$$U_2 = \frac{P_{1,2}}{P_{2,1}} U_1, U_3 = \frac{P_{1,2}}{P_{2,1}} U_2 U_1, ..., U_n = \frac{P_{n-1,n}}{P_{n,n-1}} U_{n-1} ... U_2 U_1$$

где U_1 – определяется из нормализованного условия (1.12):

$$\sum_{i=1}^{n} U_1 = 1$$

3. Вероятность изменения позиции пика от канала i к каналу $i+1$:

$$P_{i,i\pm1} = A_i \sum_{k=1}^{m} \exp\left(\frac{y(\pm k) - y()}{y(\pm k) + y()} \right)$$

где A_i – такая константа, что выполняется равенство (1.14):

$$p_{i,i-1} + p_{i,i+1} = 1$$

4. Вычисляется вектор функции g, необходимой для решения уравнения свертки (1.15) (с учетом параметра функции поиска пиков):

$$f * g = h$$

где h – записанный сигнал, f – восстановленный сигнал, g имеет вид:

$$G() = 1000 * e^{-\frac{-3*s^2}{2*s^2}}$$

где S – параметр функции поиска пиков.

Поиск пиков производится на восстановленном сигнале с учетом порогового значения, указанного в процентах. Пороговое значение высчитывается из значения наибольшего пика и означает, что пики, имеющие значение меньше порогового, не учитываются.

Для вычисления высоты и площади найденных пиков используется уровень фона, полученный ранее на первом этапе.

Для вычисления площади пика используется следующий алгоритм:

1. Вычисление крайних точек пересечения гистограммы и фона.
2. Разбиение площади пика на треугольники, как указано на рисунке 1.5 (зеленая линия – данные спектра, красная – вычисленный фон).
3. Вычисление площади треугольников.
4. Площадью пика считается сумма площадей треугольников.

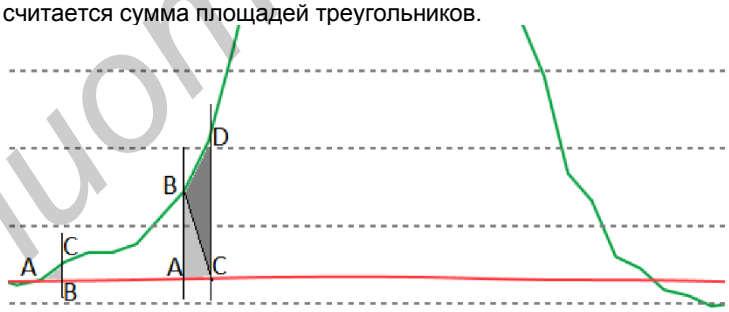

Рис. 1 – Разбиение площади пика на треугольники

Таким образом, был определен метод предварительной обработки спектра. Стоит отметить, что данный метод позволяет извлекать полезную информацию из секторов, а именно информацию о пиках,

Список использованных источников:

1. Ryan, C. G. and Clayton, E. and Griffin, W. L. and Sie, S. H. and Cousens, D. R., SNIP, a statistics-sensitive background treatment for the quantitative analysis of PIXE spectra in geoscience applications;

ОБ ОБЩИХ ПОДХОДАХ К РАЗРАБОТКЕ ГРАФИЧЕСКОГО ФРЕЙМВОРКА ДЛЯ МОБИЛЬНЫХ ПЛАТФОРМ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Ломакин Г.А.

Рудикова Л.В. – канд. физ.-мат. наук, доцент

Предлагается альтернативный вариант построения графического фреймворка для мобильной платформы на основе ОС Android. Охватываются основные аспекты реализации программного обеспечения, связанного визуализацией трехмерных объектов. Дается описание общей архитектуры разрабатываемого фреймворка.

В современном мире существуют сложившиеся и устоявшиеся подходы к разработке крупных графических приложений. Более крупные компании самостоятельно разрабатывают необходимые решения, а для всех остальных существует масса универсальных решений, например Unity 3D. В силу своей универсальности подобные платформы поддерживают несколько операционных систем или даже устройств. Кроме того, они предназначены для всевозможного применения, что сказывается, в первую очередь, на производительности. Такие решения также обычно поставляются с закрытым кодом и содержат ровно те функциональности, которые были в них заложены, т.е. без возможности не затратного расширения. Исходя из вышеизложенного, было решено разработать фреймворк на OpenGL с поддержкой одной платформы Android, который предоставляет открытый код и возможность легкого расширения. Кроме того, предлагаемый функционал должен быть в достаточной мере гибким и используемым. Абстрактная модель строится таким образом, чтобы все наиболее используемые возможности были легко доступны пользователю.

Одной из основных проблем при разработке подобных решений является реализация решения, отвечающего за позиционирование объектов в пространстве. В качестве такого решения были выбраны и доработаны кватернионы и системы кватернионов.

Положительные аспекты предлагаемого решения – математика кватернионов проще и более стабильная при реализации, чем углы Эйлера. Однако, это влечет немного большие затраты по производительности.

Наиболее интересное графическое решение было выбрано для генерации освещения в реальном времени. Алгоритм основан на базе классического Shadowmapping с некоторыми доработками для сглаживания теней. Данный алгоритм описывает отрисовку теней для сцены в реальном времени. Алгоритм состоит из двух этапов – генерация карты глубины из направленного источника света и отрисовки сцены от лица камеры с учетом карты затенения.

Первый этап. Вначале сгенерируется карта глубины из источника света. Ниже пойдет речь о направленном источнике света. Для начала необходимо понимать природу теней. Если говорить об одном источнике освещения, то в тени будет все то, что «не видно» с позиции источника света с учетом его направления. Отсюда следует необходимость сгенерировать карту глубины из источника освещения. Таким образом, получаем данные о затенении для последующего рендеринга. Для точечного источника света генерируется CubeMap из шести текстур глубины – верх-низ-право-влево-вперед-назад.

Второй этап. Зная карту глубин и расположение источника света, можно приступать к рендерингу. При отрисовке необходимо учитывать глубину, но, чтобы правильно ее извлечь, необходимо позицию конечного пикселя привести в систему координат источника освещения. Для этого необходимо определить и передать в шейдер матрицу MVP для источника освещения.

После применения матрицы к позиции пикселя получим 2D-координаты в системе освещения. Используя эти координаты, можно получить глубину затенения в этой точке и, если выбранный пиксель окажется с меньшей глубиной относительно источника света, чем сгенерированная ранее глубина в этой позиции, то пиксель будет освещен, иначе – затенен.

Главная идея предлагаемого фреймворка – расширенные возможности по сборке контента.

Изначально определяются несколько отдельных сущностей – Текстура, Шейдер, Меш, Логика обновления, Логика отрисовки, для которых разрабатывается механизм, позволяющий комбинировать все эти сущности между собой и получать на выходе требуемый результат.

Архитектура строится таким образом, что все эти сущности не связаны между собой. Основу разработки составляют несколько, так называемых, Store-хранилищ, в которых содержатся основные данные, заложенные в основу отрисовок. Итак, выделены три основных хранилища – Меши (3D-моди, Mesh), Текстуры и Iейдеры. При необходимости можно добавить также хранилище и для другого типа хранимых объектов. Все хранилища наследуются от базового хранилища, которое является абстрактным и типизируемым, которое также поддерживает целостность данных, защищено от OutOfMemory и других исключительных ситуаций. Рассмотрим указанные сущности.

Mesh. Включается в себя наборы данных о вершинах – текстурные координаты, вектора нормали для каждой вершины и индексы для отрисовки. Для Mesh реализован специальный класс MeshLoader, который загружает их из файла (формат MyGL, разработан специально для движка с целью минимизации данных в файлах описания Mesh), а также собирает Mesh из массивов текстурных координат нормалей и остальных данных. MeshLoader адаптирован под работу с нативными структурами данных, так как структуры данных в Java содержат избыточную реализацию и проблемы, связанные с производительность при их использовании.

Texture. Загружается из форматов png, jpg, gif. Важной особенностью является то, что размеры текстуры должны быть кратны степеням числа 2, так как не все графические адаптеры работают с текстурами других размеров.

Shaders. Класс-обертка над программами, выполняемыми напрямую на GPU. Создаются с использованием ShaderLoader, который компилирует шейдеры из файловой системы и связывает с программой для GPU. Shaders также содержит всю логику по инициализации конкретных шейдеров и предоставляет гибкие возможности для использования и масштабирования.

Логика обновления. Перед тем как отрисовывать объект, необходимо учесть прошедшее время и рассчитать его положение в пространстве, а также другие изменения, которые могли произойти за это время. Для разных объектов реализуется совершенно разная логика поведения. Однако можно выделить общие абстрактные зависимости. Например, объекты должны реагировать на столкновения с другими объектами. Для таких целей разработан CollisionController, который обрабатывает такие события.

Логика отрисовки. Разные объекты необходимо отрисовывать по-разному, учитывая тени, альфа-канал, материал, постобработку, а также их комбинации. В силу этого логика отрисовки вынесена как отдельная сущность, общие черты которой можно выявить и составить эффективную иерархию наследования.

Итак, обобщая все изложенное выше, получаем следующую структуру. Разработанное ядро CoreX содержит в себе информацию о текущей загруженной сцене, а также хранилища для всего контента [1, 2]. Сцена представляется собой набор объектов (комбинаций сущностей), которые «знают», как себя вести за все время своего жизненного цикла. Предлагаемая конструкция позволяет достаточно просто генерировать многообразие объектов, используя абстрактные типы поведения и отрисовки, связывая их с текстурами, мешами и произвольным шейдером для материала. Кроме того, поддерживается этап постобработки, информация для которого может быть сгенерирована в процессе отрисовки конкретных объектов.

Список использованных источников:

1. Ломакин, Г.А. Об одном подходе к построению графических приложений / Г.А. Ломакин // Наука-2011 : сб науч. ст. В 2ч. Ч. 2 / БГУ ; рекол.: В.И. Малюгин (отв. ред.) [и др.]. – Минск : БГУ, 2011. – С. 89-91.

2. Ломакин, Г.А. О разработке веб-среды, поддерживающей создание 3D-приложений / Г.А. Ломакин // Веб-программирование и Интернет-технологии WebConf2012: материалы 2-й Междунар. науч.-практ. конф., 5–7 июня 2012 г., С. 233-234.инск. – Минск Изд. центр БГУ, 2012. – С. 164-165.

ПОСТРОЕНИЕ ИНФРАСТРУКТУРЫ СИСТЕМЫ АВТОМАТИЗАЦИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Мойса Д. С

Серебряная Л.В — к.т.н, доцент

В докладе рассматривается способ построения инфраструктуры поддержки жизненного цикла программного обеспечения, а также необходимые организационные изменения для внедрения такого способа работы с инфраструктурой.

В современных процессах разработки очень важное место занимает инфраструктура поддержки и автоматизации этих процессов, а также сама среда работы продукта. Наиболее часто используемые системы: контроля версий, сборки, непрерывной интеграции, развертывания, тестирования и т. д. Очень важно, чтобы такие системы работали бесперебойно, так как нарушение их работы может негативно повлиять на процесс разработки в целом, а иногда и полностью остановить его. Наиболее частые факторы риска: ошибки конфигурации, неработоспособность систем в течение некоторого времени или их полная неработоспособность.

Ошибки в таких системах часто связаны с человеческим фактором. Чаще всего их настройка и поддержка происходит в ручном режиме, что влечет за собой высокую вероятность внесения ошибки. Также нередки ситуации, когда при смене ключевого сотрудника не происходит передача необходимых знаний новому работнику, и информация о необходимой конфигурации остается потерянной.

При серьезных сбоях возможно полное разрушение системы. В этом случае единственным выходом является настройка этой системы с нуля. Все это время процесс разработки, поддерживаемый поврежденной системой, будет нарушен и могут возникать сложности вплоть до полной остановки разработки. Таким образом, надежность инфраструктуры поддержки процессов разработки является важной и актуальной задачей.

Наиболее часто применяемыми мерами для сохранения и быстрого восстановления систем являются резервные копии, однако восстановление из такой копии может также занять длительное время. Кроме того, копия может быть устаревшей, либо содержать только данные, не включающие информацию о конфигурации.

Представление инфраструктуры в виде кода — это процесс написания сценариев от создания виртуальной машины и установки операционной системы до установки и настройки программного обеспечения, связей между экземплярами и других параметров конфигурации. Среда, описанная сценариями, позволяет применять одну и ту же конфигурацию к одному узлу или нескольким узлам. Автоматизация инфраструктуры вносит гибкость как в процесс разработки, так и в эксплуатацию, потому что любой уполномоченный член команды может изменять сценарии с применением к инфраструктуре надлежащих методов разработки, таких как автоматизированное тестирование и управление версиями. Это значит, что к описанию инфраструктуры можно применять те же метрики эффективности и качества, что и к коду приложений.