О подсистеме истории в среде научной визуализации SharpEye

П.А.Васёв[1], С.С.Кумков[1,2], Е.Ю.Шмаков[2]

[1] Институт математики и механики им. Н.Н.Красовского, УрО РАН, Екатеринбург

[2] Уральский Федеральный университет, Екатеринбург

{pavel.vasev , sskumk , schmak.ekb}@gmail.com

Аннотация

В докладе рассматривается разрабатываемая авторами модульная среда научной визуализации SharpEye. Обсуждаются пути реализации подсистемы редактируемой истории, в возможности которой должны входить функции отката и повтора манипуляций, проделанных пользователем, сохранение и восстановлении подобранного вида сцены.

Ключевые слова: *научная визуализация, система визуализации, подключаемые внешние модули, редактируемая истории, откат/повтор действий*

1. ВВЕДЕНИЕ

В течение последних лет авторы разрабатывают среду SharpEye – конструктор систем научной визуализации [3,4]. В основу среды положен модульный принцип: сама среда не содержит никаких алгоритмов восстановления геометрических объектов по данным пользователя, а лишь предоставляет программный интерфейс (API) для наполнения сцены и средства манипулирования ею. При этом подразумевается, что для каждой конкретной задачи или класса задач пользователь самостоятельно создаёт процедуру восстановления геометрического образа визуализируемых данных и оформляет её в виде внешнего модуля, подключаемого системой при запуске и используемого при загрузке соответствующих файлов данных. Такой подход основывается на идее об уникальности визуализируемых особенностей для каждого класса задач и, как следствие, специфичности процедур восстановления геометрических образов.

Существующие системы визуализации (например, ParaView) зачастую всё-таки идут по пути включения в свой состав более или менее широкого набора готовых процедур, а на случай обработки нестандартных данных пользователю предоставляется возможность формирования сцены в файле оговоренного формата. Подход, предложенный авторами, представляется более гибким, так как помимо собственно процедуры восстановления программный модуль, разработанный пользователем, может содержать и другие процедуры, например, анимации созданных объектов в случае визуализации динамических процессов или специфических элементов управления.

Основной целью пользователя при работе с системой визуализации является подбор такой точки зрения (расположения и ориентации объектов перед камерой, их визуальных атрибутов, режима освещённости и т. д.), чтобы наиболее чётко были видны визуализируемые особенности. Разрабатываемая система предоставляет соответствующий функционал управления сценой и объектами как через пользовательский интерфейс (GUI), так и через программный.

В процессе ручного подбора точки зрения зачастую оказывается, что в результате действия или набора действий, проделанных пользователем, наглядность получаемой картинки ухудшилась. Поэтому удобно было бы отменить проделанные действия и попробовать другие направления улучшения наглядности изображения. Соответственно, система должна предоставлять пользователю функционал редактируемой истории [1,2].

Но реализация редактируемой истории достаточно нетривиальна. Не слишком сложно сохранять и восстанавливать стандартные параметры объектов и сцены (текущая камера, её пространственное и угловое положение; параметры и положения источников света; цвет, прозрачность, видимость отдельных объектов в сцене). Однако система позволяет пользователю задавать собственные свойства объектов и управлять ими. При этом их тип и методика работы с ними системе априори неизвестны — соответствующий функционал полностью заключён в пользовательских модулях. Это и объясняет сложность реализации данной подсистемы. Кроме того, непросто спроектировать взаимодействие подсистемы истории и подсистемы анимации, поскольку действие последней заключается в значительном количестве изменений параметров сцены и/или объектов, осуществляемых через программный, а не пользовательский интерфейс.

Также запись действий пользователя может быть сохранена в файл и в дальнейшем повторена, что даёт возможность сохранения удачно подобранной точки зрения не в виде статичной растровой картинки, а именно в виде состояния сцены.

В статье обсуждается реализация подсистемы истории, преодолевающей указанные проблемы.

2. ЭЛЕМЕНТЫ ИСТОРИИ

При разработке системы авторы рассмотрели два подхода к реализации подсистемы истории.

Первый подход подразумевает, что перед выполнением любой операции, которая может повлиять на состояние сцены, система подаёт команду менеджеру истории на сохранение полного снимка сцены (*независимый объект истории*). В данный снимок входит информация о местоположении источников света и их свойствах, информация о расположении и направленности камер, а также значения, которые принимают стандартные и пользовательские свойства моделей.

Преимуществом данного подхода является простота реализации операций отмены и повтора: необходимо лишь восстановить сохранённые в снимке объекты и установить их свойства в соответствии со снимком. При этом сам процесс сохранения снимка не зависит от типа операции. Однако

снимок системы может быть достаточно большим, если сцена содержит много объектов. Как следствие, это может вызвать проблемы как с хранением снимков в оперативной памяти, так и с сохранением их на диске. Кроме того стоит заметить, что если мы храним историю в виде набора снимков, то б***о***льшая часть информации в каждом из них избыточна, поскольку каждая из операций, как правило, изменяет лишь малую часть параметров сцены, что приводит к тому, что соседние в истории снимки отличаются незначительно. Данное свойство является недостатком первого подхода и приводит к мысли о том, что можно хранить только изменения сцены (*инкрементальный объект истории*).

Во втором варианте реализации подсистемы истории для каждой операции сохраняются только те параметры сцены, которые были изменены. При этом удобно сопоставить с каждой операцией отдельный класс для сохранения изменяемых данных. Классы элементов истории для разных операций реализуют общий интерфейс, декларирующий всего два метода: `Undo` и `Redo`. Первый метод позволяет отменить операцию, а второй повторно ее применить, если она была отменена ранее. Например, для операции «поворот камеры» можно создать класс, имеющий несколько внутренних полей: «имя камеры», «угол поворота» и «ось вращения». Тогда метод `Redo` данного класса будет поворачивать заданную камеру относительно текущего положения на заданный угол, а метод `Undo` — на угол, ему противоположный.

Преимущества данного варианта очевидны: сохраняется минимум данных, необходимых для отката операции; каждый класс элемента истории связан с конкретным преобразованием сцены, что позволяет отображать в пользовательском интерфейсе текст элемента в виде, понятном пользователю, например, для приведённого выше примера текст может выглядеть так «Поворот камеры 'имя камеры' на угол 'угол поворота'». Однако данный подход требует очень аккуратной реализации. Например, такой метод хранения поворотов камеры будет накапливать погрешность: если произвести несколько поворотов камеры, а потом их откатить, то камера может не вернуться в исходное состояние, а быть повёрнутой на некоторый небольшой угол, который возникнет из-за погрешности вычислений чисел с плавающей запятой. Чтобы избежать данной проблемы можно сохранять не «угол поворота» и «ось вращения», а начальное и конечное положение вектора направления камеры и вектора верха камеры. При таком подходе выполнение отката или повтора операции не повлечёт никаких операций с числами с плавающей запятой, кроме присваивания, таким образом, ценой небольшого увеличения сохраняемых данных проблема накопления погрешности может быть решена.

Существенным недостатком данного подхода является то, что для каждой новой операции над сценой приходится писать код, позволяющий сохранять данные, относящиеся к данному преобразованию, в отличие от первого подхода, где всегда сохраняются одни и те же данные вне зависимости от операции.

Общим у описанных подходов является то, что при произведении действия система сохраняет некий объект, так или иначе поддерживающий действия `Undo` и `Redo`. При этом `Undo` подразумевает, что система находится в состоянии, имевшемся сразу после совершения действия, и переводит её в состояние, имевшееся непосредственно перед свершением этого действия. `Redo` производит прямой перевод.

Для удобства использования разумно потребовать от подсистемы истории, чтобы она группировала несколько последовательных однотипных действий в одно суммарное. Например, несколько последовательных поворотов сцены могут быть заменены одним итоговым поворотом. Или несколько последовательных изменений цвета одного и того же объекта могут быть сгруппированы в изменение исходного цвета на конечный. Для этого объект элемента истории может поддерживать метод `Merge`, который, возможно, соединяет его с другим элементом истории, передаваемым в этот метод.

Конкретная реализация элемента истории для того или иного действия обуславливается сущностью этого действия. В частности, если пользователь в рамках модуля работы со своими объектами предусматривает введение нестандартных свойств и работу с ними, то соответствующие обработчики могут взаимодействовать с менеджером истории, поставляя ему объекты истории работы с этими свойствами.

3. ТРАНЗАКЦИИ В ИСТОРИИ

Некоторые действия пользователя могут разворачиваться в значительные наборы простейших операций. Например, загрузка файла данных реально приводит к созданию модели, наполнению её геометрии, установке свойств цвета, прозрачности, видимости; тут же возможно создание, редактирование и/или удаление камер и/или источников света и т. д. Соответственно, такие наборы простейших действий должны откатываться и повторно выполняться не по одному, а в совокупности.

Это размышление приводит к необходимости создания *транзакций*, специальных объектов истории, задачей которых является объединение набора тривиальных действий в одно составное. При этом методы `Undo` и `Redo` транзацкции вызывают в соответствующем порядке методы `Undo` и `Redo` объектов, объединённых в транзакцию.

4. СОХРАНЕНИЕ И ЗАГРУЗКА СЕССИИ

Объект истории может автоматизированно поддерживать сохранение себя в указанный поток и чтение себя из указанного потока. Соответственно, для этого предусмотрены два метода: `Serialize` и `Deserialize`. В случае, когда запись объектом поддерживается, решается несколько задач.

Во-первых, сохранение на диске удачно подобранного состояния сцены с целью его дальнейшего быстрого восстановления. Во-вторых, на этом принципе может быть построена подсистема удалённой визуализации: на стороне пользователя интерфейсная часть программы генерирует действия и передаёт их в канал связи с серверной частью. На второй стороне канала серверная часть производит считывание объектов истории и применяет их к сцене, высылая пользователю получаемую картинку.

Данная возможность может быть полезна, как при личном использовании системы, так и при коллективном. При личном использовании сохранение истории изменений позволяет продолжить сеанс работы с того места, на котором он был прерван. При коллективном же использовании данная возможность становится удобным инструментом для

публикации результатов, поскольку результат представляется не в виде статичного изображения, а поставляется вместе с системой визуализации, позволяющей рассмотреть его во всех подробностях.

5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Система SharpEye реализована на языке C# для среды исполнения Microsoft .Net 4.0 и выше. Для доступа к своему API среда SharpEye предоставляет библиотеку `Contracts`, в которой определены интерфейсы всех классов, доступ к которым может быть полезен для организации работы пользовательского модуля со средой. Авторами разработана документация к этой библиотеке (и, фактически, справочник для программиста).

В частности, в этой библиотеке определён интерфейс `IHistoryElement`, в котором определены методы `Undo`, `Redo`, `Merge`, `Serialize`, `Deserialize` и свойство `IsIndependent`. Последнее свойство позволяет менеджеру истории оптимизировать процессы отката и повтора пользовательских действий: если откатывается или повторяется указанный пользователем набор действий, то можно не производить часть или все промежуточные инкрементальные изменения, а сразу установить состояние сцены из независимого объекта (если таковой присутствует в этом наборе) и продолжить применение инкрементальных поправок от этого состояния.

Внутри самой системы SharpEye наличествует достаточное количество классов, производных от интерфейса `IHistoryElement`, которые обслуживают запись элементов истории для стандартных действий.

Следует отметить, что доступ в система SharpEye может быть получен не только в рамках программ, написанных для среды .Net, но и в скриптах, написанных на языке Ruby. В систему интегрирован интерпретатор IronRuby. Поэтому при реализации пользовательского объекта типа `IHistoryElement` методы `Redo` и `Merge` могут представлять собой скрипты на языке Ruby, в том числе, генерируемые системой.

Программный доступ к сцене обеспечивается через ссылку на объект типа `IStorage`, реализующий интерфейс хранилища сцены. (Такая ссылка передается модулю пользователя при его инициализации.) Этот объект, в частности, имеет свойство, возвращающее ссылку на объект, реализующий интерфейс `IHistoryManager`. Объект такого типа предоставляет средства для работы с историей:

- `UndoLast` — откат последнего совершённого действия;
- `RedoLast` — повтор последнего отменённого действия;
- `UndoUpTo`, `RedoUpTo` — откат и повтор группы действий до указанного элемента или до конца записанной истории;
- `AddElement` — добавление элемента истории в конец имеющейся записи; если имелась запись отменённых действий, она теряется;
- `OpenTransaction`, `CloseTransaction` — начать/закончить транзакцию; при этом все элементы истории, добавляемые в процессе формирования транзакции, автоматически добавляются в элемент транзакции;
- `SaveSession`, `LoadSession` — запись и чтение истории в/из заданного потока.

В объекте хранилища `IStorage` предусмотрены методы `MakeRestorationPoint` и `ApplyRestorationPoint`. Первый из них является открытым и создаёт снимок сцены, возвращая его в независимом объекте истории; второй, закрытый — принимает независимый объект истории и переводит сцену в соответствующее состояние.

6. ВЗАИМОДЕЙСТВИЕ ПОДСИСТЕМ ИСТОРИИ И АНИМАЦИИ

Проблема с анимацией заключается в том, что для «оживления» изображения в сцене производится множество изменений: либо перемещается камера, либо меняется какое-либо свойство того или иного объекта, либо меняются свойства у большого количества объектов (например, одни объекты делаются невидимыми, а другие «включаются»). Если делать эти изменения инкрементальными, то существует опасность накопления ошибки, отмеченная в разделе 2. Поэтому объекты системы, ответственные за управление анимацией, делают снимки системы, соответствующие началу и концу анимации для быстрого перевода системы в эти состояния. Перевод системы в состояния, соответствующие промежуточным моментам анимации, вообще говоря, возможен только при проигрывании анимации с начала (если пользователь специально не организует управление своей анимацией иным образом).

Однако в процессе проигрывания анимации пользователь может дополнительно преобразовывать сцену (поворачивать, перемещать и т. д.) для того, чтобы более чётко рассматривать динамику изменения тех или иных частей сцены. В настоящее время система SharpEye реализована таким образом, что пользовательские действия, производимые во время анимации, в историю не записываются и теряются при перемотке анимации в начало или в конец. При этом необходимый режим функционирования менеджера истории обеспечивается вызовом его соответствующих методов в начале и конце проигрывания анимации.

7. ЗАКЛЮЧЕНИЕ

В рамках среды – конструктора систем визуализации SharpEye, разрабатываемой авторами доклада, создана подсистема записи, отката и повтора действий пользователя. Данная система позволяет отменять неудачные изменения точки зрения, а также сохранять и восстанавливать сессию работы с системой для быстрого восстановления полученного вида отображения. В рамках технологии сохранения и восстановления сцены также может быть реализована удалённая визуализация.

При проектировании и разработке подсистемы истории авторам пришлось решить ряд проблем идеологического и программного характера (в том числе, для оптимизации работы данной подсистемы). В настоящее время подсистема истории полностью реализована и функциональна в соответствии с изложенными требованиями, позволяющими

удобно использовать её как через пользовательский, так и через программный интерфейс.

8. СПИСОК ЛИТЕРАТУРЫ

[1] Cypher A. (Ed.), Watch What I Do. Programming by Demonstration. MIT Press, Cambridge, MA. 1993.

[2] Авербух В.Л., Визуализация программного обеспечения. Конспект лекций. Екатеринбург: мат.-мех. ф-т, УрГУ. 1995.

[3] Васёв П.А., Кумков С.С., Шмаков Е.Ю., Конструктор специализированных систем визуализации // Научная визуализация, 2012, Т. 4, № 2, С. 64–77. МИФИ. ISSN 2079-3537. http://sv-journal.com/2012-2/05.php?lang=ru.

[4] Васёв П.А., Кумков С.С., Шмаков Е.Ю., О создании среды разработки систем научной визуализации // Труды XIII Международного семинара «Супервычисления и математическое моделирование» (3–7 октября 2011 г.) под редакцией Р.М. Шагалиева. — ИПК ФГУП «РФЯЦ-ВНИИЭФ», г. Саров, 2012. С. 131–140.