УДК 681.3.06

Б. С. Долговесов, **Б. С. Мазурок**, **В. Г. Ванданов**

Институт автоматики и электрометрии СО РАН
пр. Акад. Коптюга, 1, Новосибирск, 630090, Россия
E-mail: bsd@iae.nsk.su

ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ БАЗА ДАННЫХ В ИНТЕРАКТИВНЫХ СИСТЕМАХ 3D ВИЗУАЛИЗАЦИИ

Исходя из общих принципов оптимального построения систем выбрана архитектура для организации интерактивной системы 3D визуализации сцен в реальном времени на основе ускоренного обмена данными с глобальной объектно-ориентированной иерархической базой данных. Для унифицированного межмодульного обмена данными используется специально разработанный язык запросов – XQL (eXtended Query Language).

Ключевые слова: объектно-ориентированная база данных, XQL, система 3D визуализации.

Введение

Кривая роста производительности процессоров уже несколько десятилетий экспоненциальна – по «закону» Мура удвоение производительности происходит каждые 18 месяцев [1]. Если раньше основное внимание фокусировалось на эффективности, ограничении используемых ресурсов, экономии, производительности, авторских монолитных программах и математических моделях, то сегодня на первый план выходят устойчивость, гибкость, адаптируемость, распределенность вычислений и программирование.

С другой стороны, приложения, требующие высокой производительности, будут существовать всегда, например скорость, с которой необходимо обрабатывать видеоизображения, напрямую зависит от скорости, с которой машина способна его генерировать. По мере возрастания производительности будет возрастать качество и детализация видео – переход на стандарты повышенной четкости HD, Super HD и пр. Другими словами, существует класс задач, которые по определению обладают неограниченной способностью поглощать все доступные ресурсы: визуализация, криптография, моделирование.

В связи с этим задачу построения оптимальной архитектуры системы визуализации можно переформулировать как задачу о внедрении расширенного контроля за высокоскоростными процессами визуализации. На фоне огромных затрат на обработку потоков, визуализацию и пр. затраты на гибкость и устойчивость, которые раньше считались расточительной тратой ресурсов, теперь становятся вполне приемлемыми и даже необходимыми.

Предыдущие разработки авторов (системы визуализации *Аксай*, *Альбатрос* [1; 2], *Ариус* [3] и др.) осуществлялись в условиях ограниченной производительности процессоров, поэтому основной задачей было добиться максимальной отдачи от оборудования и получить максимальное количество граней, максимальный поток текстур и т. д.

В новой системе визуализации [4] предпринята попытка сбалансировать расширенный контроль и высокую производительность. Главным элементом этой системы является база данных (БД), которая используется для хранения информации 3D сцены и ее изменения во времени. В настоящей

[1] *Graham P.* The Hundred-Year Language. URL: http://www.paulgraham.com/hundred.html

статье представлены обоснование и более детальное описание основного элемента системы – базы данных реального времени.

Принципы построения системы

По мере роста любого проекта взаимная зависимость классов и объектов непрерывно усложняется, и если не предпринимать специальных мер, то проект может постепенно приблизиться к порогу, после которого код становится трудно управляемым, скорость разработки падает, проект не поддается изменениям, и у разработчиков возникает желание переделать все заново. Для того чтобы избежать этих проблем, целесообразно следовать основополагающим принципам, соблюдение которых позволит выбрать наилучшую архитектуру.

Будем исходить из следующих принципов эффективной организации программных модулей [2]:

- функциональная независимость (*Functional Independence*) – модули не должны интересоваться данными из других модулей;
- высокое сцепление (*High Cohesion*) – код, ответственный за какую-то одну функциональность, должен быть сосредоточен в одном месте;
- низкая связанность (*Low Coupling*) – модули должны иметь минимальные зависимости от других модулей.

Первые два принципа дают рекомендацию об организации внутренней структуры каждого программного модуля, но не касаются его окружения. Третий же принцип – связанность – позволяет сделать вывод об оптимальном способе организации модулей в *систему*. Основываясь на метрике, оценивающей степень связанности *Coupling* [3], можно показать, что наиболее эффективный способ организации модулей – это так называемое *звездное* соединение, когда все или большинство модулей взаимодействуют только с одним управляющим объектом и не связаны между собой прямыми перекрестными ссылками. Исходя из этого рассмотрим возможность введения *звездного* соединения как основу организации всей системы.

Основной обмен данными предлагается делать через управляющий объект, который будем называть *звездным.* Прямые межмодульные связи предлагается минимизировать так, чтобы каждый модуль все данные для работы либо хранил внутри себя, либо получал от *звездного* объекта. Такая организация приводит к высокой модульной независимости и высокому сцеплению кода *Cohesion* – высокой концентрации функциональности.

Поскольку на *звездный* объект возлагается задача распределения данных для *всех* модулей, то протокол обмена должен быть максимально унифицирован. Унификация обмена позволяет распределить работы по созданию программного обеспечения, независимо отлаживать отдельные модули и комбинировать модули для получения различных конфигураций системы. Кроме того, унификация обмена повышает взаимозаменяемость модулей и, как следствие, существенно упрощает тестирование.

Рассмотрим возможность применения одной из *стандартных* БД в качестве управляющего *звездного* объекта.

Реляционные базы данных

Для максимально эффективного использования *звездного* соединения, необходимо обеспечить высокую пропускную способность связей и высокую скорость обработки событий. Поэтому решающими требованиями к БД становятся высокая производительность и возможность обработки запросов в реальном времени.

Этим требованиям частично удовлетворяют объектно-ориентированные реляционные СУБД (OO-RDBMS), основная задача которых – отображение классов в таблицы БД, доступные стандартным SQL запросам [4]. Назовем основные объектно-ориентированные СУБД.

Illustra Server – первая в мире объектно-реляционная СУБД [5]. Сервер поддерживает объектно-ориентированное управление слож-

[2] См.: *Pressman R.* Software Engineering – A Practitioner's Approach.

[3] Там же.

[4] См.: *Codd E. F.* A Relational Model of Data for Large Shared Data Banks.

[5] См.: *Гвоздев А.* Архитектура и функциональные возможности объектно-реляционной СУБД Illustra.

ными типами данных, предоставляет эффективный язык запросов, основанный на расширении индустриального стандарта SQL, работает как единый репозиторий объектов.

ObjectStore – СУБД расширяет виртуальную память операционной системы [6]. *ObjectStore* поддерживает транзакции, допускает методы доступа: хеш-таблица для несортированных данных и B-дерево для сортированных, также возможно использование SQL.

Стандарт ODMG (Object Data Management Group) – консорциум поставщиков объектно-ориентированных баз данных (ООБД) [7]. Стандарт на хранение объектов ODMG разработан на основе трех существующих стандартов: управление базами данных (SQL), объекты (стандарты OMG – Object Management Group) и стандарты на объектно-ориентированные языки программирования (C++, Smalltalk, Java). ODMG добавляет возможности взаимодействия с БД в объектно-ориентированные языки программирования: определяются средства долговременного хранения объектов и расширяется семантика языка, вносятся *операторы* управления данными.

Перечисленные СУБД – реляционные, т. е. работают с таблицами данных. С одной стороны, это дает удобство формирования запросов, поскольку можно использовать SQL стандарт. Но, с другой стороны, время обработки запроса сильно возрастает при увеличении количества объектов, поскольку все объекты одного *типа* собираются в одну большую таблицу. В задачах 3D визуализации данные лучше объединять не по типам, а по признаку принадлежности.

Другой недостаток реляционной базы в том, что объекту трудно добавить новое поле данных. Для этого нужно либо перестраивать таблицу, либо заранее резервировать поля или строить дополнительные таблицы для новых свойств. Иначе говоря, реляционная модель плохо расширяема. В задачах 3D визуализации новые свойства появляются достаточно часто – по мере развития 3D акселераторов и появления новых методов обработки, поэтому описание 3D сцен, как правило, представляет собой иерархическую структуру и больше похоже на файловую систему.

Таким образом, для систем 3D визуализации более применима иерархическая модель базы данных (IMDB), а не реляционная. Такая система была реализована на основе иерархической БД реального времени в формате XQL

Организация иерархической БД

Одна из причин медленной обработки стандартных SQL запросов состоит в том, что ко времени исполнения запроса и формирования данных добавляется еще время синтаксического анализа запроса. Поэтому для ускорения обработки запросов в иерархической XQL базе данных максимально упрощен синтаксис и исключены комбинированные запросы, которые требуют анализа сложных выражений.

Для формирования запроса используются операции присваивания, разделенные точкой с запятой. В качестве переменной используется строковый идентификатор узла, состоящий из комбинации имен родительских узлов, разделенных точкой, корень дерева обозначается двоеточием, например:

```
:MODEL.Buran.CAMERA.View.FOV = 48;
MATERIAL.Black.DIFFUSE = [0,0,0];
```

Для сокращения записи пути к узлам, имеющим общего родителя, используется оператор {}, например:

```
:MODEL.Buran.CAMERA.View = { FOV =
48; POS=[0,0,0]; }
```

Идентификация узлов дерева, происходит практически так же, как в файловой системе или системном реестре, поэтому структура БД может рассматриваться как расширение файловой системы, т. е. некоторые ветки БД могут быть обычными файлами или частью системного реестра.

Для организации иерархической базы данных, вводится два основных класса:

XNode – узел дерева, который имеет имя, хранит в себе данные и список дочерних узлов;

XData – обобщенные данные хранят тип данных и сами данные, возможны комбина-

[6] См.: http://web.progress.com/en/objectstore/

[7] Object Data Management Group. Standard for Storing Objects. http://www.odbms.org/odmg/

ции трех типов: целое (int), вещественное (float) и строка (string). Например:

```
MESH.Bottom.MATERIAL =
Black,[0,100],White,[100,200];
MESH.Quad.FACE = [0,1,2],[1,2,3];
```

Доступ к базе данных в C++ возможен двумя способами: запуск XQL скрипта на исполнение либо прямое обращение к DB, например:

```
DB.parse(«Model.Buran.Camera.View.FOV
= 48»);
DB[«Model.Buran.Camera.View.FOV»] =
48;
```

Обработка событий

Важным параметром, кардинально влияющим на архитектуру всей системы, является скорость доступа к БД. Слишком большое время доступа вынуждает размещать запросы к БД в отдельных тредах или даже процессах, что сильно усложняет не только разработку, но и функционирование системы. Уменьшение времени обработки запроса до времен, сравнимых с обращением к простому хеш-массиву, позволяет значительно упростить систему и работать с БД синхронно, без специальных тредов ожидания и отложенных запросов.

К сожалению, хранить *все* данные в БД пока мешают ограничения по времени доступа, прямое обращение к данным класса все-таки быстрее, поэтому предлагается использовать оба подхода одновременно – часть данных класс хранит в своих структурах и часть считывает из БД.

Для того чтобы обеспечить обновление данных, хранящихся в классе, предлагается использовать систему плагинов – обработчиков, которые реагируют на события *create*, *write*, *read*, *delete* и т. п. Плагин регистрируется на определенные события по модификации БД, и по мере изменения данных вызывается соответствующий метод в плагине обработки. В качестве параметра передается узел, в котором произошла модификация. В C++ функция обработки выглядит так:

```
on_write(XNode node) { fov = node; }
on_read(XNode node) { node = fov; }
```

Кроме простой синхронизации данных, плагины могут использоваться для более сложных вычислений, поскольку результат обработки может формироваться на основе других данных класса и данных из других узлов или на основе запросов к внешним устройствам. Другими словами, результат работы плагина в общем случае является процедурным объектом и может динамически меняться от запроса к запросу.

Собственно, регистрация в узле БД определенного плагина и определяет функционирование этого узла. И соответственно распределение плагинов по БД определяет функциональность всей системы.

Поскольку плагины имеют возможность взаимодействовать с другими аппаратно-программными устройствами, то именно через них и осуществляется обмен БД с внешним миром – 3D акселераторами, манипуляторами, джойстиками, мышью и др. Такой обмен в конечном счете приводит к формированию изображения и пополнению БД новыми данными.

Интерактивность

Интерактивность подразумевает способность системы к модификациям и исполнению скрипт-команд в реальном времени. В этой связи чем больше данных содержится в глобальной БД, тем проще управлять системой, поскольку *любой* модуль получает через БД универсальный доступ практически к *любым* параметрам системы, таким как положение объектов, состояние анимации, параметры растрирования – вплоть до цвета отдельной грани, или *shader* параметров GPU и т. п.

Особенно актуален такой подход при внедрении 3D сцен на HTML страницы Web приложений, поскольку позволяет приложению полностью контролировать создание и модификацию 3D объектов через модификацию глобальной БД.

Разработанная система XQL запросов интегрирована с распространенными скриптовыми языками программирования JScript, VBScript и др. на основе входящей в состав Windows компоненты ActiveScript [8]. Инте-

[8] *Cluts N.* Microsoft ActiveX Controls Overview. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaxctrl/html/msdn_actxcont.asp

грация выполнена на уровне внедрения в контекст ActiveScript машины COM объекта (с символьным именем «DB»), предоставляющего доступ к глобальной базе данных системы.

При изменении командами скрипта свойств «DB» выполняется запись в базу данных системы и производится стандартная отработка изменений соответствующими плагинами. При чтении свойств DB возвращаются данные из БД системы.

Такой подход позволяет не только использовать стандартные языки программирования для моделирования внешнего вида виртуальных объектов, но и программировать сколь угодно сложное их поведение. Также система предоставляет механизмы для динамического создания 3D объектов, процедурных 3D объектов, таких как 3D графики, гистограммы и т. п.

Формат представления данных

Поскольку функционирование системы сводится к обращениям к БД, то загрузка данных из файла – это по сути всего лишь исполнение запросов, содержащихся в этом файле. Поэтому проще всего выбрать формат хранения данных, прямо совпадающий с языком запросов БД. Тогда для загрузки файла достаточно отправить его содержимое как единый запрос DB. Альтернативным форматом хранения данных может быть XML формат. Но в этом случае потребуется дополнительный модуль для загрузки XML файлов в базу данных.

Поскольку файл полностью описывает сцену, то загрузка сцены из файла всегда будет выполняться максимально детально. Но в результате унификации обмена у любого модуля (например, скрипт-интерпретатора) существуют те же средства контроля БД, что и у загрузчика сцены. Поэтому любой модуль имеет те же возможности, что и загрузчик сцены, т. е. максимально полный доступ.

В настоящее время существуют XQL экспортеры для наиболее распространенных редакторов 3D сцен, таких как 3DS Max и Maya. Система также поддерживает импорт популярных форматов представления 3D данных VRML, DXF.

Заключение

В работе проведен анализ требований к интерактивной системе визуализации, сформулированы основные принципы построения системы и предложен вариант реализации системы 3D визуализации, основанный на использовании иерархической объектно-ориентированной базы данных в качестве *звездного* объекта, связывающего составные части системы.

Список литературы

1. *Долговесов Б. С.* Семейство компьютерных систем визуализации «Альбатрос» // Автометрия. 1994. № 6. С. 3.

2. *Белаго И. В., Кузиковский С. А., Некрасов Ю. Ю.* Управление компьютерными генераторами изображений реального времени // Автометрия. 1994. № 6. С. 48.

3. *Вяткин С. И., Долговесов Б. С. и др.* Архитектурные особенности системы визуализации реального времени на основе сигнальных процессоров // Автометрия. 1999. № 1. С. 110.

4. *Долговесов Б. С., Мазурок Б. С., Ванданов В. Г.* Разработка новых и адаптация существующих базовых алгоритмов визуализации 3D сцен для Web приложений // Тр. XVI Междунар. конф. по компьютерной графике и ее приложениям «Графикон-2006». Новосибирск, 2006.

Материал поступил в редколлегию 14.04.2010

B. S. Dolgovesov, B. S. Mazurok, V. G. Vandanov

OBJECT-ORIENTED DATABASE FOR INTERACTIVE 3D VISUALIZATION SYSTEM

In this paper the requirements for interactive visualization system has being analyzed, the basic principles of system construction was formulated, and been proposed 3D visualization system, based on the usage of fast hierarchical object-oriented database linked components of the system.

Keywords: OODB, XQL, 3D visualization systems.