Математические вопросы кибернетики и вычислительной техники 28, 2007, 128 – 134.

Разработка библиотеки трехмерной визуализации для мобильных устройств

Геворг Карапетян

Институт проблем информатики и автоматизации НАН РА
e-mail: gevorgk@gmail.com

Аннотация

Работа посвящена исследованиям в области библиотек трехмерной визуализации и разработке алгоритмов и методов оптимизации для них, с учетом их использования для мобильных устройств.

1. Введение

Индустрия игр для мобильных устройств (мобильных телефонов и КПК) на сегодняшний день является одной из самых динамично развивающихся. Так, например, исследования компании Datamonitor показали, что в ближайшие пять лет суммарная прибыль компаний, вовлеченных в разработку и продвижение игр для мобильных телефонов, вырастет в 18 раз - с $950 млн до $17,5 млрд.
По сравнению с играми для обычных ПК, данная отрасль имеет свой особенности. Во первых мобильные телефоны (или КПК) по своим техническим характеристикам намного отличаются от ПК. Обычно у них достаточно слабый процессор и небольшой объем оперативной памяти. Кроме того у них отсутствуют мощные графические акселераторы, без которых сегодня трудно представить обычный ПК. Однако несмотря на это на сегодняшний день существует достаточно большое количество игр для мобильных телефонов, разработанных различными компаниями. Учитывая тот факт, что игры для мобильных устройств появились сравнительно недавно, на сегодняшний день существует достаточно мало средств и библиотек для разработки игр для мобильный устройств, и каждый производитель вынужден разрабатывать свои собственные библиотеки, а то и вовсе обходиться без них, что, конечно, замедляет процесс разработки игры. Целью данной работы является разработка библиотеки для трехмерной визуализации в играх для мобильных устройств. Библиотека разработана для мобильных устройств с операционной системой Windows Mobile.

2. Структура библиотеки

Разработанная библиотека состоит из следующих модулей

1. модуль визуализации,
2. модуль экспорта сцены из графического редактора 3D StudioMax в формат библиотеки,
3. редактор сцены.

Модуль визуализации обеспечивает собственно трехмерную визуализацию сцены и позволяет управлять отдельными объектами сцены. Алгоритмы и методы трехмерной визуализации использованные в библиотеке будут рассмотрены в следующей главе. Данный модуль также предоставляет функциональность для построения самой сцены, однако учитывая объемность сцены, количество находящихся в ней объектов и их структуру, программное построение сцены слишком неоптимально и бессмысленно. Большинство производителей трехмерных игр создают трехмерные сцены в каком либо трехмерном графическом редакторе (3D Studio Max, Maya и.т.д) и загружают полученную сцену в игру. Однако по техническим причинам формат файлов вышеупомянутых редакторов не подходит для использования в игре, и по этой причине производители игр создают программные модули, для конвертации из формата файлов редактора в свой собственный. Данная библиотека также имеет такой модуль, который подключается к графическому редактору 3D Studio Max и позволяет сохранять созданную сцену в своем формате. Однако после создания трехмерной сцены в нее требуется добавление некоторых элементов специфичных для данной библиотеки, что конечно же невозможно сделать с помощью графического редактора. Для этой цели был разработан редактор сцены, который предоставляет возможность как предварительного просмотра сцены, так и ее редактирования. Заметим, что несмотря на то, что данный редактор разработан для операционной системы Windows 2000/XP, однако для визуализации сцены он опять таки использует разработанный модуль визуализации, т.е. данные модуль работает как для платформы Windows Mobile, так и для Windows 2000/XP.

3. Алгоритмы трехмерной визуализации

Алгоритмов для 3-х мерной визуализации объектов существует достаточно много. Рассмотрим некоторые из них для обоснования выбора конкретной группы алгоритмов для данной библиотеки.

3.1 Метод трассировки луча (Ray tracing [1])

Данные метод работает следующим образом. Через каждую точку на плоскости камеры проводится луч, исходящий из точки наблюдателя, и находится пересечение данного луча с объектами сцены.

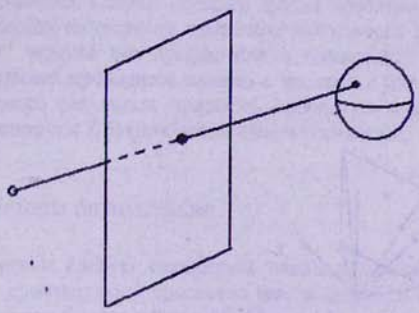

Рис. 1

После того как точка пересечения найдена, соответствующая точка на плоскости закрашивается цветом данной точки. Однако у данного метода существуют свои

недостатки. Во первых, так как с точки зрения наблюдателя объекты сцены могут перекрываться, то надо найти точку пересечения с ближайшим объектом, а для этого надо найти пересечения со всеми объектами сцены, и найти ближайшее из них. Данный процесс содержит большое количество математических вычислений с плавающей запятой и весьма трудоемкий. Учитывая тот факт, что библиотека разрабатывается для мобильных устройств, у которых процессор не имеет модуля для арифметики с плавающей запятой (все вычисления с плавающей запятой производятся программно), то данный метод весьма неэффективен. Даже если его оптимизировать, заменить все операции с плавающей запятой на операции с фиксированной запятой, количество производимых вычислений настолько велико, что никакая оптимизация не поможет достичь желаемой скорости работы библиотеки. Заметим, что мы даже не рассмотрели возможности расчетов освещенности объектов, без которых не достичь реалистичного изображения. Единственным плюсом данного метода является высокое качество полученного изображения, и именно по этой причине он используется в большинстве графических редакторов.

3.2 Ray Casting

Данный метод является несколько усовершенствованным видом предыдущего алгоритма [2], однако он все же не обеспечивает достаточного быстродействия, чтобы быть пригодным для мобильного устройства.

3.3 Метод Z буфера (Z buffer)

Сразу отметим, что данный метод не обеспечивает полного цикла рисования 3-х мерного объекта (рендеринга). Он только дает возможность отображения объектов в правильной последовательности, то есть решает проблему наложения объектов друг на друга [3]. С другой стороны данная проблема порождает большую часть того объема вычислений, из-за которого мы не использовали алгоритм трассировки лучей. Остальная часть трасформации и рисования объектов обеспечивается стандартными математическими методами, коротко описанными ниже.

Как показано на **рис. 2**, 3-х мерный объект находится в мировой системе координат, но кроме этого он имеет свою собственную систему координат. Конечное изображение проецируется на плоскость камеры, которая тоже имеет свою систему координат.

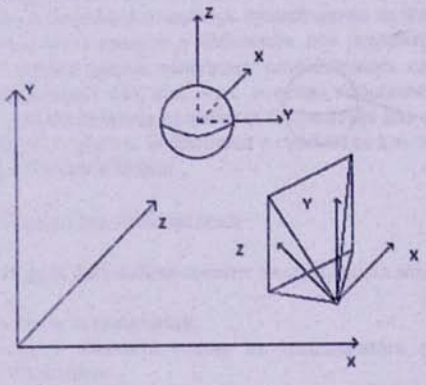

Рис. 2.

Теоретически процесс получения конечного изображения объекта достаточно прост: следует перевести координаты всех вершин объекта из локальной системы координат в мировую систему координат, после чего перевести в систему координат камеры, спроецировать все вершины на плоскость камеры и нарисовать проекцию объекта на плоскости. Перевод координат из одной системы в другую осуществляется умножением координат вершин объекта на соответствующие матрицы преобразований. Более детально данный процесс описан в [3]и [4].

На данный момент нас интересует решение проблемы наложений проекций объектов. Во время рендеринга конечного изображения объекты должны отрисовываться в правильной последовательности – от более близких к более дальним. Алгоритм Z буфера предлагает следующее решение. Во время отрисовки каждой точки объекта значение координаты Z данной точки запоминается в специальном буфере (отсюда и название – Z буфер). После этого перед закрашиванием каждой точки на экране, значение координаты Z данной точки сравнивается со значением сохраненном в буфере. Если значение координаты меньше значения сохраненного в буфере (т.е данная точка находится ближе к камере) точка выводится на экран, в противном случае она не выводится. Остается одна проблема – определить координату Z для каждой точки объекта. Ведь когда объект спроецирован на плоскость камеры, он уже находится в 2-х мерном пространстве и определить 3-ю координату не удастся. Однако решение данной проблемы достаточно простое – после проецирования вершин объекта мы сохраняем координату Z каждой вершины. Здесь следует отметить, что объект представляется в виде множества треугольников, соединенных между собой ребрами [3]. Соответственно при отрисовке треугольника, если мы имеем координату Z для каждой вершины треугольника, координаты Z всех остальных точек получаются при помощи линейной интерполяции. Однако в при линейном изменении координат x и y линейно изменяется не координата Z а ее обратная величина [1], следовательно при линейной интерполяции мы вычисляем обратную величину данной координаты.

Если сравнить данный метод рендеринга с алгоритмом трассировки луча, сразу видно, что в отличие от метода трассировки луча, где для отрисовки каждой точки объекта проводятся достаточно сложные и трудоемкие вычисления, данный метод использует достаточно менее трудоемкие вычисления (умножение матриц и линейную интерполяцию). Исходя из этого данный метод был взят за основу нашей библиотеки визуализации. Однако осталась другая проблема – несмотря на сравнительно малое количество вычислений процессор мобильного устройства достаточно слабый и не имеет модуля для арифметики с плавающей запятой, а так как большая часть вычислений проводится именно с числами с плавающей запятой, то для достижения желаемого результата пришлось прибегнуть к оптимизации технического характера (сам алгоритм Z буфера достаточно оптимален).

4. Методы оптимизации

Для начала следует определить некоторые количественные величины, по которым будет производиться сравнение быстродействия библиотеки. Так как общепринятым параметром быстродействия библиотек 3-х мерной визуализации является количество кадров отрисованых за секунду (FRS – frames per second), мы тоже будем придерживаться этого стандарта. Кроме того сравнение должно производиться на 3-х мерных сценах, которые имеют одинаковое количество полигонов (все 3-х мерные объекты в сцене представляются при помощи полигонов - трегольников). Для

сравнения была взята сцена, содержащая 3000 полигонов. Без применения каких либо методов оптимизации производительность разработанной библиотеки была равна 4-5 FPS.

Рассмотрим по очереди методы оптимизации примененные в нашей библиотеке.

4.1 Арифметика с фиксированной запятой

Так как операции с плавающей запятой являются одним из самых медленных, то было решено начать проводить оптимизацию именно для этой проблемы. Было решено все операции над числами с плавающей запятой заменить на числа с фиксированной запятой. Число с фиксированной запятой было представлено следующим образом: так как процессор мобильного устройства 32-х разрядный, то первые 16 разрядов были отведены для представления целой части числа, а остальные 16 разрядов для преставления дробной части. В результате все операции с числами представляются следующим образом. Предположим мы имеем 2 числа с фиксированной запятой – *a* и *b*. Результат операций сложения, умножения, вычитания и деления *c* будет иметь следующий вид

1. сложение *c=a+b*
2. вычитание *c=a-b*
3. умножение *c=(a*b)/65536*
4. деление *c=(a*65536)/b*

Умножение и деление на 65536 (2^{16}) объясняется так: Для представления числа с плавающей запятой в виде числа с фиксированной запятой данное число умножается на 65536 (2^{16}) (ведь для представления дробной части числа мы выделили 16 разрядов) и округляется. Следовательно при умножении получится *(b*65536)*(b*65536)*, и для получения правильного результата результат умножения нужно разделить на 2^{16}. Аналогично, при делении результат нужно умножить на 2^{16}.

Однако это еще не все. Известно, что операция деления намного медленнее операции умножения. А так как после умножения мы делим результат на степень двойки, то вместо этого мы можем применить логическую операцию сдвига. Окончательно операция умножения примет вид *c=(a*b)>>16.*

После проверки полученного результата оказалось, что быстродействие повысилось почти в 4 раза, а именно до 15-16 FPS. То есть вышеописанный метод дал достаточно большой прирост производительности.

4.2 Табличные вычисления математических функций.

Библиотека визуализации для рендеринга объектов производит некоторые трансформации над объектами (например перевод координат из одной системы координат в другую), где используются некторые математические функции, такие как синус, которые вычисляются при рисовании каждого кадра и значительно замедляют процесс рендеринга. Было решено заменить вычисление подобных функций на таблицу дискретных значений для данной функции. Так, например для функции синуса хранится таблица значений данной функции для 360 значений параметра. Кроме того, как показали тесты, операция деления тоже является достаточно медленной. Для операции деления тоже была создана соответствующая таблица, хранящая значения от 1 до 1/65536, и большинство операраций деления, не требующих

большой точности приняли следующий вид: $\frac{a}{b}$=>a*$\frac{1}{b}$, где значение $\frac{1}{b}$ берется из заранее подготовленной таблицы. Отметим, что значения данных таблиц вычисляются при загрузке библиотеки, что никак не влияет на быстродействие библиотеки во время процесса рендеринга.

После данной стадии оптимизации быстродействие библиотеки повысилось до 24 FPS. Естественно данные методы имеют и свои минусы. Так, например арифметика с фиксированой запятой ограничивает возможные размеры 3-х мерной сцены в пределах от -65536 до +65536 по всем трем координатам. Однако для игр в небольших или замкнутых пространствах это приемлимо. Кроме того вследствии данной арифметики теряется точность вычислений и появляются некоторые погрешности, как например некоторая размытость текстур наложенных на объекты и некоторые видимые дефекты после проецирования перекрывающихся объектов, что является следствием неточного вычисления обратной величины координаты Z для точек объектов. Первая проблема решается сугубо дизайнерским подходом при помощи предварительной обработки текстур графическими редакторами. Для решения второй проблемы было решено увеличить точность чисел для вычисления обратной величины координаты Z. Так как во время рендеринга используется только обратная величина данной координаты ($\frac{1}{Z}$) [3], очевидно, что данная величина может изменяться в пределах (0, 1] (в алгоритме не используются значения Z меньшие чем 1). В соответствие с этим было принято решение повысить точность для данных вычислений, выделив для дробной части числа 30 разрядов вместо 16, оставив один разряд для знака и один для целой части. После этого качество изображения достигло достаточно приемливого уовня.

5. Результаты

Создана библиотека для 3-х мерной визуализации, На основе данной библиотеки разработана 3-х мерная игра, при разработке которой были проверены качественные характеристики и удобство использования данной библиотеки, были устранены некоторые недочеты и проведены доработки. Кроме того было проведено сравнение данной библиотеки с доступными на данный момент библиотеками.

Ниже приведена таблица результатов сравнения с двумя из них.

Название библиотеки	Кол-во полигонов в сцене	Быстродействие (fps)
EdgeLib (http://www.edgelib.com/)	3000	10
Microsoft DirectX for Mobile devices	3000	15
MP3DEngine	3000	24

Правда, по качеству изображения разработанная библиотека уступает первым двум, однако как было отмечено ранее, качество изображения достаточно приемлимое для разработки коммерческих игр. Кроме того разработка библиотеки продолжается и ведется работа над улучшением ее качественных характеристик.

Литература

[1] http://en.wikipedia.org/wiki/Raytracing
[2] http://en.wikipedia.org/wiki/Ray_casting
[3] А. Ламот, Программирование трехмерных игр для Windows, изд. Вильямс, 2006 г.
[4] Е. А. Никулин, Компьютерная геометрия и алгоритмы машинной графики, изд. БХВ-Петербург, 2005 г.

Եռաչափ տեսանելիացման գրադարանի մշակում շարժական սարքերի համար

Գ. Կարապետյան

Ամփոփում

Աշխատանքը նվիրված է եռաչափ տեսանելիացման գրադարանների հետազոտմանը և դրանց համար ալգորիթմների և օպտիմալացման միջոցների մշակմանը՝ հաշվի առնելով նրանց օգտագործումը շարժական սարքերում: Շարժական սարքերի տակ այստեղ ի նկատի ունենք ժամանակակից բջջային հեռախոսները և անձնական թվային օգնականները (Personal Digital Assistant):

Աշխատանքում բերված են եռաչափ օբյեկտների ներկայացման մոդելները, նրանց տեսանելիացման ալգորիթմները և նշված են շարժական սարքերի համար դրանց իրականացման հիմնական բարդությունները և հիմնախնդիրները: Տրված են լուծումներ ընտրված ալգորիթմների օպտիմիզացիայի համար և ներկայացված է իրականացված եռաչափ գրադարանի ընդհանուր կառուցվածքը: Գրադարանը իրականացված է C++ լեզվով Windows Mobile 4.2.x/5.0 օպերացիոն համակարգի ղեկավարմամբ աշխատող սարքերի համար: