

УДК 371.385:681.3

Н.В. БЕЛОУС, А. П. ВЫРОДОВ, И. Ю. ШУБИН

О НЕКОТОРЫХ АЛГОРИТМАХ ПОСТРОЕНИЯ ВИРТУАЛЬНЫХ ПРОСТРАНСТВ

Одной из актуальных задач при построении математических моделей формального описания интеллекта в системах с элементами искусственного интеллекта является проблема компьютерного моделирования и, в частности, моделирования способов построения виртуальных пространств в темпе реального времени, а также методов создания систем проектирования таких пространств.

Интерактивные средства разработки трехмерного пространства.

При разработке трехмерных презентаций, или “трехмерных” Web-страниц возможно применение коммерческих программных продуктов, таких как пакет Virtual Home Space Builder (VHSB) от фирмы Paragraph International. Этот пакет выгодно отличается от аналогичных интуитивно понятным интерфейсом и простотой в обращении. Развитая система контекстно-зависимой помощи полностью охватывает все возможности данного пакета. Возможность применения текстур, выполненных в режиме высокого разрешения с использованием палитры, содержащей более миллиона различных цветов, позволяет придать генерируемому пространству профессиональный реалистичный вид [1+2].

Однако у такого подхода, кроме очевидных преимуществ, есть свои недостатки. Во-первых, предъявляются довольно существенные требования к ресурсам компьютера: для ознакомительной версии пакета необходим, по крайней мере, компьютер с процессором i486DX4-100 и 16 Мегабайт оперативной памяти. Кроме того, отсутствие графического акселератора не позволяет развить необходимую для нормальной работы частоту смены кадров (30 кадров в секунду) при приемлемом размере окна просмотра. Во-вторых, к настоящему времени лишь анонсируются программные средства, позволяющие создавать объекты, перемещающиеся в пространстве независимо от пользователя (наблюдателя).

Метод заранее подготовленной анимации.

Существует также следующий, довольно распространенный метод построения трехмерного пространства: заранее в пакете трехмерной графики создаются анимационные фильмы для каждого перемещения пользователя в определенном направлении. После чего разрабатывается

программа, основной функцией которой является “прокрутка” соответствующей анимации при выборе пользователем одного из возможных направлений перемещения. Одним из главных достоинств данного метода является то, что сложность создаваемого пространства практически не ограничивается. Самый существенный недостаток — ограничение свободы перемещения пользователя.

Разработка системы проектирования виртуальных пространств.

Под термином “система проектирования виртуальных пространств” подразумевается комплекс программных средств, позволяющий пользователю создавать трехмерное пространство и затем путешествовать по нему. Основной сложностью создания данных систем является достижение необходимой частоты смены кадров. Для создания реалистичной иллюзии движения необходимо обеспечить частоту, равную 30 кадрам в секунду. Описываемая система математических моделей и инструментальных средств генерации трехмерного пространства имеет следующие характеристики:

1. Возможность исполнения на компьютере с i286 микропроцессором.
2. Использование всего лишь 450К оперативной памяти.
3. Размер исполняемого файла — 57 Кбайт;
4. Время трансляции компилятором Borland C++ 4.2 на компьютере с процессором i486DX4-100 составляет 39 с.
5. Возможность работы с анимированными текстурами и перемещающимися независимо от пользователя объектами.

Обобщенная схема алгоритма.

Основой алгоритма генерации трехмерного пространства в реальном времени является метод трассировки луча. (Ray Casting) — термин, которым описывается процесс превращения двухмерного представления пространства в трехмерное. При первом приближении этот процесс выглядит следующим образом. Пусть угол обзора пользователя составляет 60° и пусть имеется двухмерная карта нашего пространства с нанесенной на ней стенами и позицией пользователя. О самом пользователе необходимо знать три характеристики: X и Y координаты, а также направление взгляда. Последнее зададим величиной угла, заданной в системе отсчета углов, принятой в языке программирования C++ (рис. 1).

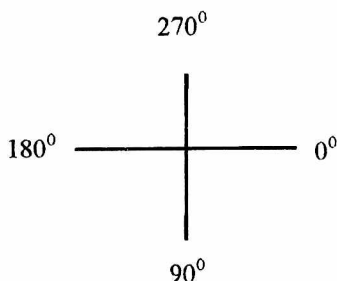


Рис. 1

Пусть пользователь расположен так, как показано на рис. 2.

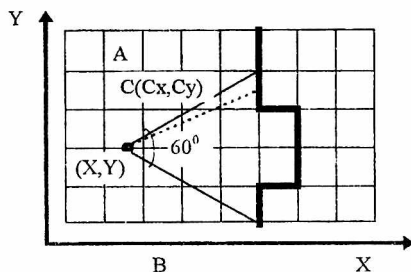


Рис. 2

Сегмент стен, заключенный между точками *A* и *B*, должен быть прорисован на экране, так как он находится внутри угла обзора. Чтобы создать иллюзию трехмерности, необходимо вычислять высоту стен как функцию расстояния до них. Определение расстояния до стены и является функциональным назначением метода трассировки луча. Алгоритм этого метода следующий:

Из текущих координат пользователя проводится луч до пересечения со стеной или выхода за пределы карты (см. рис. 2). В точке *C* луч пересекается со стеной. Обозначим через *Cx* и *Cy* координаты пересечения. Тогда по формуле

$$\text{расстояние} = \sqrt{(Cx - X)^2 + (Cy - Y)^2}$$

будет определено искомое расстояние до стены для точки C . На этом метод трассировки луча завершается.

В зависимости от этого расстояния происходит регулировка высоты одной колонки стены, которую на карте (как на виде сверху) обозначили точкой C . Данная манипуляция производится с целью достижения эффекта перспективы.

После этого включается процедура поиска текстуры, которую необходимо наложить на данную стену, и в ней выделяется колонка, которая будет отображаться в точке C . Следующим этапом является установление соответствия между высотой колонки текстуры и высотой колонки стены. Достигается это с помощью метода масштабирования растровой графики. После этого отмасштабированная колонка текстуры выводится на дисплей. Таким образом, была прорисована всего лишь одна колонка дисплея. Чтобы прорисовать остальные, необходимо протрассировать для каждой из них по лучу, как это описано выше. Поэтому при использовании дисплея с графическим режимом 320×200 , необходимо протрассировать 320 лучей.

Оптимизированный алгоритм трассировки луча.

Описанный выше алгоритм трассировки луча имеет ряд недостатков. Самый существенный из них — слишком низкая скорость выполнения. 70 % всего времени расходуется на вычисление квадратного корня по приведенной формуле и на проверку пересечения в данной точке луча со стеной. Очевидно, чтобы ускорить процесс, необходимо, во-первых, исключить операцию извлечения квадратного корня, а во-вторых, сократить число точек, в которых луч может пересекаться со “стеной”. С этой целью создадим новую карту нашего пространства. Разобьем ее на одинаковые квадраты размером, например, 70×70 единиц. Это позволит пользователю переместиться на 70 единиц в направлении оси X или оси Y перед тем, как попасть в другой квадрат. Стены будем строить только на сторонах квадратов (см. рис.2). Таким образом, имеются стены двух типов:

- 1) проектирующиеся в точку на оси X (для краткости будем в дальнейшем называть их X -стенами);
- 2) проектирующиеся в точку на оси Y (аналогично назовем их Y -стенами).

Вернемся к рис.2 и снова будем вести луч до пересечения его со стеной. Но как же определить тип стены, пересеченной лучом? Самый простой способ — трассировать по одному направлению не один луч, а два. При трассировке одного из них будем обращать внимание только на горизонтальные стороны квадратов, где могут находиться Y -стены, а при трассировке другого — только на вертикальные, где могут находиться X -

стены. Первое, что необходимо определить — координаты пересечения нашего луча с горизонтальной (вертикальной) стороной первого встретившегося на пути квадрата. А так как размеры квадратов фиксированы, то координаты следующего пересечения можно экстраполировать. Ниже приведена общая схема оптимизированного таким образом алгоритма трассировки луча:

1. Вычесть из угла направления взгляда пользователя 30^0 , чтобы получить текущий угол просмотра (экран заполняется слева направо).

2. Проводим на карте первый луч для текущего угла просмотра и обращаем внимание только на вертикальные стороны квадратов (т.е. ищем X -стену). Определяем X , Y координаты пересечения луча с нужной стороной.

3. Используя тангенс текущего угла просмотра, определяем положение точки пересечения относительно начала вертикальной стороны пересеченного квадрата.

4. Зная ширину каждого квадрата (в нашем случае — 70 единиц), определяем следующую Y координату точки, с которой пересечется наш луч, если мы увеличим X координату на 70. Т.е. мы сразу пропускаем 70 точек карты, так как из ее построения следует, что там стена находится не может.

5. Шаги 2—4 выполняются до тех пор, пока луч не пересечется со стеной или не выйдет за пределы карты.

6. Как только определена X -стена, используется Y координата, чтобы определить, какую колонку стены надо прорисовать, и X координата, чтобы определить расстояние до стены. Таким образом, исключена необходимость вычисления квадратного корня.

7. Повторяя шаги 2—5, обращаем внимание лишь на горизонтальные стороны квадратов (т.е. ведется поиск Y -стены). Теперь, если будет “обнаружена” стена, то будет использована X координата, чтобы определить, какую колонку стены надо прорисовать, и Y координата для определения расстояния.

8. Сравниваются два полученных на предыдущих шагах расстояния с целью определения ближайшей к наблюдателю стены и сохраняется номер колонки этой стены.

9. Так как алгоритм использует графический режим 320×200 и угол обзора в 60^0 , то необходимо увеличить угол просмотра на $60:320=0.1875^0$. После этого процесс повторяется снова и снова, пока не будут протрассированы все 320 пар лучей. Следует отметить, что данный способ оптимизации метода трассировки луча имеет недостаток — стены могут соприкасаться только под прямым углом. Этим объясняется “угловатость” создаваемого

описанными алгоритмическими средствами виртуального пространства. Следующим этапом оптимизации данных математических моделей является поиск нового способа оптимизации алгоритма трассировки луча, лишенного указанного выше недостатка.

Наложение текстуры.

Предположим, что произведена трассировка одной пары лучей и найдена ближайшая стена. Результатом является номер колонки определенной стены и расстояние до нее. Что касается последнего, то необходимо, разделив его на экспериментально подобранную константу, получить высоту первой. При подборе данной константы следует сделать высоту стены равной высоте пользователя (наблюдателя). По мнению авторов, это должно способствовать повышению реализма создаваемого пространства. В качестве такой константы используется 20 000, однако возможно применение других субъективно выбранных констант.

Теперь определяем текстуру, которую надо наложить на стену и, используя номер колонки стены, найденный на этапе трассировки луча, находим нужную колонку текстуры. Если вычисленная алгоритмически высота колонки стены и высота колонки текстуры не совпадают, то к последней должен быть применен алгоритм масштабирования.

Масштабирование растровой графики.

Сам алгоритм масштабирования растровой графики можно найти в [3], где он излагается в популярной форме, а также приводится текст программы на языке программирования Microsoft C, реализующей указанный алгоритм. Конечно, это не самый совершенный вариант данного алгоритма, но зато самый простой и, что самое главное, достаточно быстрый. Фрагмент кода из [3] позволяет масштабировать изображение как вдоль горизонтальной, так и вдоль вертикальной осей. Для целей данной разработки необходимо масштабирование только вдоль вертикальной оси. Следует отметить также необходимость перевода этого фрагмента кода с языка программирования высокого уровня C на Ассемблер, что позволит на 80 % ускорить его исполнение. Ниже описан специализированный вариант данного алгоритма.

Предположим, что в результате масштабирования должна быть получена колонка текстуры, высотой 90 пикселей, из имеющейся колонки, высотой 60 пикселей. В этом случае алгоритм должен “вытянуть” колонку, т.е. изменить масштаб вдоль оси Y. Коэффициент масштабирования вычисляется как пропорция:

$$90:60=1.5.$$

Сам алгоритм можно представить следующим образом:

1. Пиксели исходной колонки проиндексируем целыми числами от 1 до 60.
2. Проиндексируем все 90 ячеек результирующей колонки от 1 до 60, но дробными числами. При этом шаг приращения дробного индекса вычисляется по формуле:
шаг приращения = 1: коэффициент масштабирования.
3. Для данной ячейки результирующей колонки текстуры, округлив ее дробный индекс до ближайшего целого, получим номер пикселя исходной колонки.
4. Выполним копирование пикселя с полученным номером в данную ячейку результирующей колонки текстуры.
5. Повторим шаги 3, 4 для каждой из 90 ячеек.

Реализация статических и перемещающихся объектов.

Для манипуляций с объектами приходится создать отдельную карту, совпадающую по размерам с картой конструируемого пространства. Описанный выше алгоритм трассировки луча модифицируется с целью наблюдения за объектами. Теперь в этом алгоритме происходит отслеживание трассируемого луча по карте объектов. Если луч пересечется с объектом, то этот объект заносится в специальный список и у него устанавливается флаг видимости с целью предотвращения добавления того же самого объекта еще раз в список при трассировке “соседнего” луча. Причем необходима сортировка списка по дальности объектов так, чтобы дальние объекты были первыми. Следует отметить, что трассировка луча должна продолжаться и за объектами, так как информация о стенах, которые могут находиться позади объектов и только частично ими перекрываться, также необходима при прорисовке пространства. Объекты обрабатываются лишь после того, как стены будут прорисованы в специальном буфере экрана. Обработка списка объектов начинается по порядку записи, т.е. с самых дальних. При этом расстояние каждой колонки спрайта объекта следует сравнивать с расстоянием до соответствующей стены: если стена ближе, то надо рисовать колонку стены, а не объекта. Отметим, что нет необходимости проверять еще и расстояния до других объектов, так как ближние перекроют дальние. Этот процесс продолжается до тех пор, пока все объекты не будут прорисованы. Очевидно, что при каждой перерисовке экрана необходимо очищать флаги видимости объектов. Это позволит находить новые объекты и перемещать старые.

В настоящее время авторами проводятся исследования по созданию новой структуры данных, которая позволила бы разделить карту на

элементы произвольной формы, а не только квадратной, как в данный момент. Это предоставило бы возможность не только располагать стены под произвольным углом друг к другу, но и перейти от "одноэтажных" конструкций к "многоэтажным". Кроме того, оценивается целесообразность перехода из реального режима работы в защищенный, что может положительно сказаться на производительности всей системы в целом. Все эти методы преследуют единственную цель — добиться максимальной близости к реальному миру и при этом минимизировать требования к аппаратным ресурсам вычислительной системы.

Список литературы: 1. *Алексеев А.Н.* Построение виртуальных миров // Компьютер Пресс. 1997 № 2. С. 25—30. 2. *Алексеев А.Н.* Пакет VHSB // Компьютер Пресс. 1997. №3. С. 85—88. 3. *А. ЛаМот.* Секреты программирования игр: К.: BHV, 1995. 355 с.

Поступила в редколлегию 23.10.97