УДК 004.946

МЕТОДЫ И СПОСОБЫ ОРГАНИЗАЦИИ ХРАНИЛИЩ ДЛЯ РАСПРЕДЕЛЕННЫХ ИНТЕРАКТИВНЫХ СИСТЕМ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

Черняев А.А., Башков Е.А.
Донецкий национальный технический университет,
кафедра прикладной математики и информатики
chernyaev_a_a@mail.ru

Рассмотрены задачи и области применения виртуальных миров, разбор организации хранилищ на основе систем виртуальной реальности, сделаны выводы об актуальности исследований способов организации хранилищ для распределенных интерактивных систем.

Введение

Виртуальный мир – это интерактивное смоделированное окружение, к которому множество пользователей имеют доступ через онлайн интерфейс [3]. Виртуальные миры иногда называют «цифровыми мирами», «искусственными мирами» и «МОИ» (мультипользовательские он-лайн игры, MMOG). На сегодня существует множество разных виртуальных миров, все из которых имеют шесть признаков, которые присущи всем мирам:

1. Совместное пространство: участвовать в жизни мира могут одновременно много пользователей.
2. Графический пользовательский интерфейс: пространство в мире отражено виртуально, и варьируется по стилю от 2D «мультипликационного» изображения до более впечатляющих 3D изображений.
3. Оперативность: общение происходит в режиме реального времени.
4. Интерактивность: мир позволяет участникам изменять, развивать, строить или принимать содержание, подобранное специально для него.
5. Постоянство: существование мира продолжается независимо от того, находятся ли отдельные пользователи в системе.
6. Общение / общество: мир дает возможность и содействует формированию социальных групп внутри мира, таких как команды, гильдии, клубы, клики, соседства, комьюнити и так далее.

Виртуальные миры делятся на несколько типов по своему назначению:

– коммерческие;
– коммуникация / построение онлайн сообществ;
– образование;
– военное обучение.

С дальнейшим развитием подобных технологий в течение следующих лет, виртуальные миры смогут использовать для самых разных целей, поскольку все больше людей проводят в них время.

Анализ структур данных

Системы виртуальной реальности должны содержать всю информацию об устройстве внутреннего мира в некоторых хранилищах, базах данных. Примером такой информации могут служить данные о персонажах, текстуры, данные о взаимодействии объектов, типы объектов и т.д. Существует большое количество способов хранения информации в базах данных, но оптимальным вариантом является логическое разделение между таблицами. Например, для пользователей создать таблицу users, в которой будут присутствовать все поля, необходимые для описания данных о пользователе, для структур создать таблицу textures, в которой будет храниться информация о текстуре и т.д. В табл. 1 приведены примеры простых таблиц для описания информации.

Таблица 1. Пример простых таблиц для хранения информации объектов

Название таблицы	Название поля	Назначение поля
Users		содержит данные о пользователях
	Id_user	уникальный идентификатор пользователя
	Email	электронный адрес
	Pass	пароль пользователя
	Name	имя пользователя, которое будет отображаться в виртуальном мире
	Id_avatar	идентификационный номер аватара
Avatars		содержит данные об аватарах
	Id_avatar	идентификационный номер аватар
	Id_sex	тип пола аватара
	Id_hair	тип волос
	Id_face	тип лица
	Id_body	тип строения тела
	Id_clothes	тип одежды
Textures		содержит данные о текстурах
	Id_texture	идентификационный номер текстуры
	Path	путь к изображению текстуры
	Id_type	тип текстуры
	Height	высота
	Width	ширина

У каждого аватара может быть свой инвентарь, список друзей и людей, с которыми не хотелось бы общаться, список сообщений. Всю эту информацию тоже необходимо хранить в базе данных. Все вышеперечисленные таблицы можно отнести к обязательным, которые должны быть реализованы во всех многопользовательских играх (мирах). Кроме того, существуют специфические особенности, которые могут быть реализованы не во всех виртуальных мирах (например, начисление опыта, убийство персонажей и т.д.). Следовательно, какие данные хранить в базе данные, каким способом распределять данные между таблицами, зависит от назначения виртуального мира.

В реальных системах количество полей и их назначение может отличаться от тех, что приведены в табл. 1. Примером могут служить базы данных платформ Opensim [2], UEF [1], ActiveWorlds и OpenFlight. Платформа Opensim является достаточно популярной среди других подобных систем (предназначена для общения между людьми, проведение виртуальных конференций, выставок, галерей). Проект UEF является будущей многопользовательской игрой, главной целью игры – прокачивание персонажа. В данной работе было проведено более глубокое исследование БД OpenSim и других систем

Информация о приме может содержаться в различных таблицах. При создании объекта, в таблицы prims и primshapes добавляется запись о данном примитиве. Эти таблицы являются связанными, т.к. разделяют между собой информацию об объекте, следовательно, эти таблицы имеют одинаковое количество записей. Они связаны между собой с помощью поля UUID, которое характеризует уникальный ключ объекта. В таблице prims содержится такая информация, как угол наклона примитива, его расположение в виртуальном мире, цвет и т.д. В таблице primshapes содержатся данные, влияющие на деформацию объекта.

Новый созданный объект не принадлежит пользователю, следовательно, он не хранится в его инвентаре. Пользователь может поместить объект в инвентарь с помощью нескольких вариантов:

- удалить объект (прим будет помещен в корзину);
- надеть объект (прим будет помещен в папку «objects»);
- привязать объект к некоторым частям тела (прим будет помещен в папку «objects»).

Когда объект помещается в инвентарь, о нем создается запись в таблице inventoryitems. Но для того, чтобы данные находились в таблице inventoryitems, необходима запись в таблице assets, поэтому при перемещении объекта в инвентарь пользователя, запись из таблиц prims и primshapes удаляется, а в таблицу assets заносится. Кроме того, есть еще одна таблица primitems, которая служит для связывания скрипта и объекта (группы объектов), для которых данный скрипт должен выполняться. В таблице содержится такие поля, как primID и assetID. Поле primID указывается на поле с записью из таблицы prims (объект), а поле assetID на поле с записью из таблицы aseets (скрипт)

Любой примитив можно получить из квадрата, путем изменения нескольких его свойств. В табл. 2 содержатся данные, которые получает куб по умолчанию, без изменения каких-либо параметров. Данные взяты из таблицы primshapes. На рис. 1 изображен куб с параметрами по умолчанию.

При установке Top Shear X в значение 0,5; Taper X – в значение 1 получилась призма, в базе данных появились новые значения (табл. 3). На рис. 2 изображена получившаяся призма.

Если поменять Top Shear X на значение -0.5, то значение Path Shear X станет равным 206. Является очевидным, что когда Top Shear изменяется от 0 до 0.5, запись в поле PathCurve меняется от 0 до 50, если значение Top Shear отрицательное, то данное значение умножается на 10 и вычитается из значения 256. Следовательно, если для Top Shear указать значение -0.5, то в Path Shear будет записано значение 206.

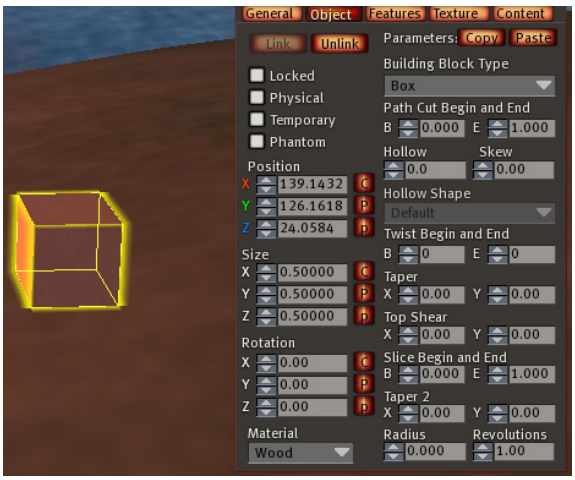

Рисунок 1. Куб с параметрами по умолчанию

Таблица 2. Стандартные параметры куба в таблице primshapes

PathBegin	0
PathEnd	0
PathScaleX	100
PathScaleY	100
PathShearX	0
PathShearY	0
PathCurve	16
ProfileCurve	1
ProfileBegin	0
ProfileEnd	0

Если Taper X установить в значение –1, то запись в поле PathScale примет значение 0. Следовательно, значение 100 для данного поля PathScale является средним, т.е. когда Taper установлен в значение 0. Если Taper равен 1, PathScale будет равен 200, если Taper равен –1, PathScale будет равен 0.

Если значения в полях Top ShearX и TopShearY установить в значения 1, то из квадрата получится призма.

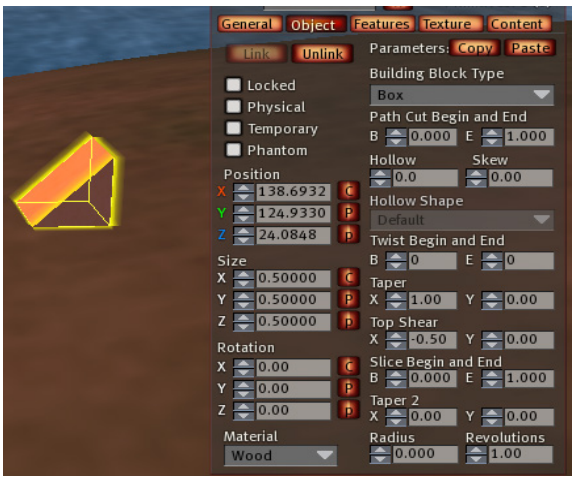

Рисунок 2. Призма, полученная из куба

Таблица 2. Измененные параметры куба

PathBegin	0
PathEnd	0
PathScaleX	200
PathScaleY	100
PathShearX	50
PathShearY	0
PathCurve	16
ProfileCurve	1
ProfileBegin	0
ProfileEnd	0

В настройках примитива можно указать тип строительного блока (building block type), который указывает на то, как должен выглядеть примитив, возможные значения: box, cylinder, prism, sphere, torus, ring, sculpted.

Информация о том, какой тип использовать, хранится в поле ProfileCurve.

Для того, чтобы вырезать часть объекта, необходимо изменить параметры Path Begin and End, параметры меняются от 0 до 1.

Второй проект – UEF, представляет многопользовательскую игру, главной целью которой является развитие персонажа. Вся информация об исходном мире хранится в базе данных. Учитывая, что эта игра будет направлена на развитие персонажей, а не общение как в Opensim, то должна быть еще некая ветка развития. Например,

персонажи могут поднимать свой уровень, открывая перед собой новые способности и возможности. Данную информацию тоже необходимо хранить в базе данных.

Масштабируемость архитектуры закладывается изначально [2]. Все процессы станут крутиться вокруг баз данных, где будут храниться динамически изменяемые свойства объектов. Причём группы объектов могут выноситься в отдельные базы, которые можно будет переносить на отдельный сервер (или даже кластер) вместе с сопутствующими обработчиками событий. Вплоть до выделения особо нагруженных звёздных систем или высокоразвитых планет, а возможно и крупных (галактического масштаба) объединений, в отдельную базу данных вместе с программами управляющими событиями именно в данной группе объектов.

В главной базе будут храниться общие понятия о мире: информация о пользователе, его инвентарь, достижения. Для создания крупных объектов целесообразней будет выделить отдельную базу для хранения данных о нем, чтобы разгрузить нагрузку на главный сервер. В итоге создание планеты будет сопровождаться созданием комплекта из базы данных и модуля управления природой: освещённость и температура, облака и осадки, наполнение рек, приливы в морях и океанах, циклы активности животных и растений, возникновение и развитие поселений.

Кроме того, в главной базе хранятся все данные об окружающем мире: константы, формулы, простые алгоритмы, статичные и динамичные данные. В системе существует децентрализованная система менеджеров обработки событий. Она запускает экземпляры обработчиков событий в соответствие с потребностью системы (возникновение новых событий) прекращающая их деятельность при простое, либо зависании (бесконечной рекурсии), а также осуществляющая сравнение версий обработчиков событий на устойчивость и эффективность работы. Основная задача менеджеров, распознать возникновение нового события и запуск подходящего обработчика

OpenFlight больше не платформа, а формат. OpenFlight организует визуальные базы данных в группы и обеспечивает в режиме реального времени функции, такие как выбор полей-обзора, переключение между уровнями детализации. Каждая база OpenFlight организована в древовидной структуре.

Древовидная база данных состоит из узлов. Большинство узлов может иметь дочерние узлы, также как и соседние. В общем, эти узлы можно рассматривать в трех иерархических классах. Начиная с вершины иерархии, эти три классовых узла включают контейнер узлов, узлы геометрии и узлы вершин.

Контейнер узлов – узлы, которые навязывают некоторую логическую группировку или поведение на множество узлов в ней. Группа узлов, например, позволяет собирать похожие узлы под одного предка, если существует такая необходимость. Вы можете выбрать группировку своих пространственно или по каким-либо другим критериям. Другой распространенный контейнер узлов, уровень детализации узла, накладывает определенное визуальное поведение на узлы, содержащиеся в нем. Он определяет диапазон расстояний внутри, которые узлы в ней видны.

Узлы геометрии – узлы, которые на самом деле представляют некоторые физические (визуализируемые) геометрии. Атрибуты узлов геометрии в основном включают визуальные атрибуты, такие как, цвет, материал, текстура и т.д. Два

основных геометрических узла в OpenFlight – лица и меши. Другие геометрические узлы включают точки света и текстовые узлы. Хотя OpenFlight позволяет, очень мало случаев, когда хотя бы один узео геометрии не содержался бы где-нибудь внизу узла контейнера.

Узлы вершин являются строительным блоком для узлов геометрии. Индивидуально, узлы представляют собой дискретную точку в пространстве. Собранные вместе в узле геометрии, такие как узел лица, набор узлов вершин представляет замкнутый (или не замкнутый) цикл. Замкнутый цикл узла вершин определяет лицо (или полигон). Передняя сторона лица определяет порядок, в котором узел вершин появляются над узлом лица. Незамкнутый цикл вершин узлов определяется набор отрезков, опять же ориентированных в соответствии с порядком, в котором узлы вершин появляются. Каждый тип узла имеет атрибуты данных, специфичные для его функции в базе данных.

Иерархическая структура базы данных OpenFlight хранится на диске в виде файла. Файл состоит из линейного потока двоичных записей. Порядок байтов big endian. Все OpenFlight записи начинаются с 4х байтов. Первые два байта этой последовательности байтов идентифицируют тип записи (код операции), а вторые два байта указывают длину записи. Заметим, что длина включает данные четыре байта, поэтому минимальная длина записи (без каких-либо дополнительных данных), будет равна четырем байтам. Учитывая эту постоянную структуру, записи OpenFlight можно легко считать с диска и обработать.

- Все записи OpenFlight кратны четырем. Если запись получается не кратной, то добавляется байты, чтобы быть кратными четырем. В некоторых случаях записи добавляются, чтобы быть кратными восьми.
- длина всех записей (а также поля во всех записях), также как и смещение всех полей, выражена в байтах
- если не оговорено иначе, битовые поля и маски начинаются с нуля (т.е. первый бит, это номер 0)
- если не оговорено иначе, элементы матрицы записи, хранящиеся в OpenFlight, появляются в большой строковый заказ. Это означает, что элементы матрицы появляются в следующем порядке.

  row0col0, row0col1, row0col2, row0col3,
  row1col0, row1col1, row1col2, row1col3,
  row2col0, row2col1, row2col2, row2col3,
  row3col0, row3col1, row3col2, row3col3
- длина всех OpenFlight записей ограничена наибольшим значением, которое может быть закодировано двумя байтами (65536). для фиксированного размера записи, этого размера достаточно.
- максимальное число дочерних узлов при любом первичном – 65534.

Active Worlds является платфомой виртуальной реальности, где пользователи сами могут назначить себе имя, войти во вселенную и исследовать 3D мир, построенный другими пользователями.

SDK Active Worlds обеспечивает простой способ программистам для реализации приложений, которые функционируют в рамках виртуальной среды ActiveWorlds. SDK может быть использован для автоматизированных программ, которая исследует мир и

создает карту.

Основной компонент SDK является файл aw.dll. Для разработки приложений с использованием ActiveWorlds SDK, программист пишет программы на языке C, которая включает файл aw.h и подключает библиотеку aw.dll. Скомпилированная программа может быть запущена на любом компьютере в любом месте, при условии что есть подключение к сети и aw.dll доступен на данном компьютере.

Active World SDK использует атрибуты для передачи данных туда и обратно между приложением и SDK. Атрибуты похожи на переменные в том, что они хранят значения, которые могут быть запрошены и изменены. Многие атрибуты, доступны только для чтения, это означает, что они не могут быть изменены приложением и существуют только для передачи информации в приложение из SDK.

Как правило, для того, чтобы передать данные в SDK, SDK приложение устанавливает один или несколько атрибутов перед вызовом SDK метода. После завершения работы метода, приложение запрашивает один или несколько атрибутов для извлечения результатов операции или события.

Выводы

Исследования показали, что существует множество различных конфигураций баз данных для распределенных систем. Не существует конкретного способа создания конфигураций, все зависит от того, с какой целью создается проект [4].

К сожалению, нет обоснования, почему именно так организована база данных в OpenSim. В данной работе ставится задача проанализировать эффективность имеющихся решений и предложить модификации, с целью повышения эффективности критериев:

– размер;
– время доступа к информации;
– минимизация объема передаваемых данных.

Данные критерии важны для систем, в которых участвует большое количество пользователей. Одним из решений для повышения эффективности является построение многоуровневой БД (фронт-энд, бэк-энд сервера с разделением таблиц). Данный этап еще в разработке.

Список источников

[1] Виртуальные миры [Электронный ресурс]. http://uef.me

[2] Developer Documentation – OpenSim [Электронный ресурс]. http://opensimulator.org/wiki/Development

[3] Виртуальные миры. SecondLife. [Электронный ресурс] http://world2.ru

[4] Graham Morgan. Scalable Massively Online Games. Режим доступа http://www.cs.ncl.ac.uk/publications/trs/papers/888.pdf свободный. – Загл. с экрана. – Яз. англ.