УДК 000.41+004.921

О РАЗРАБОТКЕ МОБИЛЬНОЙ ПЛАТФОРМЫ 3D-ВИЗУАЛИЗАЦИЙ

Л.В. Рудикова, Г.А. Ломакин

Предлагается общая концепция построения интегрированной платформы для создания визуальных приложений, доступ к которой осуществляется с мобильных клиентов, а также – обобщение подходов, используемых при построении 3D-графики, и абстрагирование конечного пользователя от таких проблем как поиск контента, написание графического ядра и т.д. Описывается общая архитектура разрабатываемой платформы.

Введение

В настоящее время широкое распространение получили несуществующие синтезированные изображения различных объектов. Так, достаточно хорошо известны примеры научной визуализации, моделирования в промышленности (например, создание прототипов автомобилей), спецэффекты для кинематографа, трехмерные игры и т.д.

Значительный научный и практический интерес представляют также и задачи создания виртуальных реальностей, которые отождествляют реальные прототипы, например, визуализация различных памятников архитектуры и археологии. Как правило, подход к решению такого рода задач требует, в основном, интерактивной экранизации.

Экранизация реальных моделей является достаточно сложной задачей. Для экранной визуализации синтезированной 3D-модели используются различные алгоритмы, выбор, адаптация и модификация которых зависит от рассматриваемой задачи предметной области. Более того, чрезмерная трудоемкость создания объектов реального мира приводит к необходимости разработки специальных моделей и новых методов экранизации для таких объектов.

Отметим также, что при современном развитии графических систем и технологий, актуальным является подход, связанный с переводом визуализации 3D-объектов на новый уровень. Разработка 3D-приложений характеризуется определенным набором ограничений, что, естественно, сказывается на результатах визуализации.

Наиболее часто разработчики 3D-приложений обращаются к двум основным библиотекам 3D-графики: DirectX и OpenGL. Однако работа с ними вызывает трудности, связанные, прежде всего, с написанием многих строк кода, использованием API и указателей. Часть проблем по реализации 3D-объектов можно решить с использованием XNA framework, используя соответствующие классы. Однако XNA не позволяет полностью устранить проблемы, возникающие при разработке 3D-приложений, т.к. указанный framework является .NET оберткой DirectX.

С другой стороны, широкое распространение и развитие мобильных платформ также предоставляет необходимые возможности для разработчиков.

Мобильные телефоны уже давно используются не только для разговоров. Они стали выполнять такой широкий спектр компьютерных задач общего профиля, что, вероятнее всего, такие устройства могут стать новым поколением персональных компьютеров.

На сегодняшний день для пользователей имеется широкий выбор мобильных телефонов под управлением Android. В отличие от большинства мобильных систем, закрывающих и ограничивающих разработку, а также развертывание сторонних приложений, Android предлагает альтернативу: позволяет писать приложения, использующие весь спектр современного аппаратного обеспечения, и предоставлять новые возможности по разработке и использованию инструментальных средств.

Таким образом, разработка среды, которая предоставит пользователю возможность быстрого создания 3D-объектов, используя современные Интернет и мобильные технологии, является востребованным направлением в развитии компьютерной графики.

Общая характеристика разрабатываемой платформы и метода реализации

Следует отметить, что на данный 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;
- 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;
- синхронизация контента клиентского приложения с сервером.

Рассмотрим ниже, 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 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 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 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.

Модульная модель универсальной платформы

Проанализировав предметную область, функциональность предлагаемой платформы и учитывая модульную структуру при разработке универсальной платформы для создания 3D-приложений, были выделены отдельные блоки-модули (рисунок 1).

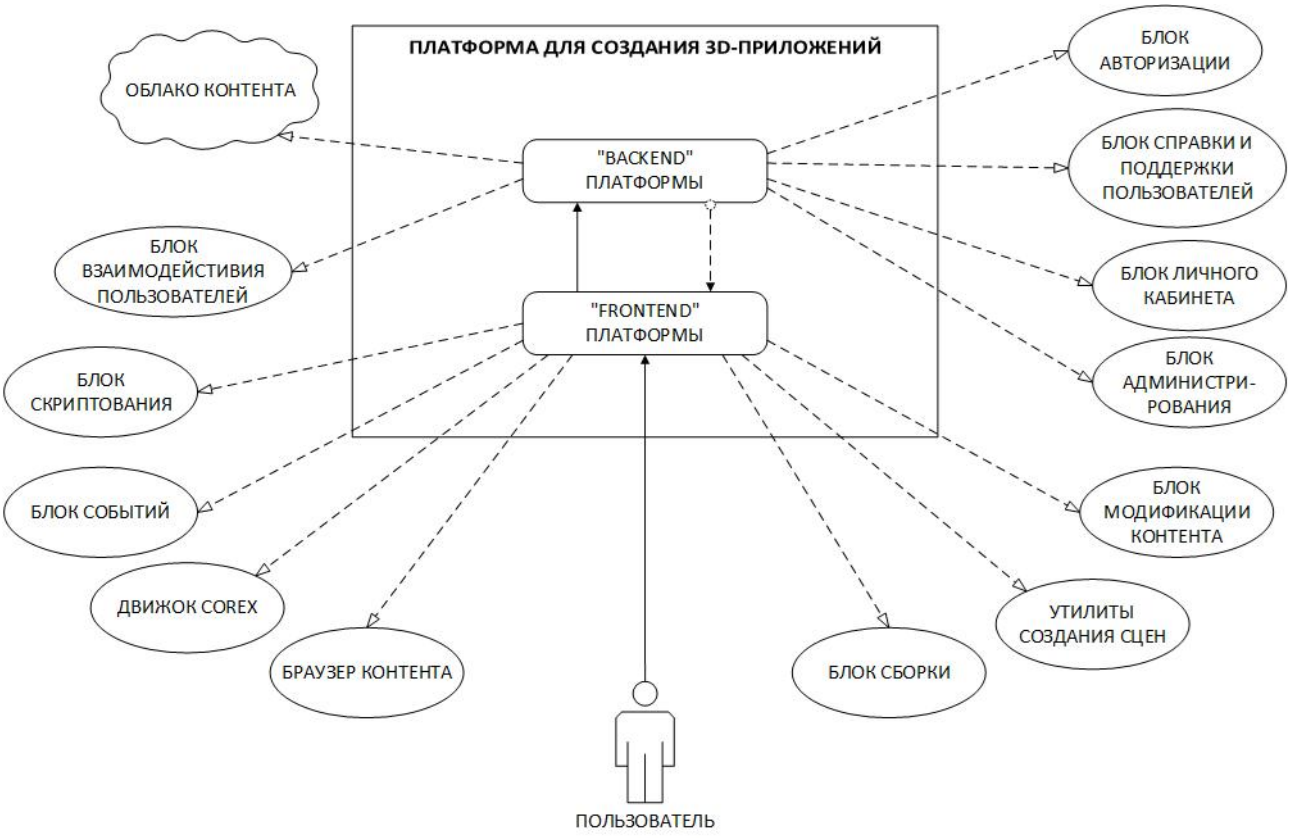

Рис. 1 – Блочно-модульная схема платформы

Охарактеризуем каждый блок.

Блок авторизации. Отвечает за авторизацию пользователей в системе. В данном блоке также определяется роль пользователя и назначаются права. Здесь наблюдается частичная интеграция с блоком личного кабинета.

Блок личного кабинета. В данном блоке пользователь может изменять личные данные.

Блок взаимодействия пользователей. Блок включает в себя обсуждение опубликованного контента.

Блок администрирования. В данном блоке предусмотрена настройка контента. Добавление, удаление, редактирование контента. Учет авторских прав.

Облако контента. Содержит весь добавленный пользователями контент в виде текстуры, модели, звука, музыки и сцены.

Блок справки и поддержки пользователей. Блок поддержки пользователей обеспечивает: возможность двухстороннего общения с пользователями приложений портала, анализ пользовательских предпочтений, исправление проявившихся ошибок или недостоверностей. Предлагаемый модуль организован в достаточно простой для восприятия и использования форме. Воспользовавшись центром поддержки, пользователь может отправить на рассмотрение свой вопрос, замечание или предложение. После этого ответственные за определенные типы обращений люди принимаются за решение возникших вопросов.

Блок скриптования. Описание поведения объекта на сцене.

Блок событий. Связан с блоком скриптования. Реагирование и делегирование событий. Обработка системных событий (нехватка памяти, заряд батареи и т.д.).

Движок GOREX. Отрисовка всех графических объектов в режиме реального времени.

Браузер контента. Позволяет получать контент с сервера, а также добавлять локальный контент и размещать его (при необходимости) на сервере.

Утилиты создания сцен. Основной инструмент взаимодействия пользователя с системой позволяет получать доступ как к сцене, так и ко всем остальным компонентам системы (класс-контроллер).

Блок сборки. Упаковка готовой сцены в независимый объект.

Блок модификации контента. Позволяет комбинировать уже существующий контент для создания нового контента.

В качестве средств реализации предлагаемой платформы на базе ОС Android выбраны следующие средства: программный стек Android [1], виртуальная машина Dalvik, СУБД SQLite, язык программирования Java и среда разработки Eclipse.

Общие концепции реализации утилит для платфомы визуализации 3D-объектов

В любом современном движке должны присутствовать удобные для разработки утилиты. Они могут быть разделены или содержаться внутри среды разработки. В качестве основных утилит разрабатываемой платформы можно выделить следующие.

ContentNavigator – утилита которая позволяет импортировать в движок контент такого типа, как 3D-модели, анимация для них, текстуры, звуки. Кроме того, для каждого вида контента должна быть реализована возможность просмотреть(прослушать), а также изменить базовые свойства. Для 3D-моделей такими свойствами являются масштаб, сдвиг, поворот, материал, тип объекта (статический или нет). Для звука – скорость воспроизведения. Для текстур – сдвиг, масштаб. Также для утилиты ContentNavigator важно иметь базовые редакторы для типов контента. Например, возможность удалять ненужную анимацию у моделей, добавлять Socket в объект. Socket-ом называется позиция

и ориентация в 3D-пространстве относительно модели, в которую может быть добавлена другая модель. Особенностью является привязка сокета к скелету модели, таким образом, при анимировании сокет и интегрированная в него модель будут двигаться зависимо друг от друга. Кроме этого, в утилите должен быть редактор материалов, позволяющий комбинировать текстуры, задавать карты нормалей, задавать атрибуты затенения или блеска и другие эффекты.

LandscapeBuilder – утилита для генерации ландшафтов с помощью карты высот. Такая утилита строит по заданным свойствам, таким как размер ландшафта и коэффициент высоты и, собственно, карты высот, mesh-содержащий ландшафт. В утилите также корректно реализован расчет нормалей и текстурных координат для каждой из вершин. Пример кода, который генерирует ландшафт по карте высот, приведен в листинге 1.

Листинг 1. Пример кода, который генерирует ландшафт по карте высот

```
public class LandscapeBuilder {

    private static final float SCALE = 64.0f;

    private LandscapeBuilder() {

    }

    private static int unsignedToBytes(byte b) {
        return b & 0xFF;
    }

    public static Mesh build(Context context, String fileName) {

        Bitmap bitmap;

        try {
            bitmap = BitmapFactory.decodeStream(context.getAssets().open(fileName));
        } catch (IOException e) {
            return null;
        }

        int width = bitmap.getWidth();
        int heigth = bitmap.getHeight();
        int[][] map = new int[heigth][width];
        ByteBuffer highMap = ByteBuffer.allocateDirect(width * heigth * 4);

        bitmap.copyPixelsToBuffer(highMap);

        for (int i = 0; i < width * heigth; i++) {
            map[i / width][i % width] = unsignedToBytes(highMap.get(i * 4));
        }

        bitmap.recycle();

        float[] points = new float[width * heigth * 3];
        int position = 0;
        int texPosition = 0;
        int normPosition = 0;
        short[] indecies = new short[(width - 1) * (heigth - 1) * 2 * 3];
        float[] texPoints = new float[width * heigth * 2];
        float[] normals = new float[width * heigth * 3];

        float[][] mH = new float[heigth][width];

        for (int i = 0; i < heigth; i++) {
            for (int j = 0; j < width; j++) {
                points[position++] = i;
                points[position++] = map[i][j] / 255.0f * SCALE;
```

```
                points[position++] = j;

                mH[i][j] = map[i][j] / 255.0f * SCALE;

                if (i == 0 || j == 0 || i == heigth - 1 || j == width -1) {
                    normals[normPosition++] = 0.0f;
                    normals[normPosition++] = 1.0f;
                    normals[normPosition++] = 0.0f;
                } else {
                    Vector3 p0 = new Vector3(i, map[i][j] / 255.0f * SCALE, j);

                    Vector3 p1 = new Vector3(i - 1, map[i - 1][j - 1] / 255.0f * SCALE, j
- 1);
                    Vector3 p2 = new Vector3(i - 1, map[i - 1][j] / 255.0f * SCALE, j);
                    Vector3 p3 = new Vector3(i - 1, map[i - 1][j + 1] / 255.0f * SCALE, j
+ 1);

                    Vector3 p4 = new Vector3(i, map[i][j - 1] / 255.0f * SCALE, j - 1);
                    Vector3 p5 = new Vector3(i, map[i][j + 1] / 255.0f * SCALE, j + 1);

                    Vector3 p6 = new Vector3(i + 1, map[i + 1][j - 1] / 255.0f * SCALE, j
- 1);
                    Vector3 p7 = new Vector3(i + 1, map[i + 1][j] / 255.0f * SCALE, j);
                    Vector3 p8 = new Vector3(i + 1, map[i + 1][j + 1] / 255.0f * SCALE, j
+ 1);

                    p1.sub(p0);
                    p2.sub(p0);
                    p3.sub(p0);
                    p4.sub(p0);
                    p5.sub(p0);
                    p6.sub(p0);
                    p7.sub(p0);
                    p8.sub(p0);

                    Vector3 normal = Vector3.cross(p1, p2);

                    normal.add(Vector3.cross(p2, p3));
                    normal.add(Vector3.cross(p3, p5));
                    normal.add(Vector3.cross(p5, p8));
                    normal.add(Vector3.cross(p8, p7));
                    normal.add(Vector3.cross(p7, p6));
                    normal.add(Vector3.cross(p6, p4));
                    normal.add(Vector3.cross(p4, p1));

                    normal.normalize();

                    normals[normPosition++] = normal.getX();
                    normals[normPosition++] = normal.getY();
                    normals[normPosition++] = normal.getZ();
                }

                texPoints[texPosition++] = i / 1.6f;
                texPoints[texPosition++] = j / 1.6f;
            }
        }

        for (int i = 0; i < heigth - 1; i++) {
            for (int j = 0; j < width - 1; j++) {
                indecies[i * (width - 1) * 6 + j * 6] = (short) (i * width + j);
                indecies[i * (width - 1) * 6 + j * 6 + 1] = (short) (i * width
                        + j + 1);
                indecies[i * (width - 1) * 6 + j * 6 + 2] = (short) ((i + 1)
                        * width + j);

                indecies[i * (width - 1) * 6 + j * 6 + 3] = (short) (i * width
                        + j + 1);
                indecies[i * (width - 1) * 6 + j * 6 + 4] = (short) ((i + 1)
                        * width + j + 1);
                indecies[i * (width - 1) * 6 + j * 6 + 5] = (short) ((i + 1)
                        * width + j);
```

```
            }
        }

        GameLandscape.setMapHeigth(mH);
        GameLandscape.setHeigth(heigth);
        GameLandscape.setWidth(width);

        Mesh mesh = new Mesh(points, indecies, texPoints, normals);
        return mesh;
    }
}
```

В данном коде метод build позволяет создать меш с ландшафтом по заданной карте высот.

GlobalLight – утилита, которая отвечает за глобальное освещение и эффекты. Содержит настройку эффектов освещения, глубина, вектор направления и сила света, настройка тумана – дальность, цвет, различные postprocess эффекты такие как blur, motionblur, HDR и др.

AnimationManager – утилита для управления анимацией объектов; позволяет просматривать каждую анимацию и именовать для дальнейшего использования в скриптах.

LequidManager – утилита для симуляции жидкостей. Содержит настройки материала жидкости, такие, как базовый материал, коэффициенты преломления света, прозрачность и искажение цвета света; также возможность задавать симуляцию ветра, а, именно, сила ветра, размер волн (зависит от плотности жидкости).

Кроме того, в предлагаемой платформе при реализации графического ядра была добавлена скелетная анимация, т.е. механизм, позволяющий анимировать не каждую вершину объекта, а конкретные части в зависимости от расположения других частей.

Отметим также, что процесс анимации включает следующие этапы.

1. Прохождение по ключам анимации и нахождение двух ключей, между которыми расположено искомое время. Для каждой кости в матрице Release записывается интерполированное значение.

2. После заполнения Release матрицы всех костей обновляется вся иерархия.

3. Домножение Release на матрицу из bind pose и сохранение её в релизный массив.

4. На уровне шейдеров полученная матрица используется для преобразования вершин и нормалей.

Заключение

Предлагаемая универсальная платформа является альтернативой существующим средам и фреймворкам, но без использования единого кода для разных платформ [2-4]. В структуре движка заложена архитектура и нейминги, которые позволяют перенести среду на другую платформу или, даже, язык программирования с минимальными изменениями. Кроме этого, 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.

Предлагаемая разработка может быть полезна во многих сферах деятельности, связанных с визуализацией. Например, в образовательных или исследовательских целях – визуализации физически химических и др. процессов, в сфере развлечений и маркетинга – реклама и компьютерные игры, а также в промышленной разработке программного обеспечения – использование фреймворка для написания коммерческих приложений.

Список литературы

1. The Developer's Guide // Android [Electronic resource]. – 2010. – Mode of access: https://developer.android.com/guide/index.html. Date of access: 12.12.2014.
2. Ломакин, Г.А. Общие подходы к разработке интегрированной среды для построения графических приложений / Г.А. Ломакин, А.С. Гусейнова // Весник БГУ, Серия 1, 2013. – С.54-60.
3. 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.
4. Ломакин, Г.А. Об общих подходах к разработке графического фреймворка для мобильной платформы / Г.А. Ломакин, Л.В. Рудикова // Информационные технологии и системы 2014 (ИТС 2014) : материалы международной научной конференции, БГУИР, Минск, Беларусь, 29 октября 2014 г. = Information Technologies and Systems 2014 (ITS 2014) : Proceeding of The International Conference, BSUIR, Minsk, 29th October 2014 / редкол. : Л. Ю. Шилин [и др.]. – Минск : БГУИР, 2014. – С. 316-317.

Рудикова Лада Владимировна, заведующий кафедрой современных технологий программирования Учреждения образования «Гродненский государственный университет имени Янки Купалы», кандидат физико-математических наук, доцент, rudikowa@gmail.com

Ломакин Герман Александрович, аспирант кафедры программного обеспечения информационных технологий факультета компьютерных систем и сетей Белорусского государственного университета информатики и радиоэлектроники, spellbound.fpmi@gmail.com