КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2009 Т. 1 № 2 С. 119–127

Ки&М

МАТЕМАТИЧЕСКИЕ ОСНОВЫ И ЧИСЛЕННЫЕ МЕТОДЫ МОДЕЛИРОВАНИЯ

Визуализация 3D-сцен. Способы контроля и хранения ресурсов

Я. М. Русанова[1,a], М. И. Чердынцева[1]

[1] Южный федеральный университет,
факультет механики, математики и компьютерных наук,
344090, г. Ростов-на-Дону, ул. Мильчакова, д. 8-А

E-mail: [a] dem@math.rsu.ru

Получено 20 марта 2008 г.,
после доработки 17 июня 2008 г.

Основные задачи, представленные в этой статье, — это описание и хранение всей информации, необходимой для визуального представления объектов. Разрабатываемая технология хранения и контроля ресурсов может применяться для визуализации трехмерных сцен в режиме реального времени. Были использованы средства Sample Framework, предоставляемые DirectX SDK, и библиотека Direct3D Extension Library (D3DX).

Ключевые слова: 3D-визуализация, хранение информации

Visualization of three-dimensional scenes. Technology for data storing and manipulating

Ya. M. Rusanova[1], M. I. Cherdyntseva[1]

[1] *Southern Federal University, Faculty of mathematics, mechanics, and computer science, Milchakova str. 8-A, Rostov-on-Don, 344090, Russia*

Abstract. — This article is devoted to some problems of declaring and storing information for objects' visualization. The storage structure and resources control technology can be applied for real-time visualization of three-dimensional scenes. Such instruments as Sample Framework from DirectX SDK and Direct3D Extension Library (D3DX) were used in the implementation

Key words: 3D visualization, information storing

Citation: *Computer Research and Modeling*, 2009, vol. 1, no. 2, pp. 119–127 (Russian).

Введение

Рассмотрим задачу по разработке программных модулей, реализующих технологию хранения и контроля ресурсов для визуализации трехмерных сцен. Эти модули предназначены для использования в качестве основы для создания обучающей программы, которая будет иметь вид трехмерной компьютерной игры на открытом пространстве. В проектируемых модулях необходимо реализовать функциональность графического конвейера, а также разработать базовые классы для представления ландшафта и динамических ландшафтных объектов. Учитывая возможность использования средств DirectX для работы с моделями, объекты окружения, т. е. деревья, кусты, камни, тривиальны в реализации. Ландшафт же с учетом его особенностей требует особого подхода при построении и позволяет управлять его геометрическими данными с целью увеличения скорости его визуализации.

Структура модулей

Для решения поставленной задачи по реализации графической части игрового проекта был выбран подход, который подразумевает, что управление и контроль над работой игрового приложения осуществляются при помощи так называемых менеджеров ресурсов и устройств. Менеджер ресурсов — это компонент, назначение которого доставлять всему проекту любую информацию или, иными словами, заботиться о ее доступности в памяти без участия других модулей. Менеджер устройства — это компонент, отвечающий за связь приложения с соответствующим устройством. К примеру, менеджер видеоустройства отвечает за конвейер рендеринга, менеджер устройств ввода (мышь, клавиатура, джойстик) — за пользовательский ввод, менеджер аудиоустройства — за воспроизведение звуков и музыки. Поскольку разрабатывается только графическая часть, то основной тип используемых ресурсов — это видеоресурсы, и работа осуществляется только с видеоустройством. Но в ходе дальнейшей разработки возможно внедрение менеджеров других устройств. Эти самодостаточные управляющие объекты требуют наличия общего компонента как средства доступа к ним и взаимодействия с ними со стороны приложения. Для выполнения данных функций создается ядро приложения. Поскольку в ходе расширения проекта возможно добавление новых компонент и изменение существующих, в том числе и самого ядра, то оно должно быть спроектировано таким образом, чтобы новые возможности, включаемые в проект, не конфликтовали с основными его частями, и не было бы необходимости на каждом этапе кардинально менять структуру проекта [1]. Схема взаимодействия всех модулей представлена на рис. 1.

При построении модели управления ресурсами следует учитывать несколько аспектов. Во-первых, исходя из условий работы приложения, необходимо выбрать способ хранения данных. Во-вторых, поскольку проектировать для каждого типа ресурсов отдельный менеджер неэффективно, то необходимо создать единый интерфейс для управления всеми видами ресурсов. В-третьих, следует учитывать, что один ресурс может использоваться совместно многими объектами и возникает проблема контроля ссылок на ресурс при его создании и освобождении. Для решения этих проблем в проекте создаются классы для работы с пулами данных, отдельными ресурсами и менеджерами однотипных ресурсов, а также с общим для всех ресурсов менеджером.

Помимо управления ресурсами и устройствами, для любого игрового приложения возникает задача построения сцены, т. е. размещения объектов в пространстве и изменения их положения при действиях пользователя или автоматически по заданному алгоритму. При создании графического приложения разработчик может включать в сцену множество различных визуальных объектов. Прежде всего, это ландшафт или потолок и стены, ограничивающие всю сцену. Затем добавляются какие-то динамические объекты, то есть те, которые могут со временем изменять свои свойства. Таким образом, общая для всех объектов сцены информация включает в себя данные о положении в пространстве, размерах, а также методы визуализации и обновления объектов.

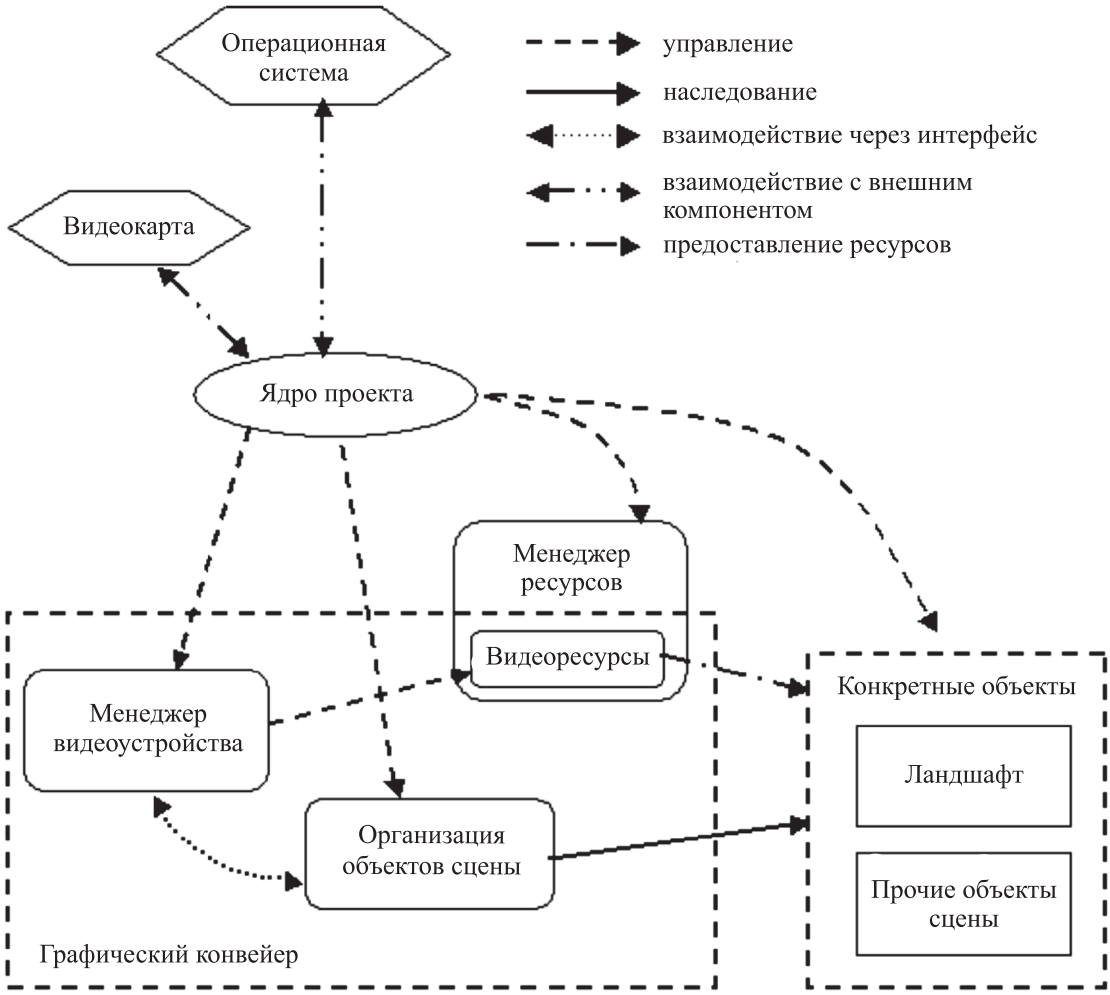

Рис. 1. Схема взаимодействия модулей

Для описания объектов необходимо спроектировать некоторый базовый класс, содержащий эту информацию. Это позволит всем объектам взаимодействовать с другими компонентами приложения и между собой по единому интерфейсу и, тем самым, свести к минимуму потребность в изменении всего проекта при добавлении в сцену новых элементов. В дальнейшем все классы моделей будут порождаться от этого базового класса путем добавления новых, специфических свойств, а также реализации виртуальных методов интерфейса, осуществляющих обновление и визуализацию объектов.

После построения сцены возникает основная для графического конвейера задача — визуализация или рендеринг сцены. В этом процессе, так или иначе, задействованы все вышеперечисленные компоненты. Прежде всего, требуется выделить видимые объекты пространства. Чтобы ускорить поиск этих объектов, пространство разбивается на участки. В данном проекте для разбиения пространства используется такая структура, как квадрадерево. Для построения квадрадерева следует предусмотреть специальные классы и способы их взаимосвязи с организацией объектов сцены.

Далее, видимые объекты могут последовательно выводиться во внеэкранный буфер. Но для того чтобы сократить временные затраты, которые требует данный процесс, надо осуществлять за ним контроль, для чего создается специальный объект — очередь на рендеринг. Этот компонент является частью менеджера видеоустройства. При рендеринге объекты передают очереди свои параметры, благодаря чему появляется возможность изменить порядок отображения объектов и уменьшить количество операций смен состояния видеоустройства.

Организация управления ресурсами — создание пулов данных

Один из основных аспектов эффективного управления памятью — ограничение общего числа операций ее выделения. Простейший метод добиться этого состоит в объединении схожих объектов в группы для выделения большего объема пространства. Такие группы, как правило, называют пулами данных. Самый простой способ организации хранения данных — разместить в памяти большой массив из объектов и ссылаться на них по индексам в этом массиве. Но данный подход лишен гибкости. Размер такого массива должен быть известен до запроса его первого члена, и он не сможет стать больше начального количества размещенных в памяти элементов, что ограничивает общее число объектов, которые можно использовать в процессе игры. Этот подход идеален только для тех приложений, в которых количество всех объектов известно. Однако более-менее масштабное игровое приложение требует большей гибкости при хранении данных.

Чтобы достигнуть компромисса между эффективным расходом памяти и переменной численностью объектов, объекты, хранящиеся в пуле данных, разбиваются на группы. Каждая из групп содержит фиксированное число элементов и может добавляться или удаляться из пула при необходимости. Объекты, которые находятся в пуле, считаются его членами и по запросу передаются вызывающей подпрограмме при помощи дескрипторов. Члены пула могут запрашиваться и освобождаться клиентами. При необходимости пул может расти, увеличиваясь в размере на фиксированный шаг для размещения в нем большего числа членов.

Для работы с пулами данных предназначен класс cDataPool, группа объектов в пуле данных описывается классом cPoolGroup. Эти классы реализованы как шаблонные, т. е. могут использоваться для хранения объектов любого типа. Но при этом, как было отмечено выше, для каждого типа используется свой пул. Схема пула данных представлена на рис. 2.

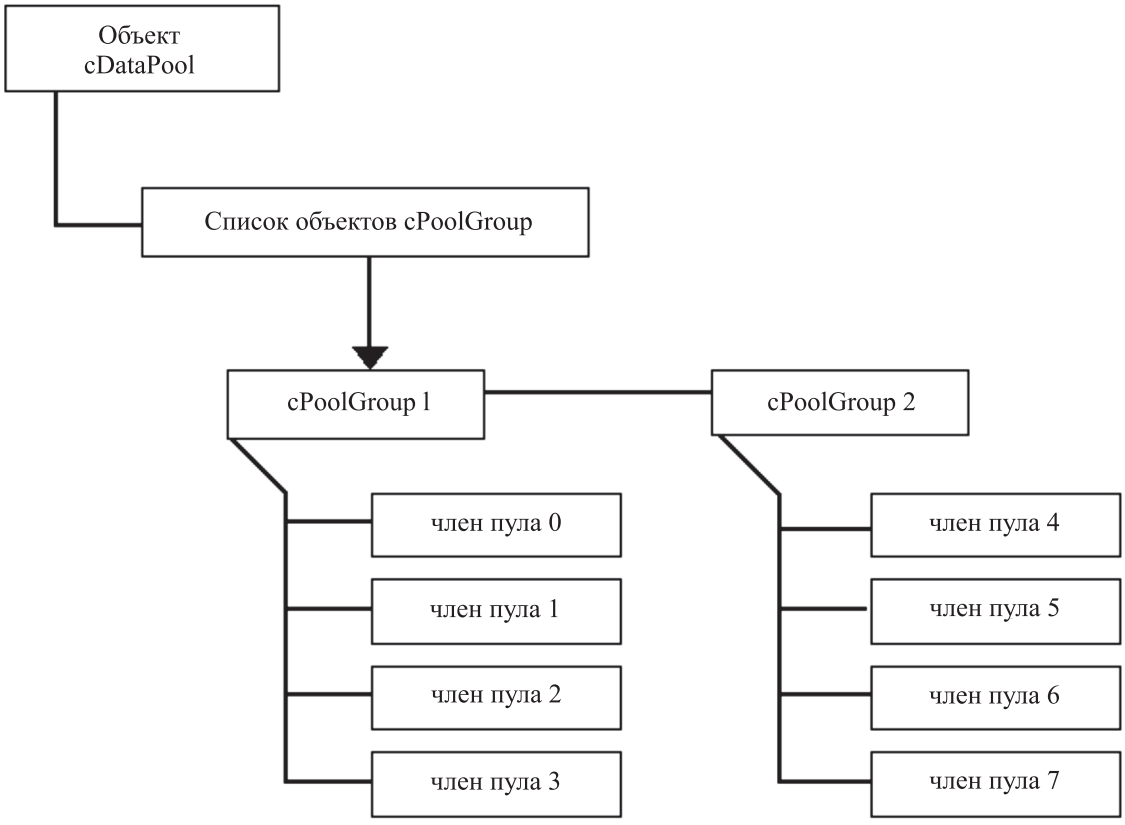

Рис. 2. Схема пула данных, используемого в проекте

Каждая группа cPoolGroup содержит массив элементов, количество свободных мест, массив номеров свободных мест и номер первого свободного места. В массиве для каждого свободного члена группы хранится номер следующего свободного члена. Последний свободный элемент содержит свой собственный индекс, что является признаком конца данной цепи. Каждый используемый член характеризуется константой, хранящейся в этом массиве и свидетельствующей о том, что данный член недоступен. Класс cPoolGroup также содержит методы добавления в группу, удаления из группы члена и выдачи элемента по его номеру.

В каждой группе конкретного пула содержится фиксированное число элементов. Для добавления в пул нового члена используется метод, который сначала проверяет, есть ли свободные места в существующих группах, и если нет, то создает новую группу. Этот метод возвращает дескриптор на член пула, по которому внешние подпрограммы могут обращаться к данному элементу. По дескриптору можно определить номер группы, которой принадлежит элемент, и номер элемента в этой группе. Таким образом, доступ к членам пула осуществляется, как в обычном массиве, по номеру.

Организация объектов сцены

Базовые классы. Любой объект характеризуется в первую очередь своим положением в пространстве. Однако описывать каждый объект в общей системе координат неудобно, поскольку, как правило, объекты располагаются на сцене не поодиночке, а группами, причем в группе может прослеживаться иерархия: положение «младших» объектов зависит от положения более «старших».

Таким образом, целесообразно при описании объектов учитывать эту иерархию, что подразумевает использование дерева в качестве основной структуры для хранения объектов. Схема одного уровня такого дерева изображена на рис. 3.

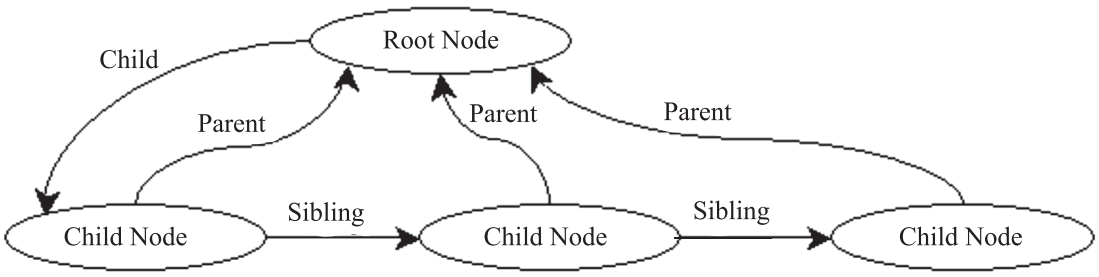

Рис. 3. Схема дерева узлов сцены

Для организации трехмерной сцены, прежде всего, был спроектирован и реализован класс узла сцены — cSceneNode. Узел cSceneNode формирует особую систему координат в трехмерной среде. Такие узлы могут быть связаны воедино в иерархии «родитель – потомок», породив всеобщий граф среды сцены.

В каждом объекте класса cSceneNode содержится ссылка на родительский узел — parent. Но для организации обхода по дереву, допустим при визуализации или обновлении сцены, необходима также обратная связь. Поэтому все дочерние узлы, имеющие общего родителя, формируются в список, а родительский узел хранит указатель на первый элемент этого списка.

Ниже представлены основные поля и методы класса cSceneNode:

— ссылка на родительский узел;

— ссылка на соседний узел;

— ссылка на дочерний узел;

— локальная матрица трансформации — относительно родителя;

— глобальная матрица трансформации;

— ориентация (сдвиг, повороты, масштаб) относительно родителя;

— интерфейс для конвейера рендеринга — реализуется потомками.

Для описания положения узла сцены в относительной системе координат используется специальный класс cOrientation. В своих полях этот класс содержит трансформации родительской системы координат: масштаб, сдвиг, изменение направления осей координат (повороты). Методы данного класса позволяют устанавливать новые значения этих трансформаций или же изменять текущие значения, а также строить матрицу преобразования.

При помощи поля m_Inheritance можно определять способ наследования трансформаций родительского узла. Это поле может принимать значения из следующего перечисления:

— при наследовании учитывается только сдвиг;

— при наследовании учитываются только поворот и масштаб;

— при наследовании учитываются все изменения.

На основании данных, хранящихся в объекте m_orientation, строится локальная матрица трансформации m_localMatrix. Эта матрица используется при промежуточных расчетах. Глобальная матрица трансформации m_worldMatrix, по которой определяется положение объекта в общем пространстве и которая используется при рендеринге объекта, получается из локальной путем ее умножения слева на глобальную матрицу родителя узла. Матрица, обратная глобальной матрице, m_inverseWorldMatrix применяется, когда необходимо поменять родителя данного узла и, соответственно, пересчитать локальную матрицу.

Класс cSceneNode предназначен лишь для построения всеобщего графа сцены, он не содержит никаких геометрических данных о визуальных элементах сцены. Однако, поскольку довольно часто встает вопрос о видимости какого-либо объекта или столкновении этого объекта с другими, то необходимо иметь какие-то сведения об объеме, занимаемом геометрией объекта. Обычно в роли таких сведений выступают ограничивающий параллелепипед или сфера. Для представления всех визуальных элементов сцены предназначен класс cSceneObject. Этот класс является потомком класса cSceneNode и дополняет его полями и методами для хранения и расчета границ объекта.

Для определения ограничивающего параллелепипеда применяется специальный класс cRect3d. Этот класс задает прямоугольный параллелепипед двумя точками пространства, которые не принадлежат одной грани параллелепипеда, а также предоставляет методы для изменения размеров и положения фигуры.

Локальные границы mlocalBounds, т. е. заданные в системе координат данного объекта, рассчитываются при построении конкретной модели. В частности, для ландшафта такие расчеты производятся вручную, а для объектов, использующих в качестве моделей меши, применяется стандартная функция DirectX D3DXComputeBoundingBox. Границы параллелепипеда в пространстве, общем для всей сцены, хранятся в поле m_worldBounds и рассчитываются на основании локальных границ путем применения глобальной матрицы трансформации данного объекта. Эти границы необходимо пересчитывать после изменения положения объекта, для чего предназначен метод recalcWorldBounds.

Каждый объект может быть частично или полностью проницаемым при столкновении с другими объектами, поэтому помимо ограничивающего параллелепипеда также вводится дополнительно поле m_localPermeableBounds, содержащее сведения о действительных границах объекта. Эти границы необходимо задавать вручную, по умолчанию они совпадают с границами видимости. Для того чтобы отделить не требующие проверки на столкновение, добавлено поле m_bPermeable. Если оно имеет значение true, значит, объект проницаем полностью.

Имея в своем распоряжении базовый класс, содержащий матричное преобразование и выровненный по осям ограничивающий прямоугольник объекта, можно построить любой характерный элемент в составе графического приложения. Также класс cSceneNode предоставляет для конвейера рендеринга общий интерфейс, который может использоваться для обработки объектов на каждом его шаге.

Квадрадеревья

Игровое пространство в целом, как правило, включает в себя значительное число объектов. Рендеринг такого количества объектов может занять довольно много времени, и при перемещении камеры сцена не будет обновляться с соответствующей скоростью. Между тем выводить на экран требуется лишь объекты, попадающие в поле зрения камеры. Поэтому, прежде чем осуществлять рендеринг сцены, необходимо выделить видимые объекты. В качестве пространства для проверки на пресечение с полем зрения камеры берется его ограничивающий параллелепипед. Проверка каждого по отдельности объекта на видимость опять же может занять значительный промежуток времени ввиду большого числа объектов. Поэтому более целесообразно разбить все пространство на большие участки (сектора) и вначале выделить видимые области пространства, а затем уже проверять только те объекты, которые лежат в этих областях. В данном проекте для разбиения пространства используется такая структура, как квадрадерево.

Квадрадеревья рассчитаны на применение в двумерном случае. Для трехмерного случая логически правильно использовать октадеревья. Но в данной ситуации объекты игрового пространства разбросаны в горизонтальной плоскости, по вертикали же ландшафт и объекты, на нем расположенные, занимают сравнительно мало места. Поэтому более целесообразно отказаться от обычного разбиения пространства по вертикали и строить квадрадерево, рассматривая вместо объемных фигур их проекции на горизонтальную плоскость. Для анализа третьего измерения все пространство разбивается по вертикали на 32 слоя, и затем к каждому из объектов cSceneObject добавляется 32-битное поле m_quadTreeZMask. В этом поле для каждого слоя пространства устанавливается соответствующий бит в том случае, когда объект пересекается с этим слоем. Следовательно, при помощи этого 32-разрядного числа m_quadTreeZMask можно определить, в каких слоях располагается объект. Для каждого узла квадрадерева также вводится подобное поле m_zMask. По мере добавления объектов в квадрадерево, к битовым полям m_quadTreeZMask объектов применяется операция логического сложения or, и ее результат запоминается в битовом поле m_zMask. Это битовое поле характеризует содержимое данного узла. При этом соответствующие битовые поля m_zMask в родительских узлах рассчитываются как логические суммы битовых полей m_zMask дочерних узлов и битовых полей m_quadTreeZMask объектов, принадлежащих непосредственно этому узлу.

Поиск объектов в расширенном квадрадереве производится следующим образом. Сначала нужно задать прямоугольник, который характеризует положение пространства поиска в горизонтальной плоскости, а также битовое поле слоев высоты, в которых находится пространство поиска. Далее поиск производится по всем уровням дерева, причем для более младших уровней поиск производится только среди тех узлов, предки которых имели пересечение с пространством поиска. Для определенного узла дерева, проекция на горизонтальную плоскость которого имеет пересечение с прямоугольником, задающим пространство поиска, вначале при помощи побитовой операции and производится сравнение битового поля пространства поиска с битовым полем m_zMask узла дерева. Это сравнение позволяет сразу же узнать, находятся ли члены узла в исследуемых слоях. Если были обнаружены слои, в которых может произойти пересечение объектов узла с областью видимости, то производится дальнейшая проверка объектов.

При операциях добавления и поиска в расширенном квадрадереве для третьего измерения используются только логические операции. Поэтому этот процесс не вызывает таких больших издержек по времени, как в случае с полноценным разбиением пространства по вертикали для построения октадерева.

Также следует отметить, что в данном проекте используется вполне развернутое представление квадрадерева, что предполагает постоянное хранение всех узлов дерева, в том числе и пустых. Довольно часто квадрадеревья создаются так, чтобы включать в них минимальное число узлов. Если ветвь дерева может быть признана пустой (ниже нее в дереве нет ни одного объекта), то все узлы в этой ветви можно отбросить. Эта стратегия предполагает минимум места для хранения дерева, так как пустые узлы не расходуют память.

Однако поскольку объекты в процессе игры могут перемещаться, то приходится заботиться о динамическом создании и уничтожении узлов. Для размещения вполне развернутого представления квадрадерева требуется фиксированный объем памяти, что позволяет применить нетривиальную схему управления деревом. Для добавления объектов используется не рекурсивный алгоритм, а прямой доступ к узлам дерева по методу, описанному Мэттом Притчардом. Этот метод использует природу логической операции «исключающее ИЛИ» (XOR), а также тот факт, что каждый узел квадрадерева делится по центральной точке каждой оси. Если входящие в состав квадрадерева сектора сохранить как целые значения степени числа 2, то вычисление XOR для границ любого объекта ведет к образованию битового шаблона, который может быть протестирован для отыскания нужного уровня дерева с целью размещения в нем объекта.

При работе с квадрадеревом горизонтальная проекция игрового пространства с помощью операций сдвига и масштабирования преобразуется к квадрату размера 2^nx2^n, где n — глубина квадрадерева. Соответственно, при вставке в квадрадерево те же операции производятся над координатами объекта. Также считается, что координаты могут иметь только целые значения, поэтому вещественные числа необходимо округлять. Рассматривается прямоугольник, ограничивающий проекцию объекта на горизонтальную плоскость.

Уровень дерева, на который следует вставить объект, определяется по следующему алгоритму:

1. Применяется операция XOR к наименьшей и наибольшей x-координатам объекта. Полученный результат рассматривается как битовый шаблон. rx — позиция старшего единичного разряда. Запоминается число $levx = n - rx$.
2. Аналогичные действия производятся для y-координат. Запоминается число $levy = n - ry$.
3. Номер уровня берется как максимальное значение из чисел $levx$ и $levy$. Самый верхний уровень дерева, включающий в себя все пространство, считается нулевым.

После того как искомый уровень дерева стал известен, определяется, какой именно узел данного уровня является родителем проверяемого объекта. Если все сектора на данном уровне узлов дерева хранятся как двухмерный массив, то нужный сектор можно найти, взяв координаты проекции объекта и приведя их к масштабу сетки узлов данного уровня дерева, т. е. к диапазону $[0, 2^{lev} - 1]$ по каждой оси, что и позволит найти индексы столбца и строки для размещения в них объекта. Это преобразование осуществляется при помощи операция сдвига вправо на число $n - lev$, что практически равносильно делению на 2^{n-lev}.

Все побитовые операции (XOR, СДВИГ) осуществляются довольно быстро, а значит, данный метод позволяет ускорить работу с квадрадеревом.

Помимо анализа выделения видимых объектов сцены, квадрадерево также может использоваться для проверки столкновения объекта, ограниченного пространством поиска, с другими объектами.

Заключение

Таким образом, в разработанных модулях были реализованы в виде классов компоненты, необходимые для работы графического конвейера, такие как ядро, менеджер ресурсов и менеджер видеоустройства. Для хранения ресурсов одного типа предназначены пулы данных. Создан базовый класс для описания всех ресурсов, содержащий единый интерфейс, а также шаблонный класс для контроля над ресурсами одного типа. Сам менеджер ресурсов управляет всеми ресурсами через объекты этого шаблонного класса. С использованием средств, предоставляемых DirectX, на основе базового класса ресурсов были созданы классы видеоресурсов, таких как текстуры,

материалы поверхностей, файлы эффектов, вершинные и индексные буферы, а также методы рендеринга. Для оптимизации процесса рендеринга в состав менеджера видеоустройства был введен специальный объект — очередь на рендеринг.

Для построения графа сцены спроектирован класс узла сцены, а для представления всех визуальных объектов игры — класс объекта сцены, на основе которого строятся классы реальных объектов, таких как деревья и участки ландшафта. Задача разбиения игрового пространства и выделения видимых объектов для ускорения вывода сцены была решена при помощи расширенного варианта квадрадерева, для чего были созданы классы квадрадерева и его узла, способные размещать объекты в узлах квадрадерева и производить поиск объектов, попадающих в заданное пространство.

Для управления всеми вышеперечисленными компонентами на основе класса приложения DirectX было создано ядро приложения. Этот объект отвечает за создание и уничтожение всех менеджеров, получение и реакцию на системные сообщения и пользовательский ввод, а также регулирует процессы рендеринга и обновления сцены.

С использованием этих инструментов были также реализованы классы для работы с простыми моделями и специальная система классов для работы с ландшафтом. Построение ландшафта осуществляется по карте высот, при этом для удобства отображения он разбивается на участки, в которых хранится часть данных о геометрии ландшафта. При помощи метода блочного ландшафта проводятся дополнительные построения, позволяющие оптимизировать вывод участков ландшафта.

В качестве примера использования всех этих возможностей была написана демонстрационная программа.

Список литературы

1. *Грег Снук*. Создание 3D-ландшафтов в реальном времени с использованием C++ и DirectX9. М.: КУДИЦ-ОБРАЗ, 2006. 368 с.
2. *Плюммер Дж*. Гибкая и масштабируемая архитектура для компьютерных игр, часть первая. 2004 [Электронный ресурс]. — Режим доступа: http://www.dtf.ru/articles/read.php?id=40757&page=l.
3. *Поздняков К*. Статьи по разработке графического движка [Электронный ресурс]. — Режим доступа: http://www.gamedev.ru/articles/?id=70035, http://www.gamedev.ru/articles/?id=70029, http://www.gamedev.ru/articles/?id=30036.
4. *Ворсин С. В*. Визуализация лесных массивов и рельефа местности в реальном времени [Электронный ресурс]. — Режим доступа: http://www.gamedev.ru/articles.
5. *Фролов А*. Эффективный менеджер памяти для однотипных элементов [Электронный ресурс]. — Режим доступа: http://www.gamedev.ru/articles/?id=70117.
6. Применение иерархии объектов в играх [Электронный ресурс]. — Режим доступа: http://www.gamedev.ru/articles/?id=70101.
7. Статья, посвященная проектирования игрового приложения [Электронный ресурс]. — Режим доступа: http://www.gamedev.ru/proj/?id= I &doc=5.