механики и оптики
svivanov@mail.ifmo.ru, demiurghg@gmail.com

Введение

Визуализация результатов компьютерного моделирования является важной частью научного исследования. Современные компьютерные модели генерируют огромное количество числовых данных, для наглядного представления и анализа которых требуются специальные подходы. Ситуация еще больше усложняется, когда возникает необходимость в визуализации результатов моделирования сложных систем, характерными чертами которых является большое число компонентов, сложные связи между ними и наличие многомасштабной (пространственной, временной, межкомпонентной и пр.) изменчивости [1]. В результате генерируемые данные требуют больших объемов оперативной памяти для хранения соответствующих структур данных, а их визуализация демонстрирует эволюцию тех или иных параметров системы во времени, что подразумевает динамические приемы визуализации. Кроме того, сложные системы характеризуются иерархией моделей, описывающих поведение системы в различных диапазонах изменчивости, что порождает разнообразие входов и выходов сложной системы как целого. Важно отметить, что модели сложных систем являются предметно ориентированными (например, климатические системы, сложные транспортные потоки, эпидемиологические модели взаимодействующих агентов и т.п.), что не всегда позволяет использовать визуализаторы общего назначения, которые поддерживают только ограниченный набор форматов данных и не могут обеспечить нужные форматы под все компоненты сложной системы. Таким образом, оптимальный подход к построению визуализаторов для компьютерных моделей сложных систем заключается в модульном подходе, основанном на отображении структуры модели сложной системы на программную архитектуру визуализатора, и использовании графа зависимостей для более гибкой настройки источников данных и средств их отображения, быстрого создания дополнений и расширения функциональности визуализатора.

Описание графа зависимостей

Визуализатор строится по схеме так называемого графа зависимостей (dependency graph) [2]. Данный подход позволяет гибко настраивать визуализатор и создавать дополнения, расширяя его функциональность. Такая схема нередко используется в 3D-редакторах (например, Autodesk Maya), звуковых редакторах, а также в системах визуального программирования.

Граф состоит из узлов (node) и связей (connection) между ними. Каждый узел имеет несколько регистров (register). Регистры хранят значения различных типов. Регистры можно произвольно добавлять и удалять посредством соответствующих вызовов. Поддерживаются следующие типы данных

1. Булево значение.
2. Сигнал.
3. Целое число.
4. Вещественное число.
5. 4-х мерный вектор (3-х мерный вектор в однородных координатах).
6. Строка.
7. Матрица 4x4.

8. Двумерное поле (скалярное, векторное, 8-,16-,32-бита на компоненту), фактически – аппаратная текстура.

Путем написания модулей, можно добавлять к системе новые типы данных. Каждый новый тип должен реализовывать как минимум две операции: сериализацию данных в XML и передачу между регистрами.

Регистры одних узлов могут быть подключены к регистрам других узлов, при условии, что типы данных регистров совпадают. Таким образом, по установленным связям во время вычисления (evaluation) графа (расчета зависимых величин) передаются данные. Циклические зависимости запрещены. Регистры могут быть в режиме Input, Output или InputOutput.

С каждым узлом графа может быть сопоставлена функция, которая:

1. Вычисляет значения выходных регистров при изменении значений на входных регистрах. Во избежание лишних вычислений устанавливается влияние (affection) одних регистров на другие (affected регистры становятся доступными только для чтения, т.е. Output).
2. Выполняет определенное действие на каждом цикле функционирования системы.

Вычисление (evaluation) графа осуществляется следующим образом:

1. Определяются регистры, значения которых изменились с момента предыдущего вычисления графа.
2. Регистры и узлы топологически сортируются.
3. Последовательно, для каждого входного регистра, данные в котором изменились, выбираются все выходные регистры, которые зависят от данного входного регистра и он помечается как требующий перерасчета.
4. Для каждого выходного регистра, который требует перерасчета вычисляется значение и использованием функции заданной в узле. Полученные данные передаются во все регистры, которые были подключены к данному выходному регистру.

Следует отметить, что если значение не изменялось, или был получен тот же результат, то лишние вычисления не производятся. Так, например, узел, читающий файлы с диска, читает его и формирует поле только при изменении входного регистра, который задает имя этого файла.

Конфигурация графа и значения данных могут быть записаны и прочитаны с использованием XML. Разрабатываемая система поддерживает командный интерфейс, и позволяет конструировать граф путем подачи команд, как локально, так и удаленно.

Также следует отметить, что система утилизирует аппаратные возможности графических ускорителей с целью обработки скалярных и векторных данных.

Пример функционирования графа зависимостей для задачи обработки и визуализации гидрометеорологических данных

Как было уже отмечено выше, задача визуализации гидрометеорологических данных может требовать сложного представления различных величин и преобразования данных из различных источников, что подразумевает:

1. Наложение различных данных друг на друга (наложение уровня волн и направления ветра).
2. Сложные конфигурации видов (например, слева – вид сбоку, справа – вид сверху).
3. Анимацию процессов (по возможности в реальном времени).
4. Чтение из разнообразных источников данных (файлы на диске, локальные и удаленные базы данных).
5. Дополнительную обработку (нахождение градиентов, фильтрацию)

6. Поиск тех или иных значений или точек (по уровням, по координатам)
7. Расстановку маркеров (например, названия и координаты станций наблюдения)

Все перечисленные задачи могут быть решены с помощью предложенной схемы.

Рассмотрим задачу, в которой необходимо:

1. Считывать данные из файлов, которые расположены на диске (в алфавитном порядке).
2. Осуществлять плавную интерполяцию между «кадрами» данных.
3. Выполнять визуализацию скалярных данных в виде изолиний, при этом автоматически подбирая максимальное и минимальное значения.
4. Выполнять расчет градиента скалярного поля и визуализировать полученное векторное поле.

Для этого вводятся следующие узлы:

1. Таймер – необходим для анимации.
2. Файловый итератор – перебирает файлы в указанном каталоге.
3. Линейный интерполятор – выполняет интерполяцию между двумя наборами скалярных полей.
4. Фильтр Гаусса – необходим для получения сглаженных изолиний.
5. Анализатор поля – выполняет различные операции, в том числе поиск минимальных и максимальных значений.
6. Оператор градиента.
7. Векторный и скалярный визуализатор.

Пример соединения узлов графа показан на рис. 1.

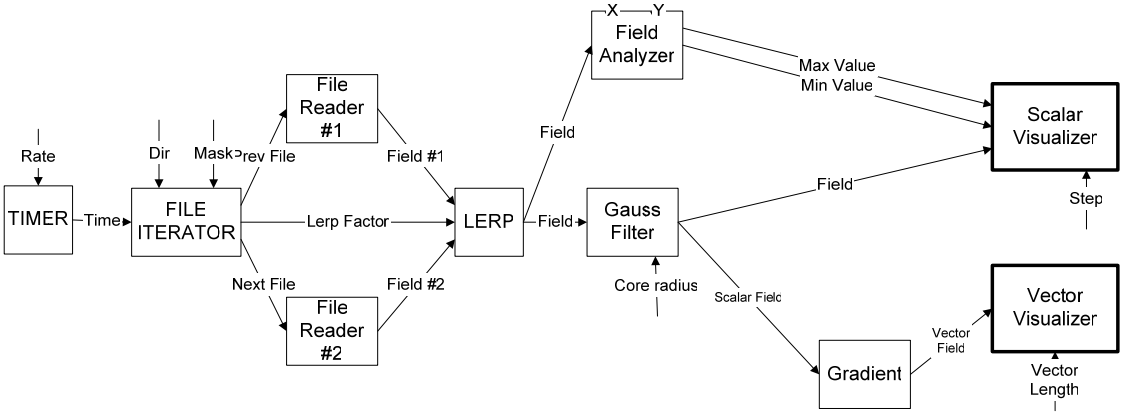

Рисунок 1. Соединение узлов графа для задачи визуализации серии скалярных полей, расположенных на диске.

Таймер (TIMER) непрерывно увеличивает значение time на выходе (число с плавающей запятой). Данное значение поступает на файловый итератор (File Iterator). Файловый итератор анализирует содержимое каталога Dir, и выбирает имена файлов под номерами n и n+1, где n – округленное вниз значение time, и передает их считывателям файлов (File Reader #1 и File Reader #2). Дробная часть значения time поступает на выход lerp factor. Файловые считыватели считывают файлы и формируют на выходе скалярное поле.

Поля подаются на линейный интерполятор (Lerp). Результат интерполяции подается на анализатор поля (Field analyzer) и фильтр Гаусса (Gauss filter). Последний нужен для сглаженного представления изолиний. Поле подается на визуализатор скалярного поля (Scalar Visualizer). Также это поле подается на узел, который считает градиент

поля (Gradient), который в свою очередь передает поле на визуализатор векторного поля (Vector visualizer).

Анализатор поля находит минимальное и максимальное значения и подает эти значения в визуализатор скалярного поля. Это необходимо, для того, чтобы все значения всегда находились в заданном диапазоне.

Заключение

Предлагаемый подход к построению визуализаторов для моделей сложных систем позволяет создавать гибкие и легко расширяемые системы, функционирующие по достаточно простым принципам.

Необходимо развитие системы в следующих направлениях:

1. Добавление новых узлов и типов данных (поля векторных величин, пространственно-временные поля, геометрические данные).
2. Расширение существующих узлов (функциональные преобразования полей, фильтры и т.п.).
3. Создание развитого интерфейса удаленного управления визуализатором.

Литература

1. Bar-Yam Y. (1997), Dynamics of Complex Systems // Westview Press, 864 pp.
2. Balmas, Francoise (2001) Displaying dependence graphs: a hierarchical approach, WCRE, p. 261 // Eighth Working Conference on Reverse Engineering (WCRE'01)