

УДК 371.385:681.3

Н.В. БЕЛОУС, А.П. ВЫРОДОВ, И.Ю. ШУБИН

## МАТЕМАТИЧЕСКИЕ МОДЕЛИ ПОСТРОЕНИЯ ВИРТУАЛЬНЫХ ПРОСТРАНСТВ

Построение пространственных графических изображений, синтез звуков, обработка больших объемов информации в реальном масштабе времени — все эти возможности, появившиеся благодаря интенсивному развитию вычислительной техники, находят применение практически во всех отраслях человеческой деятельности. Системы построения виртуальных пространств уже используются при компьютерных презентациях, создании компьютерных обучающих систем и трехмерных Web-страниц (подробную информацию о последних можно найти в [1; 2]). Кроме того, следует учесть, что 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.

В [3] рассмотрено несколько различных способов построения виртуальных пространств, включая применение интерактивных средств разработки, а также приведена общая схема оптимизированного алгоритма трассировки луча, который является ядром системы проектирования виртуальных пространств. В данной статье предложена математическая модель оптимизированного алгоритма трассировки луча, рассмотрены возникающие при ее использовании искажения и указан способ их устранения. Кроме того, изложен метод программной оптимизации, позволяющий существенно повысить скорость выполнения программы, реализующей данный алгоритм.

**Математические основы оптимизированного алгоритма трассировки луча.** В оптимизированном алгоритме трассировки луча для определения положения точки пересечения текущего луча с вертикальной стороной пересеченного квадрата используется функция тангенс от текущего угла просмотра. Другими словами, вычисляется угловой коэффициент трассируемого луча. Обозначим угол наклона текущего луча через  $\alpha$ , тогда угловой коэффициент  $K$  определяется по формуле

$$K = \operatorname{tg} \alpha = a / b, \quad (1)$$

где  $a$  — противоположный катет;  $b$  — прилежающий катет.

Так как используется графический режим 320 x 200 и угол обзора в 60°, угол наклона лучей может меняться не непрерывно, а дискретно с шагом в  $60/320 = 0,1875^\circ$ . Чтобы повысить производительность, необходимо заранее просчитать значения тангенса для всех возможных углов наклона и занести их в таблицу. После этого для получения требуемого значения тангенса достаточно извлечь его из таблицы, что оказывается гораздо быстрее непосредственного вычисления арифметическим сопроцессором. Полученная таким образом таблица будет содержать  $360/0,1875 = 1920$  элементов. Как известно, функция тангенс стремится к бесконечности при значениях аргумента в 90° и 270°. Поэтому надо ввести дополнительные проверки угла наклона, чтобы избежать в указанных точках ошибки деления на нуль.

Остановимся подробнее на вычислении координат точки пересечения трассируемого луча со стороной первого встретившегося на пути квадрата. Через  $X_1$  обозначим 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 пользователя. Тогда уравнение, описывающее трассируемый луч (уравнение прямой с угловым коэффициентом), представится в виде

$$(Y_1 - Y) / (X_1 - X) = K, \quad (2)$$

откуда

$$Y_1 = K(X_1 - X) + Y; \quad (3)$$

$$X_1 = K^{-1}(Y_1 - Y) + X. \quad (4)$$

Отметим, что каждое из полученных преобразований требует предварительного вычисления другого. Чтобы избежать такой зависимости, нужно в формулу (3) вместо абсциссы  $X_1$  подставить абсциссу любой другой точки, лежащей на трассируемом луче (рис. 1). В качестве абсциссы удобно взять абсциссу первой граничной вертикальной линии  $X'$ . Аналогично, вместо  $Y_1$  в формулу (4) подставим ординату  $Y'$  первой граничной горизонтальной линии (рис. 1). Теперь формулы (3) и (4) можно переписать в виде

$$Y_1 = K(X' - X) + Y; \quad (3a)$$

$$X_1 = K^{-1}(Y' - Y) + X. \quad (4a)$$

Формулы (3a) и (4a) полностью подходят для проведения вычислений, поскольку координаты  $X'$  и  $Y'$  могут быть легко определены исходя из следующих соображений.

Пусть координаты пользователя  $X$  и  $Y$  могут изменяться от 0 до 1023 и нужно найти абсциссу  $X'$  первой граничной вертикальной линии (см. рис. 1). Для этого разделим нацело координату  $X$  пользователя на размер стороны

квадрата, который положим равным 64 единицам. Полученная координата представляет собой абсциссу левой граничной вертикальной линии, а так как на рис. 1 вектор направления взгляда указывает вправо, необходимо увеличить найденную координату на размер стороны квадрата. Если бы вектор направления взгляда указывал влево, то необходимо было бы уменьшить эту координату на размер стороны квадрата. Аналогично находится ордината  $Y'$  первой граничной горизонтальной линии.

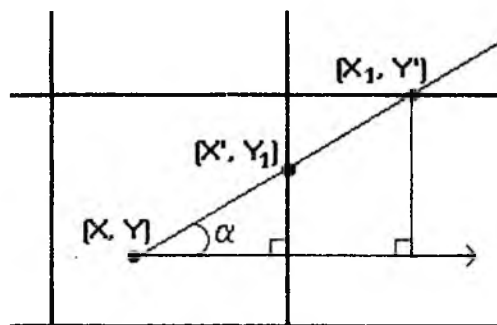


Рис. 1

Заметим, что в формулу (4а) коэффициент  $K$  входит в степени  $-1$ , а значит, требуется еще одна таблица — тангенсов всех углов в степени  $-1$ , т.е. таблица котангенсов всех углов.

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 (6) следующую координату  $Y$  точки, с которой пересечется трассируемый луч, если увеличить координату  $X$  на 64. Тем самым сразу пропускаем 64 точки карты, поскольку из ее построения следует, что там стена находиться не может. Аналогично, увеличивая координату  $Y$  на 64, по рекуррентной формуле (5) определяем координату  $X$  следующего возможного пересечения. Итак, координаты точек возможного пересечения, начиная со второй, вычисляются по следующим рекуррентным формулам:

$$X_{i+1} = X_i + K^{-1}c; \quad (5)$$

$$Y_{i+1} = Y_i + Kc, \quad (6)$$

где  $c$  — сторона квадрата.

Благодаря тому что сторона квадрата имеет фиксированный размер в 64 единицы, становится возможным составление еще двух таблиц — ре-

зультатов умножения значений тангенса и котангенса на 64, что позволит существенно повысить скорость выполнения программы.

Как только найдена X-стена, следует использовать координату Y, чтобы установить, какую колонку стены надо прорисовать, и координату X, чтобы определить расстояние до стены. Аналогично, если найдена Y-стена, то следует использовать координату X, чтобы установить, какую колонку стены надо прорисовать, и координату Y, чтобы определить расстояние до стены. Действительно, расстояние от точки с координатами (X,Y) до точки с координатами (X',Y') или от точки (X,Y) до точки (X<sub>1</sub>,Y') (см. рис. 1) можно отыскать двумя способами: 1) по теореме Пифагора, используя функцию извлечения корня; 2) с помощью известных соотношений между сторонами и тригонометрическими функциями углов прямоугольного треугольника. Как было указано в [3], извлечение корня неэффективно даже при предварительном составлении таблицы всех возможных значений корня. Поэтому целесообразно использовать тригонометрические функции от угла наклона  $\alpha$  трассируемого луча. Расстояние до точки пересечения трассируемого луча с горизонтальной стороной квадрата вычисляется по формуле

$$Dx = (X_i - X)\cos^{-1} \alpha, \quad (7)$$

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

Расстояние до точки пересечения трассируемого луча с вертикальной стороной квадрата находится следующим образом:

$$Dy = (Y_i - Y)\sin^{-1} \alpha, \quad (8)$$

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

Угол  $\alpha$  в формулах (7) и (8) является углом между трассируемым лучом и вектором направления взгляда. В программной реализации данного алгоритма угол  $\alpha$  — это индекс, изменяющийся в диапазоне от 0 до 1920, для таблиц заранее вычисленных значений функций  $\sin^{-1} \alpha$ ,  $\cos^{-1} \alpha$ ,  $\operatorname{tg} \alpha$ ,  $\operatorname{ctg} \alpha$ ,  $64\operatorname{tg} \alpha$ ,  $64\operatorname{ctg} \alpha$ .

**Проекционные искажения.** Алгоритм трассировки луча использует одновременно полярные и декартовы системы координат. Как следствие, это привело к появлению так называемых сферических искажений.

Действительно, все трассируемые лучи выходят из одной точки, и эквидистантные поверхности представляют собой концентрические сферы. Поэтому, чтобы построить стену, расположенную перпендикулярно к век-

тору направления взгляда, надо на карте, как на виде сверху, изобразить стену не прямой линией, а дугой окружности. Только в этом случае построенная стена будет выглядеть так, как на рис. 2, а.

При подготовке карты стены условно изображались в виде отрезков прямых линий, и расстояние до середины стены, расположенной перпендикулярно к вектору направления взгляда, всегда меньше расстояний до ее краев. Следовательно, высота колонки текстуры данной стены максимальна в ее середине и постепенно уменьшается при приближении к краям. Поэтому некомпенсированное изображение на экране дисплея имеет вид, показанный на рис. 2, б.

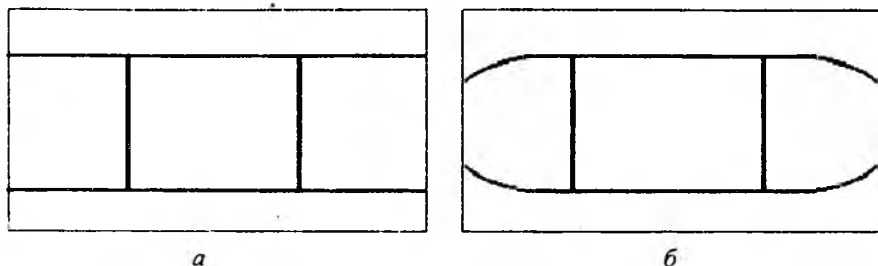


Рис. 2

Сферические искажения являются синусоидальными, и для их компенсации необходимо умножить высоту колонки стены на косинус текущего угла просмотра, т.е. высоту колонки стены надо вычислять по формуле

$$Ch = (\cos \alpha) 20\,000 / D, \quad (9)$$

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

**Реализация открывающихся дверей.** Открывающиеся двери реализованы следующим образом: если в процессе трассировки луч достигает двери, то трассировка луча продолжается дальше до пересечения со стеной или до выхода за пределы карты виртуального пространства. При поступлении от пользователя команды "открыть дверь" в специальном буфере выполняется прорисовка области, заслоненной закрытой дверью. Одновременно осуществляется сдвиг изображения двери влево (дверь убирается в косяк), а на освободившееся место выводится содержимое буфера. Это создает эффект открывания двери. Аналогичным образом возможна реализация любых полностью или частично прозрачных плоских областей (более подробная информация о реализации таких областей находится в [4]).

**Использование формата данных с фиксированной запятой.** Чтобы достичь значительного повышения производительности, надо заменить операции над числами с плавающей запятой на операции над числами с фиксированной 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, оснащенный математическим сопроцессором, операции над числами с плавающей запятой требуют в 2 раза больше времени, чем над числами с фиксированной запятой. Поэтому настоятельно рекомендуется использовать последние числа.

Для программной реализации алгебры чисел с фиксированной запятой необходимо знать, как выполняются основные операции над такими числами. Перейдем к рассмотрению этих операций.

Желательно использовать 32-разрядный формат данных и для дробной части выделить только 8 младших разрядов. Так как для построения виртуальных пространств не нужна очень высокая точность (5 и более цифр после запятой), то указанных 8 разрядов вполне достаточно. Действительно, наименьшее число, которое можно записать в таком формате, равно 0,004 и в большинстве случаев погрешности по абсолютному значению не выходят за пределы диапазона 0,01 — 0,5, что вполне допустимо, поскольку 90 % всех расчетов в программе направлено на определение местоположения пикселей на экране и, следовательно, результаты округляются до ближайшего целого. Операция присваивания выполняется различно для целой и дробной частей числа с фиксированной запятой. Для присваивания целой части надо предварительно умножить присваиваемое значение на 256 или сдвинуть его на 8 разрядов влево, чтобы оставить место для дробной части. Покажем, как это выглядит на языке программирования Borland C++:

```
int i = 500 ; long fixd = 0 ;
```

```
fixd = ( ( long ) i << 8 ) ;
```

Для присваивания дробных чисел необходимо произвести умножение чисел с плавающей точкой и записать результат в выделенные 32 разряда (в Borland C++ соответствующий формат называется long):

```
long fixd = ( long )( 23.45 * 256 ) ;
```

Здесь использовалась операция умножения на 256 вместо операции сдвига, поскольку последняя неприменима к числам с плавающей запятой.

Операции сложения и вычитания над числами с фиксированной запятой выполняются как и над целыми числами:

$$\text{fixd3} = \text{fixd1} + \text{fixd2}; \text{fixd3} = \text{fixd1} - \text{fixd2};$$

Вычитание и использование отрицательных чисел в Borland C++ возможны благодаря тому, что внутреннее представление чисел в формате long учитывает знак. Таким образом, и в данном 32-разрядном формате необходимо 1 бит выделить под знак. Под целую часть было выделено  $32 - 8 = 24$  разряда, и модуль максимального числа, которое может участвовать в операциях сложения и вычитания, равен  $2^{24-1} - 1 = 8\,388\,607$ .

Операция умножения над числами с фиксированной запятой выполняется как и над целыми числами, за исключением того, что результат должен быть сдвинут на 8 разрядов вправо:

$$\text{fixd1} = (\text{fixd2} \text{ fixd3}) >> 8;$$

Это нужно делать по следующей причине: когда инициализируется число с фиксированной точкой (выполняется операция присваивания), то оно умножается на 256 (см. выше); при умножении целая часть первого числа, умноженная на 256, умножается на целую часть второго числа, также умноженную на 256, поэтому результат оказывается дважды умноженным на 256. По этой же причине максимальное число с фиксированной точкой, которое может участвовать в операции умножения, равно 181, так как оно, будучи умноженным на 256 и возведенным в квадрат, может быть записано в выделенном для него 31 разряде.

Операцию деления целесообразно производить как операцию умножения делимого на величину, обратную делителю:

$$\text{fixd1} = (\text{long}) (256 * 1/34);$$

$$\text{fixd2} = (\text{fixd3} \text{ fixd1}) >> 8;$$

Пример изображения виртуального пространства, формируемого программой на экране дисплея, представлен на рис. 3. Данное трехмерное изображение получено при расположении пользователя, показанном на рис. 4.



Рис. 3

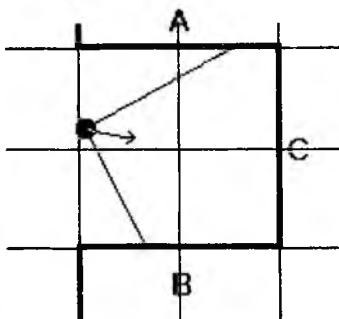


Рис. 4

Здесь сегментам стен А и В поставлена в соответствие текстура, приведенная на рис. 5, а сегменту стен С — текстура на рис. 6.

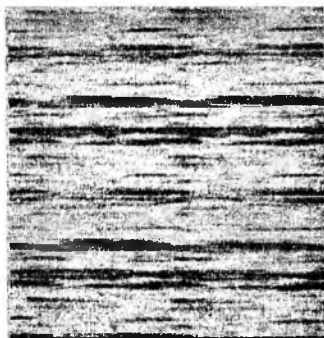


Рис. 5



Рис. 6

В заключение отметим, что в программной реализации описанного алгоритма, выполненной на языке программирования Borland C++ 4.2, используется несколько форматов чисел с фиксированной запятой, но все операции над ними производятся по изложенным выше правилам. Полученная программа при предъявлении минимальных на сегодняшний день



требований к ресурсам вычислительной системы (450 Кбайт оперативной памяти, 1 Мбайт пространства на жестком диске, i286-микропроцессор) имеет следующие особенности:

- использование стандартного VGA режима 320x200 с 256 цветами, что обеспечивает практически 100 %-ю переносимость программы;
- частоту смены кадров 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;
- реализацию открывающихся дверей и возможность работы с анимированными текстурами;
- поддержку 256 цветных спрайтов в форматах PCX и LBM.

Список литературы: 1. *Трехмерность в World Wide Web* // Компьютер Пресс. 1997. № 3. С. 84–93. 2. Татаринов О. VRML2.0: Виртуальная нереальность // Там же. С. 67–59. 3. Белоус Н.В., Выродов А.П., Шубин И.Ю. О некоторых алгоритмах построения виртуальных пространств // Проблемы бионики. 1998. № 48. С. 52–59. 4. *Секреты программирования игр*: Пер. с англ. / А. Ла Мот, Дж. Ротклифф, М. Семинаторе, Д. Тайлер. СПб.: Питер, 1995. 616 с.

Поступила в редколлегию 09.04.98