УДК 371.385:681.3

АЛГОРИТМЫ ПОСТРОЕНИЯ ВИРТУАЛЬНЫХ ПРОСТРАНСТВ

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

Исследуется проблема компьютерного моделирования трехмерных пространств в режиме реального времени. Описываются полученные в результате исследования алгоритмы, позволяющие создавать иллюзию трехмерного пространства, и их программные реализации.

Системы построения виртуальных пространств используются при создании компьютерных презентаций, компьютерных обучающих систем и Web-страниц, использующих для представления динамических данных трехмерные объекты. Архитекторы и промышленные дизайнеры применяют системы построения виртуальных пространств при создании статических моделей в целях демонстрации их своим заказчикам. Кроме того, интерфейс операционных систем 2000 года будет также трехмерным. Все это вызывает повышенный интерес как к трехмерной компьютерной графике в целом, так и к способам построения виртуальных пространств в частности.

1. Основные этапы построения трехмерного пространства

Рассмотрим идеализированную модель трехмерного пространства – гипотетический трехмерный коридор, стены которого расположены под прямым углом. Чтобы однозначно задать такую модель, нам потребуется знать:

1) высоту всех стен коридора;

2) соразмещение стен коридора, т.е. одну из проекций коридора на координатные плоскости.

Для упрощения модели сделаем высоту всех стен одинаковой, а сами стены – расположенными на одном уровне. При указанных ограничениях для описания структуры коридора целесообразно выбрать его проекцию на горизонтальную плоскость, т.е. вид сверху, так как именно этот вид содержит всю необходимую информацию для однозначной визуализации идеализированной модели. Построение трехмерного пространства в режиме реального времени подразумевает непосредственное взаимодействие с пользователем: проекция модели пространства на экран дисплея должна изменяться в зависимости от текущего положения наблюдателя, его угла обзора и направления взгляда. В случае идеализированной модели эта зависимость тривиальна: высота проекции стены изменяется обратно пропорционально ее расстоянию до наблюдателя.

Таким образом, первым этапом построения трехмерного пространства является определение расстояния между наблюдателем и объектами пространства, находящимися внутри его угла обзора. Для вычисления указанного расстояния и определения видимых наблюдателем объектов применяется так называемый метод трассировки луча.

На следующем этапе осуществляем поиск текстур (плоских графических изображений), которые необходимо наложить на объекты, найденные на предыдущем этапе. Искомое множество текстур однозначно определяется типом объекта и его состоянием.

Вследствие того, что размеры текстур фиксированы, а размеры объектов могут изменяться, возникает необходимость динамического масштабирования текстур. Итак, следующим этапом построения трехмерного пространства является масштабирование текстур, найденных на предыдущем этапе. Подчеркнем, что данный этап необходим только в случае несовпадения размеров текстур с размерами соответствующих объектов. Более подробное описание алгоритмов наложения текстур и их масштабирования можно найти в [1].

На заключительном этапе выполняется прорисовка объектов с наложенными на них отмасштабированными текстурами на экране дисплея, причем эта прорисовка выполняется в порядке, обратном нахождению этих объектов на первом этапе.

Следует отметить исключительную важность первого этапа построения трехмерного пространства – этапа трассировки луча. Реалистичность получаемого пространства в конечном счете определяется точностью вычисления расстояний от наблюдателя до видимых объектов пространства. Ошибки, допущенные на данном этапе, приводят не только к несоответствию проекции пространства его модели, но также к невозможности свободного перемещения наблюдателя. Поэтому в данной статье внимание сконцентрировано на разработанных алгоритмах трассировки луча, выполняющих построение трехмерного представления указанной выше идеализированной модели.

2. Первый вариант алгоритма трассировки луча

Зададим угол обзора наблюдателя равным 60 градусам. На рис. 1 показан вид сверху идеализированной модели трехмерного пространства с нанесенной на нем позицией наблюдателя, которая задается координатами X и Y, а также вектором направления взгляда P. Сегмент стен, заключенный между точками A и B, должен быть прорисован на экране, так как он находится внутри нашего угла обзора.

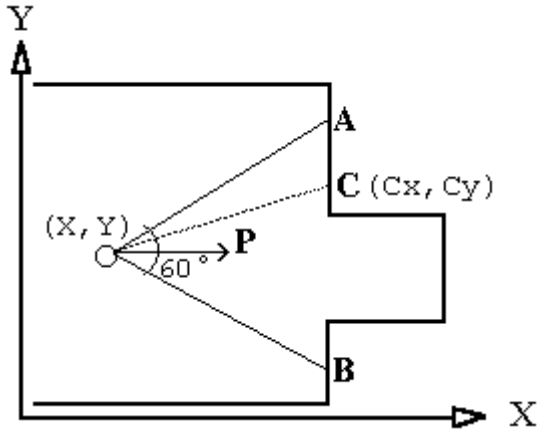


Рис.1. Идеализированная модель (вид сверху)

Итак, как было сказано выше, чтобы создать иллюзию трехмерности, необходимо вычислять высоту стен, как функцию расстояния до них от наблюдателя. Для определения расстояния до сегмента стен необходимо разбить его на фиксированное количество точек, расстояние до которых должен определять алгоритм трассировки луча. Для каждой такой точки необходимо протрассировать по одному

лучу. Количество лучей целесообразно положить равным количеству столбцов дисплея в текущем видеорежиме. С помощью этого достигается полноэкранная визуализация модели трехмерного пространства. Каждый луч выходит из текущих координат наблюдателя под определенным углом, отсчитываемым относительно вектора направления взгляда. Учитывая, что сумма всех углов трассируемых лучей должна быть равной углу обзора наблюдателя, шаг их изменения можно определить по формуле

$$\Delta A = \frac{60}{VM}, \qquad (1)$$

где ΔA — шаг изменения угла наклона трассируемых лучей; VM — разрешение дисплея по горизонтали в текущем видеорежиме.

Чтобы определить расстояние до зафиксированных точек сегмента стены, мы из текущих координат наблюдателя под определенным углом "ведем" луч до пересечения с этими точками (рис. 1). Рассмотрим более подробно процесс "ведения" луча.

Так как для каждого трассируемого луча нам известен угол его наклона, то мы можем вычислить угловой коэффициент луча по формуле

$$K = \operatorname{tg} a, \qquad (2)$$

где a — угол наклона трассируемого луча; K — угловой коэффициент луча.

Тогда уравнение, описывающее трассируемый луч, можно записать в следующем виде:

$$Y_i = Y + K*(X_i - X). \qquad (3)$$

Здесь X_i и Y_i — абсцисса и ордината возможной точки пересечения трассируемого луча со стеной; X, Y — текущие координаты наблюдателя. Координата X_i изменяется по следующему закону:

$$\begin{aligned} X_{i+1} &= X_i \pm 1, \\ X_1 &= X \pm 1. \end{aligned} \qquad (4)$$

Процесс трассировки луча в данном варианте алгоритма осуществляется следующим образом. Придавая абсциссе X_i приращение +1, если вектор направления взгляда P наблюдателя совпадает с направлением оси абсцисс, и -1 в противном случае, вычисляем по формуле (3) ординату Y_i. Если точка с координатами (X_i, Y_i) не принадлежит стене, то аналогично вычисляем координаты следующей точки и т.д. В программной реализации данного алгоритма для проверки наличия стены в указанной точке мы используем ее координаты в качестве индекса элемента двухмерного массива, описывающего структуру трехмерного пространства. Ненулевое значение элемента массива свидетельствует о наличии в соответствующей точке стены.

Допустим, что в точке C наш луч пересекается со стеной (рис.1). Обозначим через Cx и Cy координаты пересечения. Тогда по формуле

$$D = \sqrt{(Cx - X)^2 + (Cy - Y)^2} \qquad (5)$$

определим искомое расстояние D от наблюдателя до точки C стены.

Приведем обобщенную схему данного варианта алгоритма трассировки луча.

1. Вычитаем из угла направления взгляда наблюдателя 30 градусов, чтобы получить текущий угол просмотра.

2. Даем абсциссе X_i приращение в соответствии с соотношением (4).

3. Для текущего значения X_i вычисляем по формуле (3) соответствующее значение ординаты Y_i (т.е. проводим луч для текущего угла просмотра).

4. Проверяем наличие стены в точке с координатами (X_i, Y_i).

5. Продолжаем шаги 2—4 до тех пор, пока трассируемый луч не пересечется со стеной или не произойдет выход за пределы трехмерного пространства.

6. При обнаружении в точке с координатами (X_i, Y_i) стены определяем расстояние до нее, используя формулу (5).

7. Увеличиваем угол просмотра на заранее вычисленное по формуле (1) значение, после чего весь алгоритм повторяется снова и снова, пока не будут протрассированы все VM пар лучей.

Разработанный выше алгоритм трассировки луча в полной мере осуществляет процесс преобразования двухмерного представления виртуального пространства в трехмерное. Кроме того, достоинством данного алгоритма является простота его программной реализации и высокая точность определения расстояния. Последнее свойство объясняется использованием для вычисления расстояния по формуле (5) математического сопроцессора. Однако описанный алгоритм трассировки луча имеет и ряд недостатков. Самым существенным из них является слишком низкая скорость выполнения. 70 % всего времени расходуется на вычисление квадратного корня по формуле (1) и на проверку пересечения в данной точке луча со стеной. Все это обуславливает необходимость модификации существующего алгоритма с целью повысить скорость выполнения.

3. Второй вариант алгоритма трассировки луча

Как только выявили основные недостатки предыдущего варианта алгоритма, провели поиск методов их устранения. В результате поиска определены следующие подходы к оптимизации первого варианта алгоритма:

– во-первых, необходимо устранить операцию извлечения квадратного корня, так как даже при использовании математического сопроцессора данная операция влечет за собой существенное замедление работы алгоритма;

– во-вторых, сократить число точек, в которых луч может пересекаться со стеной.

С этой целью создадим новый вид сверху нашего виртуального пространства. Разобьем его на одинаковые квадраты размером, например, 64x64 единиц. Это позволит наблюдателю переместиться на 64 единицы в направлении оси X или Y перед тем, как попасть в другой квадрат. Стены будем строить только на сторонах квадратов. Таким образом, мы имеем стены двух типов:

1) проектирующиеся в точку на оси X (для краткости мы будем в дальнейшем называть их X-стенами);

2) проектирующиеся в точку на оси Y (аналогично назовем их Y-стенами).

Вернемся опять к рис.1 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 нам надо найти, — координаты пересечения нашего луча с горизонтальной (вертикальной) стороной первого встретившегося на пути квадрата. А так как размеры квадратов фиксированы, то координаты следующего пересечения можно легко подсчитать.

Подробнее остановимся на вычислении координат точки пересечения трассируемого луча со стороной первого встретившегося на пути квадрата. Обозначим через X_1 — абсциссу точки пересечения луча с горизонтальной стороной квадрата, через Y_1 — ординату точки пересечения луча с вертикальной стороной квадрата, а через X и Y — текущие координаты наблюдателя. Тогда уравнение, описывающее трассируемый луч (уравнение прямой с угловым коэффициентом), запишется в виде

$$\frac{Y_1 - Y}{X_1 - X} = K, \qquad (6)$$

откуда находим:

$$Y_1 = K * (X_1 - X) + Y; \qquad (7)$$

$$X_1 = K^{-1} * (Y_1 - Y) + X. \qquad (8)$$

Коэффициент K в этих формулах имеет то же значение, что и в формуле (2). Отметим, что каждое из полученных преобразований требует предварительного вычисления другого. Поэтому, чтобы избежать такой зависимости, необходимо в формулу (7) вместо абсциссы X_1 подставить абсциссу любой другой точки, лежащей на трассируемом луче (рис. 2).

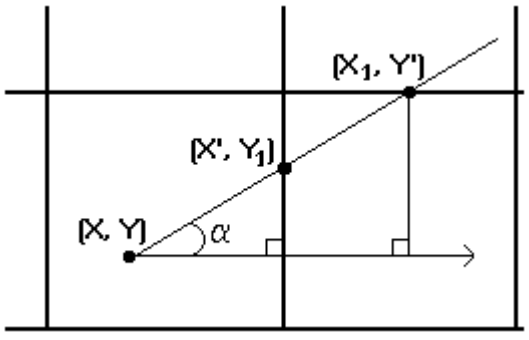


Рис.2. Геометрическая интерпретация алгоритма трассировки луча

В качестве таковой удобно взять абсциссу первой граничной вертикальной линии X'. Аналогично вместо Y_1 в формулу (8) подставим ординату Y' первой граничной горизонтальной линии (рис. 2). Теперь формулы (7) и (8) можно переписать в следующем виде:

$$Y_1 = K * (X' - X) + Y; \qquad (7^*)$$

$$X_1 = K^{-1} * (Y' - Y) + X. \qquad (8^*)$$

Формулы (7*) и (8*) полностью подходят для проведения вычислений, так как координаты X' и Y' могут быть легко определены из следующих соображений.

Пусть координаты наблюдателя X и Y могут принимать значения только из фиксированного диапазона (определяемого размерами виртуального пространства) и необходимо найти абсциссу X' первой граничной вертикальной линии (рис. 2). Для этого сначала разделим нацело, а затем умножим координату X наблюдателя на величину стороны квадрата, которую положим равной 64 единицам, как и в первом варианте алгоритма. Полученная координата есть абсцисса левой граничной вертикальной линии, а так как на рис. 2 вектор направления взгляда указывает вправо, то необходимо увеличить полученную координату на величину стороны квадрата. Если бы вектор направления взгляда указывал влево, то необходимо было бы уменьшить эту координату на величину стороны квадрата. Аналогично находим ординату Y' первой граничной горизонтальной линии.

После вычисления координат точки первого пересечения переходим к вычислению координат следующего возможного пересечения. Так как ширина каждого квадрата фиксирована (в данном случае — 64 единицы), определяем по формуле (10) следующую координату Y_{i+1} точки, с которой пересечется трассируемый луч, если увеличить координату X' на 64. Таким образом, сразу пропускаем 64 точки карты, так как из ее построения следует, что там стена находиться не может. Аналогично, увеличивая координату Y' на 64, по рекуррентной формуле (9) определяем координату X_{i+1} следующего возможного пересечения. Итак, координаты точек возможного пересечения, начиная со второй, определяются по следующим рекуррентным формулам:

$$X_{i+1} = X_i + K^{-1} * c; \qquad (9)$$

$$Y_{i+1} = Y_i + K * c, \qquad (10)$$

где c — сторона квадрата.

Как только найдена X-стена, то чтобы определить расстояние до нее, следует использовать координату X_i, а если найдена Y-стена — следует использовать координату Y_i. Действительно, расстояние от точки с координатами (X,Y) до точки с координатами (X',Y_1) или от точки (X,Y) до точки (X_1,Y') (рис. 1) можно определить двумя способами:

1) используя формулу (5);

2) пользуясь соотношениями между сторонами и тригонометрическими функциями углов прямоугольного треугольника.

Как было сказано выше, вычисление искомого расстояния с использованием операции извлечения корня не является эффективным, даже при табулировании функции извлечения корня. Поэтому целесообразно использовать тригонометрические функции от угла наклона a трассируемого луча.

Таким образом, вычисление расстояния до точки пересечения трассируемого луча с горизонтальной стороной квадрата следует производить по следующей формуле:

$$Dx = (X_i - X) * \cos^{-1} a, \qquad (11)$$

где Dx — расстояние до точки пересечения трассируемого луча с горизонтальной стороной квадрата.

Аналогично, вычисление расстояния до точки пересечения трассируемого луча с вертикальной стороной квадрата следует производить по формуле:

$$Dy = (Y_i - Y) * \sin^{-1} a, \qquad (12)$$

где Dy — расстояние до точки пересечения трассируемого луча с вертикальной стороной квадрата.

Угол a в формулах (11) и (12) является углом между трассируемым лучом и вектором направления взгляда.

Приведем общую схему второго варианта алгоритма трассировки луча:

1. Вычитаем из угла направления взгляда наблюдателя 30 градусов, чтобы получить текущий угол просмотра.

2. Определяем абсциссу X' первой граничной вертикальной линии: координату X наблюдателя сначала делим нацело, а затем умножаем на величину стороны квадрата. Если угол направления взгляда находится в пределах от 0 до 180 градусов (т.е. вектор направления взгляда указывает вправо), то увеличиваем полученную координату на величину стороны квадрата.

3. Для текущего угла просмотра по формуле (7*) определяем координату Y_1 пересечения трассируемого луча с вертикальной стороной первого квадрата.

4. Проверяем наличие стены в точке с координатами (X', Y_1), т.е. ищем X-стену. Если стена найдена или достигнут предел виртуального пространства, то сразу переходим к п.7, минуя п.5, 6.

5. Увеличиваем координату X' на величину стороны квадрата (в нашем случае — 64 единицы), если направление вектора взгляда совпадает с направлением оси абсцисс (вектор указывает вправо), иначе уменьшаем координату X' на указанную величину. Определяем по формуле (10) следующую координату Y_i (i=2,3,4...) точки, в которой пересечется наш луч со стороной следующего квадрата.

6. Проверяем наличие стены в точке с координатами (X', Y_i). Если стена не найдена и не достигнут предел виртуального пространства, то возвращаемся к п.5, иначе переходим к п.7.

7. Используя формулу (13), определяем расстояние до соответствующей точки X-стены. Таким образом, мы устранили необходимость вычисления квадратного корня.

8. Осуществляем трассировку второго луча для текущего угла просмотра (т.е. ищем Y-стену). Действия, выполняемые на этом этапе, аналогичны действиям, выполняемым в п. 2—7, с той лишь разницей, что вместо формул (7*), (10) и (13) используем формулы (8*), (11) и (12), а вместо координат X, X', Y_1, Y_i — координаты Y, Y', X_1, X_i соответственно.

9. Если для текущего угла просмотра обнаружены Y-стена и X-стена одновременно, то сравнив расстояния до них, определяем ближайшую к нам стену, которую и следует прорисовать на экране дисплея.

10. Увеличиваем угол просмотра на заранее вычисленное по формуле (1) значение. После этого весь алгоритм повторяется снова и снова, пока не будут протрассированы все VM пар лучей.

Ввиду того, что данный вариант алгоритма трассировки луча является оптимизированным по быстродействию первым вариантом алгоритма, он также подвержен влиянию проекционных искажений.

4. Проекционные искажения

Оба алгоритма трассировки луча используют одновременно полярные и декартовы системы координат. Как следствие, это привело к появлению так называемых "сферических" искажений.

Действительно, так как все трассируемые лучи выходят из одной точки, то эквидистантные поверхности будут представлять собой концентрические сферы. Поэтому, чтобы построить стену, расположенную перпендикулярно к вектору направления взгляда, надо на виде сверху изобразить стену не как прямую линию, а как дугу окружности. Только в этом случае построенная стена будет выглядеть так, как показано на рис. 3, а.

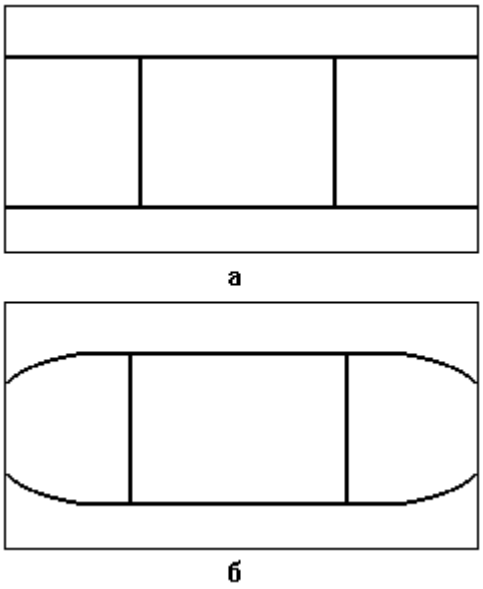


Рис.3. Изображение на экране дисплея: а — с компенсацией сферических искажений; б — без компенсации

Так как при подготовке карты стены условно изображались в виде отрезков прямых линий, то расстояние до середины стены, расположенной перпендикулярно к вектору направления взгляда, будет всегда меньше расстояний до ее краев. Следовательно, высота колонки текстуры данной стены будет максимальной в ее середине и станет постепенно уменьшаться при приближении к ее краям. Поэтому изображение на экране дисплея будет иметь вид, показанный на рис. 3,б, вместо вида, показанного на рис. 3,а.

На рис. 3,а показан вид стены с компенсацией "сферических" искажений, а на рис. 3,б — вид стены без компенсации "сферических" искажений. Данные "сферические" искажения являются синусоидальными и поэтому для их компенсации необходимо умножить высоту колонки стены на косинус текущего угла просмотра, т.е. вычисление высоты колонки стены надо выполнять по формуле

$$Ch = (\cos a) * 20\,000 / D, \qquad (13)$$

где Ch — высота колонки стены; D — расстояние до стены.

5. Особенности программной реализации алгоритмов трассировки луча

Основной сложностью создания программной реализации данных алгоритмов является обеспечение необходимой частоты смены кадров изображения на экране дисплея (не менее 30 кадров в секунду). Учитывая, что указанные алгоритмы являются циклическими, любая оптимизация их программной реализации, будучи многократно повторенной, даст

значительный эффект. Наиболее важными, с точки зрения повешения скорости выполнения, являются следующие методы оптимизации, учитывающие специфику данных алгоритмов.

Особое внимание следует уделить выбору графического видеорежима работы дисплея, так как число повторений обоих алгоритмов однозначно определяется количеством пикселей в строке экрана, т.е. разрешением дисплея в заданном видеорежиме. Кроме того, чем выше разрешающая способность дисплея (и, соответственно, качество получаемого изображения), тем больше данных должна подготовить программа и тем жестче требования к скорости ее выполнения. Следовательно, необходимо найти компромисс между качеством получаемого изображения и скоростью выполнения программы.

В обоих алгоритмах интенсивно используются тригонометрические функции углов наклона трассируемых лучей. Так как угол наклона изменяется дискретно (формула (1)), становится возможным составление таблиц заранее вычисленных для всех возможных углов наклона значений следующих тригонометрических функций: $\sin^{-1} a$, $\cos^{-1} a$, tg a, ctg a. После этого для получения значения одной из этих функций достаточно обратиться к соответствующему элементу таблицы, что оказывается гораздо быстрее непосредственного вычисления значения функции арифметическим сопроцессором. Определим размер такой таблицы для графического режима 320x200. В соответствии с формулой (1) шаг изменения угла наклона составляет 60/320=0,1875°. Тогда таблица будет содержать 360°/0,1875°=1920 элементов. Аналогичные вычисления для графического режима 640x480 дают размер таблицы в 3840 элементов. Таким образом, в программной реализации алгоритмов угол a это индекс, изменяющийся в диапазоне от 0 до 1920 (от 0 до 3840), для таблиц тригонометрических функций. Как известно, функция тангенс стремится к бесконечности при значениях аргумента 90 и 270 градусов. Поэтому необходимо ввести дополнительные проверки угла наклона, чтобы избежать в этих точках ошибки деления на нуль.

Благодаря тому, что сторона квадрата имеет фиксированный размер 64 единицы, становится возможным составление таблиц заранее вычисленных значений от функций 64 * tg a, 64 * ctg a, которые используются во втором алгоритме для вычисления координат точек возможного пересечения трассируемого луча со стеной.

В обоих алгоритмах большинство операций осуществляются над дробными числами. Чтобы избежать значительной потери производительности при оперировании дробными числами, не следует для их программной реализации использовать числа с плавающей запятой. Дело в том, что числа с плавающей запятой хранятся в специальном формате, в котором мантисса и порядок представлены в зашифрованном виде, поэтому перед использованием таких чисел их надо расшифровать и нормализовать. В силу этих обстоятельств, на компьютере, оснащенном математическим сопроцессором, операции над числами с плавающей запятой требуют в 2 раза больше времени, чем над числами с фиксированной запятой. Поэтому для реализации в программе операций над дробными числами настоятельно рекомендуется вместо операций над числами с плавающей запятой использовать операции над числами с фиксированной запятой. Желательно применять 32-разрядный формат данных и для дробной части выделить только 8 младших разрядов. Так как для программ построения виртуальных пространств не нужна очень высокая точность (5 и более цифр после запятой), то этих 8 разрядов будет вполне достаточно. Дополнительную информацию о программной реализации алгебры дробных чисел на основе формата чисел с фиксированной запятой, а также о выполнении основных операций над числами с фиксированной запятой можно найти в [2].

Приведенные алгоритмы могут быть применены как для создания системы построения виртуальных пространств, так и для разработки трехмерного программного интерфейса. Программная реализация описанных выше алгоритмов была выполнена на языке программирования Borland C++ 4.2. Так как второй вариант алгоритма предоставляет одинаковые возможности с первым вариантом при более высоком уровне производительности, то именно этот алгоритм рекомендуем использовать в качестве основы создаваемой программной системы.

Пример изображения виртуального пространства, формируемого программой, основанной на втором варианте алгоритма трассировки луча, приведен на рис. 4. Данное трехмерное изображение получено для вида сверху виртуального пространства, показанного на рис. 5. Здесь сегментам стен A, B и D поставлены в соответствие текстуры, приведенные на рис. 7, а сегменту стен C — текстура, приведенная на рис.6.

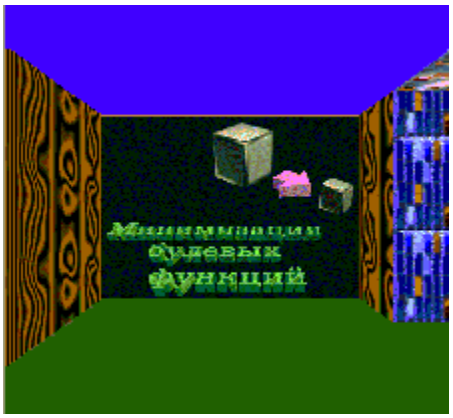


Рис. 4. Изображение на экране дисплея

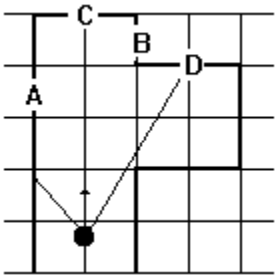


Рис. 5. Вид сверху виртуального пространства

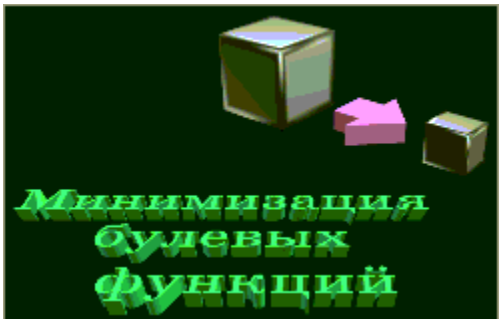
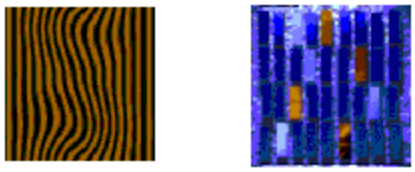


Рис. 6. Текстура для сегмента стен C



а б

Рис. 7. Текстуры для сегментов стен:
а – A и B; б – D

Заключение

В работе исследована проблема компьютерного моделирования трехмерных пространств в режиме реального времени. На примере идеализированной модели виртуального пространства выявлены основные этапы построения трехмерных пространств. Предложены два алгоритма, позволяющие создавать иллюзию трехмерного пространства, и рассмотрены особенности их программной реализации. Приведен пример изображения, созданного программой, основанной на втором варианте алгоритма трассировки луча. Указанная программа при предъявлении минимальных на сегодняшний день требований к ресурсам вычислительной системы (500Кб оперативной памяти, 2 Мб пространства на жестком диске, i386 микропроцессор) обладает следующими характеристиками: использование стандартных графических видеорежимов адаптера VGA с 256 цветами, что обеспечивает практически 100% переносимость программы; частота смены кадров составляет 30 кадров в секунду; возможность разбиения виртуального пространства, превышающего объем свободной оперативной памяти на несколько частей, загружаемых в заданной последовательности; возможность наполнения виртуального пространства неподвижными объектами произвольной формы; реализация открывающихся дверей и возможность работы с анимированными текстурами; поддержка 256-цветных спрайтов в форматах PCX и LBM.

Литература:1. *Белоус Н.В., Выродов А.П., Шубин И.Ю.* О некоторых алгоритмах построения виртуальных пространств // Проблемы бионики. 1998. №48. С. 52—59. 2. *Белоус Н.В., Выродов А.П., Шубин И.Ю.* Математические модели построения виртуальных пространств //Проблемы бионики. 1998. №49. С. 203—211. 3. *Ла Мот А., Ратклифф Дж., Семинаторе М., Тайлер Д.* Секреты программирования игр. СПб: Питер, 1995. 616 с.

Поступила в редколлегию 19.09.98

Рецензент: д-р техн. наук Петров Э.Г.

Белоус Наталия Валентиновна, канд. техн. наук, доцент кафедры ПОЭВМ ХТУРЭ. Научные интересы: разработка обучающих программ, гипертекстовых компьютерных учебников, моделирование сложных объектов. Адрес: Украина, 310093, Харьков, ул. Социалистическая, 68-5, кв. 65, тел. 72-76-47.

Шубин Игорь Юрьевич канд. техн. наук, доцент кафедры ПОЭВМ ХТУРЭ. Научные интересы: перспективные информационные технологии, трехмерная компьютерная графика. Адрес: Украина, 310170, Харьков, ул. Акад. Павлова, 134/16, кв. 263, тел. 68-64-89.

Выродов Александр Павлович, студент гр. ПОВТАС 96-1 ХТУРЭ. Научные интересы: трехмерная компьютерная графика. Адрес: Украина, 310726, Харьков, пр. Ленина, 14.

УДК 519.7

О ДЕЙСТВИЯХ С ЛИНЕЙНЫМИ ЛОГИЧЕСКИМИ ОПЕРАТОРАМИ

ГВОЗДИНСКАЯ Н.А., ДУДАРЬ З.В., ПОСЛАВСКИЙ С.А., ШАБАНОВ-КУШНАРЕНКО Ю.П.

Рассматривается затронутая в работах [4, 6] тема: линейные логические операторы и соответсвующие им матрицы. Описываются такие действия с линейными логическими операторами как дизъюнкция, произведение и умножение логического скаляра на линейный логический оператор.

Выберем в логическом пространстве L два линейных логических оператора $\boldsymbol{A}$ и $\boldsymbol{B}$ и произвольный вектор l. Применим последовательно к вектору l сначала оператор $\boldsymbol{A}$, затем оператор $\boldsymbol{B}$. В результате получим некоторый вектор

$$d=\boldsymbol{B}(\boldsymbol{A}l).$$

Оператор, переводящий вектор l в вектор d, называется произведением $\boldsymbol{B}$ на $\boldsymbol{A}$:

$$(\boldsymbol{BA})l=\boldsymbol{B}(\boldsymbol{A}l). \qquad (1)$$

Исходя из этого равенства, можно записать следующее, имея в виду, что α и β – некоторые логические скаляры, а $l, g\in L$:

$$(\boldsymbol{BA})(\alpha l\vee\beta g)=\boldsymbol{B}(\boldsymbol{A}(\alpha l\vee\beta g)).$$

В силу линейности операторов $\boldsymbol{A}$ и $\boldsymbol{B}$ имеют место равенства

$$\boldsymbol{B}(\boldsymbol{A}(\alpha l\vee\beta g))=\boldsymbol{B}(\alpha(\boldsymbol{A}l)\vee\beta(\boldsymbol{A}g))=\boldsymbol{B}(\alpha\boldsymbol{A}l)\vee\boldsymbol{B}(\beta\boldsymbol{A}g).$$

Иными словами,

$$(\boldsymbol{BA})(\alpha l\vee\beta g)=\alpha(\boldsymbol{BA})l\vee\beta(\boldsymbol{BA})g. \qquad (2)$$

Из равенства (2) следует, что произведение линейных логических операторов также является линейным оператором. Обозначим через C матрицу произведения линейных логических операторов $\boldsymbol{A}$ и $\boldsymbol{B}$. Она будет равна произведению матриц A и B, отвечающих операторам $\boldsymbol{A}$ и $\boldsymbol{B}$ соответственно, т.е.

$$C=BA. \qquad (3)$$

Для доказательства этого утверждения выберем в логическом пространстве L некоторый базис. Произвольному вектору $l\in L$ отвечает координатный столбец $[l]$ в выбранном базисе. Следовательно,

$$[(\boldsymbol{BA})l]=C[l]. \qquad (4)$$

В то же время

$$[(\boldsymbol{BA})l]=[\boldsymbol{B}(\boldsymbol{A}l)]=B[\boldsymbol{A}l]=BA[l]. \qquad (5)$$

Сравнивая равенства (4) и (5), получаем утверждение (3).

Например, возьмем вектор булева пространства размерности 4 [1, 3] $l=(0, 1, 1, 0)$, линейные логические операторы $\boldsymbol{A}$ и $\boldsymbol{B}$, которым отвечают логические матрицы