ГИАЦИНТОВ
Алексей Михайлович -
ведущий программист Центра визуализации и спутниковых информационных технологий Научно-исследовательского института системных исследований РАН
Адрес: 117218, г. Москва, Нахимовский просп., 36, к.1
e-mail: algts@inbox.ru

МАМРОСЕНКО
Кирилл Анатольевич -
кандидат технических наук, научный сотрудник Центра визуализации и спутниковых информационных технологий Научно-исследовательского института системных исследований РАН
Адрес: 117218, г. Москва, Нахимовский просп., 36, к.1
e-mail: kirillam@ya.ru

Одновременное воспроизведение разнородных видеоматериалов в виртуальной сцене в подсистеме визуализации ТОС[1]

Тренажерно-обучающие системы (ТОС) - разновидность технических систем, позволяющих решать задачи подготовки персонала с целью обучения управлению сложными техническими системами, в том числе в условиях, создающих угрозу жизни (агрессивные среды) или принудительно не реализуемых в реальной среде. Кроме формирования индивидуальных профессиональных навыков и умений, ТОС могут применяться для отработки групповых операций.

Тренажерно-обучающие системы должны отвечать требованиям методик подготовки персонала и, как правило, содержат значительное количество информационных ресурсов. Одним из видов ресурсов, используемых при обучении, являются видеоматериалы.

Подсистема визуализации обеспечивает отображение результатов моделирования внешней среды и объекта управления с помощью устройств отображения информации. Отображение разнородных видеоматериалов в виртуальной трехмерной сцене является одним из требований к ТОС.

Воспроизведение видеоматериалов внутри виртуальной трехмерной сцены является сложной задачей, так как необходимо учитывать факторы, такие как производительность подсистемы визуализации, производительность видеокарты. Подсистема визуализации должна обеспечивать отображение трехмерной сцены с приемлемой частотой кадров (не менее 25 кадров в секунду) и должна быть способна при этом реагировать на внешние воздействия, в том числе на изменения параметров трехмерной сцены или загрузку дополнительных объектов. Для отображения нескольких видеоматериалов высокой четкости в виртуальной трехмерной сцене нами была разработана и реализована архитектура декодера кадров видео. Для обеспечения требуемой производительности подсистемы визуализации при отображении видеоматериалов высокой четкости нами были созданы методы эффективной передачи данных в видеопамять.

Для декодирования видеофайлов была выбрана кросс-платформенная система FFmpeg, распространяемая по лицензии LGPL. Данная система является базовым средством декодирования видеоматериалов в операционных системах семейства Linux, а также используется в операционных системах Windows (в составе пакета FFDShow) и MacOS X. Основой системы FFmpeg является библиотека libavcodec, в которой хранятся различные кодеры и декодеры для работы с аудио- и видеоинформацией.

Структура видеофайла - контейнер, содержащий общую информацию о содержимом контейнера (количество видео- и аудиопотоков, наличие субтитров, информация о разделах, метаданные - название, год создания, автор и т.д.) и различных потоках информации (видео, аудио, субтитры и т.д.). Некоторые контейнеры допускают использование только определенных кодеров для представления информации. Наиболее популярными контейнерами являются AVI, OGV, MP4, MKV, MOV, WMV.

В большинстве контейнеров данные представляются в виде пакетов (в некоторых форматах называемые chunks (куски) или segments (сегменты)). Пакеты хранят закодированную информацию определенного типа - видео, аудио или субтитры и т.д. В большинстве видеофайлов количество аудиопакетов значительно меньше, чем количество видеопакетов - приблизительно 75% видеопакетов и 25% аудиопакетов. В зависимости от типа контейнера и используемого кодера расположение пакетов внутри контейнера может быть разным. Зачастую пакеты расположены в виде последовательности очередей пакетов разного типа, например, около 20 видеопакетов, расположенных подряд, затем несколько аудиопакетов, после которых снова подряд расположены около 20 видеопакетов.

Пакеты хранят в себе определенный сегмент аудио- или видеоинформации. В большинстве случаев в видеопакете хранится целый видеокадр, но в некоторых форматах (в частности, в формате MPEG 2) один видеокадр может состоять из нескольких пакетов. Это связано с тем, как кодер сохраняет информацию о видеокадре. Например, в формате MPEG 2 присутствует три типа кадров - независимо сжатые кадры (I-кадры), кадры, сжатые с использованием предсказания движения в одном направлении (P-кадры), и кадры, сжатые с использованием предсказания движения в двух направлениях (B-кадры). Соответствующие группы кадров от одного I-кадра до другого образуют GOP - Group Of Pictures - группу кадров.

[1] *Работа выполняется при поддержке РФФИ, грант № 11-07-00158-а.*

Термины, используемые в статье:

- *видеокадр - массив данных, хранящий декодированное изображение из видеофайла в оперативной памяти, в цветовой модели BGRA;*
- *кадр подсистемы визуализации - изображение, сгенерированное рендерером, выводимое в подсистему отображения информации;*
- *видеотекстура - текстура в цветовой модели BGRA, используемая для представления видеокадра в подсистеме визуализации;*
- *цикл генерации кадров подсистемы визуализации - процесс обновления генерируемого рендерером изображения.*

Большинство форматов видеофайлов представляют видеоинформацию в цветовой модели YUV (Y - яркость, U и V - цветоразностные составляющие).

В пакетах, кроме закодированных данных и информации о размере и типе данных этого пакета, также присутствует информация о времени, когда необходимо декодировать данный пакет (последовательности декодирования пакетов), и времени, когда декодированная информация должна быть отображена или воспроизведена.

Аудиоинформация видеофайла может быть закодирована с использованием различных форматов сжатия, таких как MP3, AAC, AC3, WMA и т.д. Для корректного воспроизведения аудиоинформации необходимо знать количество каналов (моно-, стерео-, 5.1, 7.1 и т.д.) и частоту дискретизации. После декодирования звукового пакета декодированные данные представляются в формате PCM (Pulse Code Modulation, импульсно-кодовая модуляция), которые передаются звуковой карте для воспроизведения звука.

Пакет FFmpeg не предоставляет возможностей для визуализации декодированных видеоизображений и воспроизведения звуковых данных. Для воспроизведения звуковых данных из декодируемого видеофайла был использован пакет SFML, распространяемый по лицензии zlib/png. Пакет SFML является кросс-платформенным и предоставляет возможности по работе с графикой, звуком, различными системами ввода информации и передачей данных по сети.

В качестве основы подсистемы визуализации ТОС был использован графический движок Horde3D, распространяемый по лицензии EPL. В Horde3D применена стратегия интеграции, отличная от графических движков OGRE 3D и Irrlicht. Данные графические движки предоставляют объектно-ориентированную библиотеку классов, используя которые пользователь может создать собственные реализации. В Horde3D используется более высокий уровень абстракции, чем в других движках, а основная функциональность доступна через небольшой процедурный интерфейс, в некоторых аспектах схожий с Microsoft Win32 API. Преимуществом простого С-интерфейса является большая наглядность, а также простота изучения функциональности и особенностей движка. Другим преимуществом является лучшая портируемость. Практически в любых языках программирования, в том числе и скриптовых, есть механизм доступа к функциям внешних библиотек, в то время как импорт классов намного сложнее или вовсе невозможен. Horde3D может быть использован в таких языках, как C#, Java, LUA, Python и др.

Из-за высокого уровня абстракции интерфейса прикладного программирования (API) пользователь не может добавлять новую функциональность, например, новый тип узла сцены, без модификации исходных кодов движка. Для преодоления этого недостатка Horde3D предоставляет механизм расширений, который дает пользователю полный доступ к внутренней структуре движка. Расширение статически присоединяется к библиотеке движка и предоставляет свою функциональность через процедурный интерфейс.

Разработанная архитектура графической подсистемы позволяет декодировать и отображать одновременно несколько видео высокой четкости в трехмерной сцене. Состав архитектуры включает: декодер видео, в котором происходит декодирование видео- и аудиопакетов; подсистему воспроизведения декодированного звука; управляющую структуру, необходимую для запуска видео, паузы воспроизведения, выставления громкости воспроизводимого видео и т.д.; интерфейс взаимодействия с движком, необходимый для обновления видеокадров; интерфейс управления графической подсистемой.

Общая схема работы декодера представлена на **рис. 1**.

Во время инициализации декодера считываются все необходимые параметры о видеофайле, такие как: кодеки, используемые для сжатия видео- и аудиоинформации, количество звуковых и видеопотоков в файле, ширина и высота (разрешение) видео, количество каналов звука и т.д.

В зависимости от формата видеофайла размещение пакетов в файле различается, следовательно, сложно предсказать, в каком порядке будут следовать видео- и аудиопакеты. Это может привести к ситуации, когда очередь видеопакетов заполнена, а очередь аудиопакетов будет пуста. Узнать, какого типа будет следующий пакет, невозможно до его чтения. Если следующий видеопакет окажется закодированным видеокадром, то либо этот пакет необходимо помещать в очередь видеопакетов, т.е. превышать заданные рамки очереди, либо удалять пакет, что приводит к артефактам на конечном изображении или пропущенным кадрам и рывкам воспроизводимого видео. Для решения этой проблемы создали третью очередь пакетов, в которую помещаются пакеты различного типа из видеофайла, а затем сортируются по типу пакета (аудио- или видеопакет) в одну из очередей. Пакеты сортируются по методу FIFO («первый вошел, первый вышел»).

Видеодекодер проверяет наличие пакетов в очереди видеопакетов и при наличии пакетов забирает пакет из начала очереди. Процесс декодирования видеопакета состоит из двух этапов - декодирование пакета и преобразование декодированного кадра в необходимую цветовую модель. После декодирования и преобразования в другую цветовую модель кадр помещается в очередь декодированных кадров. Аналогично аудиодекодер проверяет наличие пакетов в очереди и при наличии пакетов декодирует первый пакет из очереди. После декодирования звуковые данные в формате PCM помещаются в очередь декодированных звуковых данных.

Так как подсистема визуализации принимает изображения только в цветовой модели BGRA, то после декодирования видеокадр будет преобразован в эту цветовую модель. Цветовая модель BGRA выбрана не случайно - большинство форматов изображений и большая часть оборудо-

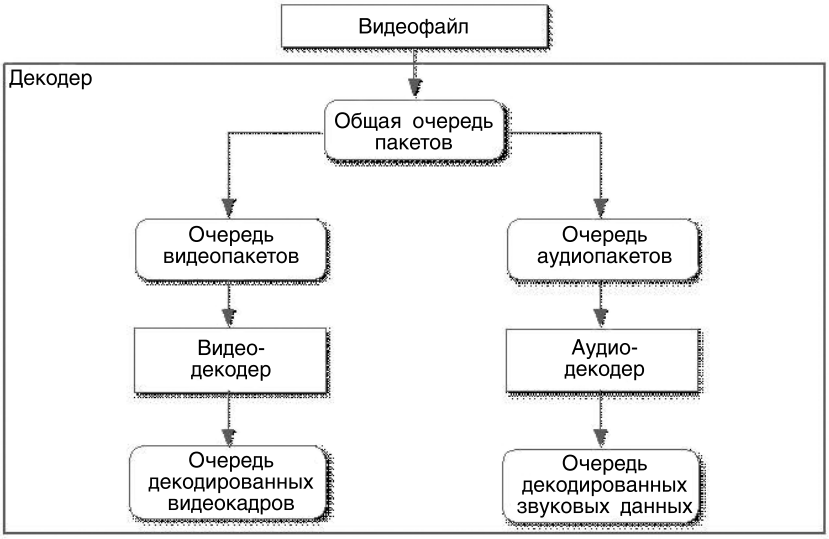

Рис. 1. *Общая схема работы декодера видеофайла*

вания хранят изображения в формате BGRA. При использовании других цветовых моделей происходит преобразование используемой цветовой модели в цветовую модель BGRA, выполняемое на центральном процессоре. Так как любое лишнее преобразование снижает производительность графической подсистемы, то использование цветовой модели BGRA является оптимальным.

Процесс декодирования является ресурсоемким процессом. Например, декодирование одного видеопакета с разрешением 1280 на 720 (720p) на процессоре Intel Core 2 Quad 2,66 ГГц занимает от пяти до десяти миллисекунд, для видео размером 1920 на 1080 (1080p) - от 10 до 20 мс. Декодирование пакета звуковых данных занимает значительно меньше времени (0,5-2 мс), но, если одновременно декодируется звук для нескольких воспроизводимых видео, это также может серьезно снизить производительность. Скорость получения пакетов из видеофайла ограничивается производительностью системы хранения данных (СХД), что в свою очередь влияет на производительность подсистемы визуализации. Задержки образуются, если в момент считывания пакетов из видеофайла СХД обрабатывала другие команды, либо файл являлся фрагментированным, что заставило бы СХД искать фрагменты данного файла. Для преодоления проблем с производительностью была использована многопоточность.

Многопоточность - свойство платформы (например, операционной системы) или приложения, состоящее в том, что процесс, запущенный на платформе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение позволяет достичь более эффективного использования ресурсов вычислительной машины.

Все потоки выполняются в адресном пространстве процесса. Выполняющийся процесс имеет, как минимум, один (главный) поток. Применение многопоточности наиболее эффективно при использовании многоядерных процессоров. На одноядерных процессорах многопоточное приложение также будет работать, но повышение производительности будет незначительным; возможно даже снижение производительности, если разные потоки одного процесса выполняют ресурсоемкие операции.

Хотя многопоточность способна существенно повысить производительность приложения, применять ее следует с осторожностью. Внедрение многопоточности существенно повышает сложность самого приложения и его отладки, а также при некорректном использовании может привести к таким проблемам, как «состояние гонки» (когда работа системы зависит от того, в каком порядке выполняются части кода), взаимной блокировке потоков (несколько потоков находятся в состоянии бесконечного ожидания ресурсов, занятых самими этими потоками) и трудностям при обработке ошибок (исключений) внутри приложения.

Для оптимизации процесса декодирования видео используются отдельные потоки для сбора пакетов, декодирования видео- и аудиопакетов. Процесс воспроизведения звука также происходит в дополнительном потоке. Первоначально в качестве системы управления потоками был выбран пакет SFML, который предоставляет базовую функциональность управления многопоточностью на системах семейств Windows и Unix (Linux, MacOS). В целом система управления потоками SFML работала без сбоев, но было выявлено несколько недостатков. Система не позволяет выставить приоритет обработки создаваемому потоку, а также привязать создаваемый поток к конкретному процессорному ядру. Система предоставляет только один способ приостановки потока - «сон», что не позволяет точно прогнозировать, через какое время потоку вновь будет выделено процессорное время. Например, если потоку дано указание «заснуть» на 5 миллисекунд, то нет гарантий, что через 5 мс потоку будет выделено процессорное время, оно может быть выделено через 15 мс. Данное обстоятельство зависит от степени загруженности процессора, реализации планировщика задач операционной системы и приоритета выполнения потока. С целью преодоления описанных выше недостатков было принято решение интегрировать библиотеку управления потоками thread системы boost.

Система boost - это набор кроссплатформенных библиотек, расширяющих стандартные инструменты языка C++, распространяемых по лицензии Boost Software License. Часть библиотек является кандидатами на включение в следующий стандарт языка C++.

Библиотека управления потоками boost thread обладает большей функциональностью, чем библиотека SFML, и предоставляет такие возможности, как управление потоками, созданными с помощью библиотеки thread, средствами операционной системы; использование условных переменных (conditional variables) и т.д. Использование средств операционной системы для управления созданным потоком позволяет реализовывать такую функциональность, как задание приоритета выполняемого потока, а также привязка потока к определенному ядру процессора. Применение условных переменных позволяет приостанавливать поток при определенных условиях и, в отличие от функции «сон», мгновенно восстанавливать работу потока.

Обновление видеотекстуры в подсистеме визуализации происходит в основном потоке программы (потоке рендеринга). Каждый кадр подсистемы визуализации происходит проверка - следует ли обновлять видеотекстуру в трехмерной сцене или нет. Если обновление требуется, то видеокадр берется из очереди декодированных кадров и помещается в память видеокарты в качестве текстуры, которая может быть наложена на поверхность любого трехмерного объекта.

При запуске видео происходит создание пустой текстуры с размерами видеокадра, которая затем применяется к материалу трехмерного объекта (материал объекта определяет, какие текстуры и шейдеры использует данный трехмерный объект). Перед применением новой текстуры к материалу происходит резервное копирование старого материала для возможности возвращения к исходным текстурам объекта после окончания воспроизведения видео. Также для повышения производительности используются два так называемых пиксельных буфера. Пиксельные буферы применяются для хранения пиксельных данных (текстуры) в видеопамяти и позволяют значительно снизить временные затраты на обновление видеотекстуры в подсистеме визуализации. При использовании стандартной функциональности движка для обновления текстур в видеопамяти сначала происходит копирование данных из видеокадра в промежуточный массив, затем данные из промежуточного массива помещаются в текстуру. При обновлении текстуры в видеопамяти сначала происходит выделение нового места под данные и только затем происходит копирование данных из промежуточного массива в видеопамять. Все описанные выше действия выполняются центральным процессором, что значительно снижает производительность подсистемы визуализации. Например, при воспроизведении FullHD видео (1920 на 1080) время копирования данных из видеокадра в промежуточный массив составляет приблизительно 3-4 мс, а обновление текстуры из промежуточного массива в видеопамяти составляет от 5 до 8 мс. Преимущество пиксельных буферов состоит в том, что затраты процессорного времени необходимы только на помещение информации в буфер, а обновление тек-

стуры из буфера происходит практически мгновенно (около 0,1 мс) за счет того, что обработка перемещения данных из пиксельного буфера в текстуру возлагается на видеокарту.

При воспроизведении одного видео с разрешением вплоть до 1920 на 1080 (FullHD) в подсистеме визуализации либо нескольких видео с разрешением не выше 720 на 576 (576p, SD) время генерации одного кадра не превышает 40 мс (в зависимости от аппаратной платформы). Однако при одновременном воспроизведении нескольких видео высокой четкости наблюдается значительное увеличение времени генерации кадра подсистемой визуализации.

Для преодоления проблем с производительностью при одновременном воспроизведении нескольких видео высокой четкости было создано несколько методов. Первый метод состоял в том, чтобы копировать данные новых видеокадров в пиксельные буферы в отдельном потоке, а затем, в основном потоке, передавать эти данные в память видеокарты. Однако данный метод себя не оправдал, так как «узким местом» было не копирование информации в оперативной памяти, а передача данных в память видеокарты.

Второй метод основывался на использовании единого пиксельного буфера, который бы содержал кадры всех воспроизводимых видео. Преимуществом данного метода должно было стать то, что пиксельный буфер обновляется только один раз для всех воспроизводимых видео. Однако данный метод имел недостатки: была необходимость сохранять предыдущие видеокадры, чтобы избежать рассинхронизации видео и аудио, а также сложность реализации динамического расширения и уменьшения пиксельного буфера при добавлении или удалении видео в ходе работы приложения. Данный метод также не смог предоставить необходимый уровень производительности при обновлении нескольких видео.

Метод, который позволил получить оптимальную производительность при воспроизведении нескольких видео высокой четкости, основан на применении приоритетов к обновлению воспроизводимых видеофайлов.

Для каждого видео применяется весовой коэффициент, отражающий общий приоритет видео и весовой коэффициент выполняемого действия. Общий приоритет видео основывается на разрешении видео - чем ниже разрешение видео, тем выше приоритет, так как видео меньшего размера обновляется быстрее. Существует три общих приоритета видео - высокий (8), средний (4), низкий (0). Для видео с разрешением до 720 на 576 выставляется высокий приоритет, до 1280 на 720 - выставляется средний приоритет, для видео с разрешением до 1920 на 1080 выставляется низкий приоритет. Весовой коэффициент выполняемой операции меняется каждый раз при обновлении видео в зависимости от действия, которое необходимо будет выполнить при следующей ИЦ подсистемы визуализации. Выполняемые операции имеют следующие весовые коэффициенты (обновление видеотекстуры имеет наибольший коэффициент): обновление видеотекстуры (3), обновление видеотекстуры и пиксельного буфера (2), обновление пиксельного буфера (1), исключение видео (0). Операция «исключение видео» выполняется в случае, когда на следующей ИЦ подсистемы визуализации нет необходимости обновлять видеотекстуру, а также все пиксельные буферы заполнены. Видео исключается из списка обновляемых, однако весовой коэффициент выполняемого действия меняется на высший для того, чтобы состояние видео было проверено на следующей ИЦ подсистемы визуализации.

Список обновляемых видео определяется на каждой ИЦ подсистемы визуализации. На основе общего приоритета видео и весового коэффициента выполняемого действия определяется позиция видео в списке обновления.

Алгоритм обновления видеотекстуры в подсистеме визуализации был также изменен для повышения производительности. Теперь первым производится обновление видеотекстуры, а не обновление пиксельного буфера. Так как операция обновления текстуры из пиксельного буфера является асинхронной, то она выполняется очень быстро, в то время как операция обновления информации в пиксельном буфере может вызвать синхронизацию процессора и видеокарты, что приведет к задержкам. Также видеокарте после обновления пиксельного буфера может понадобиться некоторое время для обработки полученных данных. Поэтому при обновлении текстуры сразу после обновления пиксельного буфера может появиться задержка. Изменение алгоритма обновления видеотекстуры дает больше времени на обработку данных пиксельного буфера. После обновления пиксельного буфера одновременно с обработкой данных в пиксельном буфере видеокартой происходит расчет весового коэффициента действия для следующей ИЦ подсистемы визуализации и переход к обновлению следующей видеотекстуры.

Заключение

Главными преимуществами использования в ТОС таких информационных ресурсов, как видеоматериалы, является возможность визуализации процессов и внедрение изображения инструктора в виртуальное окружение.

Для реализации воспроизведения видеоматериалов в подсистеме визуализации ТОС были решены следующие задачи:

- разработаны методы и алгоритмы для одновременного воспроизведения нескольких видео, в т.ч. высокой четкости;
- произведена оптимизация алгоритмов для достижения требуемого (не более 20 мс) времени генерации кадров в подсистеме визуализации при воспроизведении видеоинформации;
- разработана подсистема визуализации ТОС, внедрен программный модуль для декодирования видео, позволяющий обрабатывать видеофайлы на различных программно-аппаратных платформах.

Литература:

1. В.Н. Решетников. Космические телекоммуникации. Системы спутниковой связи и навигации. - Санкт-Петербург, 2010. - 134 стр.

2. В.Н. Решетников. Космические телекоммуникации (начала). - Тверь: издательство «ЗАО Научно-исследовательский институт «Центр-ПрограммСистем», 2009. - 128 стр.

3. K.A. Mamrosenko. Training simulation systems for open distance learning // Proceedings Conference Open Distance Learning Towards Building Sustainable Global Learning Communities. Hanoi, Vietnam: The Gioi, 2010.

4. Гиацинтов А.М., Мамросенко К.А. Описание архитектуры подсистемы визуализации тренажерно-обучающих систем // Труды Международной молодежной научной конференции «Гагаринские чтения». - М: ИЦ МАТИ, 2011. - С. 85-87.

5. OpenGL 2.1 Reference Pages [Электронный ресурс]. - URL: http://www.opengl.org/sdk/docs/man/ (дата обращения 18.05.2011).

6. Nicolas Schulz. Horde3D Documentation [Электронный ресурс]. - URL: http://www.horde3d.org/docs/manual.html (дата обращения 18.05.2011).

7. MPEG-2 video compression [Электронный ресурс]. - URL: http://www.bbc.co.uk/rd/pubs/papers/paper_14/paper_14.shtml (дата обращения 18.05.2011).

8. YUV pixel formats [Электронный ресурс]. - URL: http://www.fourcc.org/yuv.php (дата обращения 18.05.2011).