О ПОДХОДАХ К РЕАЛИЗАЦИИ ПЛАТФОРМЫ ДЛЯ СОЗДАНИЯ ПРИЛОЖЕНИЙ ПОД ОС ANDROID

Г. А. Ломакин, Л. В. Рудикова
Кафедра программного обеспечения информационных технологий, Учреждение образования «Белорусский государственный университет информатики и радиоэлектороники»
Кафедра современных технологий программирования, Учреждение образования «Гродненский государственный университет имени Янки Купалы»
Минск, Гродно, Республика Беларусь
E-mail: {spellbound.fpmi, rudikowa}@gmail.com

В статье излагаются общие подходы разработки платформы для создания графических приложений на базе мобильной платформы ОС Android. Приводится общая характеристика и метод реализации предлагаемой платфор-мы, связанной с визуализацией трехмерных объектов.

I. Общая характеристика разрабатываемой платформы

Следует отметить, что на текущий момент существует небольшое количество доступных сред для создания интерактивных 3D-визуализаций, к недостаткам которых можно отнести отсутствие хранилища различного 3D-контента в облаке и возможностей, предоставляющих пользователю высокоуровневый подход для работы с 3D-пространством и различными объектами. Поэтому для создаваемой платформы важно выявить следующие требования, связанные с разработкой. Прежде всего, это – определение конкретных рамок и цели создаваемого графического ядра, обеспечение доступа к контенту, реализация набора программ-утилит для использования контента и графического ядра под различные платформы, реализация фреймворка с высоким уровнем абстракций и обеспечение широкого выбора различных алгоритмов для рендеринга изображения. Предлагаемая интерактивная платформа состоит из нескольких компонентов: облака для хранения контента в различных категориях, графического ядра, реализованного с использованием OpenGL, набора утилит приложения для мобильной операционной системы Android, фреймворка, обеспечивающего доступ ко всем возможностям ядра. Таким образом, главными особенностями предлагаемой платформы являются: открытое ядро на OpenGL, набор классов и интерфейсов для рендеринга примитивов, а также набор базовых шейдеров для реализации различных эффектов на графическом конвейере, утилиты для построения визуализаций на ОС Android и синхронизация контента клиентского приложения с сервером.

II. О методологии разработки

Рассмотрим, основные аспекты, связанные с методом разработки и технологиями реализации, а также – с основными конструктивными особенностями предлагаемой интегрированной платформы для разработки 3D-приложений.

Главная идея предлагаемой интегрированной платформы – расширенные возможности по сборке контента, которые аккумулируются в графическом ядре. 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. Метод реализации платформы заключается в разработке универсальной архитектуры, которая строится таким образом, что все эти сущности не связаны между собой. Основу реализации состав-ляют 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 Шейдеры. 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 поразному, 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.

III. Основная математическая база

Одной из основных проблем при разработке 3D-объектов является реализация решения, отвеча-ющего за позиционирование объектов в пространстве. В качестве такого решения были выбраны и доработаны кватернионы и системы кватернионов.

Отметим, что кватернион можно представит в виде матрицы. Так как OpenGL работает с матрицами, то перед их использованием необходимо конвертировать кватернион в матрицу. Таким образом, получаем следующий алгоритм – вся повороты проходят в кватернионах, а затем конвертируются в матрицу, которая передается OpenGL. Основное преимущество такого подхода в том, что при перемножении кватернионов поворачиваются и сами оси и, таким образом гарантируется независимое вращение относительно всех осей. На практике имеем следующее: для конкретного 3D-объекта хранится три скаляра – повороты относительно XYZ . Далее строятся три кватерниона – повороты относительно каждой из оси. После этого кватернионы перемножаются, и получается кватернион конечного поворота, из которого извлекается матрица. В дальнейшем на эту полученную матрицу и будет происходить умножение при формировании матрицы модели. Положительные аспекты предлагаемого решения – математика кватернионов проще и более стабильная при реализации, чем углы Эйлера. Однако, это влечет немного большие затраты по производительности.

IV. Алгоритм ShadowMapping

Наиболее интересное графическое решение было выбрано для генерации освещения в реальном времени. Алгоритм основан на базе классического ShadowMapping с 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 отрисовка 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 пик-сель окажется с меньшей глубиной относительно источника света, чем сгенерированная ранее глубина в этой позиции, то пиксель будет освещен, иначе – затенен.

V. Выводы

Предлагаемая универсальная платформа является альтернативой существующим средам и фреймворкам [1], но без использования единого кода для разных платформ. В качестве средств реализации выбраны: программный стек Android, виртуальная машина Dalvik, СУБД SQLite, язык Java и среда разработки Eclipse. Разрабатываемый продукт предполагается не слишком абстрактным, но позволяет добавлять мало используемые функции. В тоже время, предполагается возможность расширения предлагаемого продукта сторонними разработчиками и выпуск некоторых плагинов и сборок.

Список литературы

1. Ломакин, Г. А. О разработке графического фреймворка для мобильной платформы / Г. А. Ломакин // Наука-2014 : сб. науч. ст. В 2 ч. Ч. 2 / ГрГУ им. Я. Купалы; редкол.: Г. М. Третьяков (гл. ред.) [и др.] – Гродно : ГрГУ, 2014. – С. 81–84.