**Один метод построения полигональных изображений**

Василий Терешков

Построение изображений трехмерных объектов при помощи компьютера – тема, которая издавна привлекала особое внимание программистов и разработчиков аппаратных средств. С появлением эффективных графических библиотек (Direct3D, OpenGL и т.п.) и специализированных видеокарт интерес к математическим основам машинной графики снизился, поскольку у программистов исчезла необходимость самостоятельно создавать алгоритмы построения изображений. В этом одна из сторон печальной тенденции превращения программирования из искусства в ремесло.

Все же немало есть и тех, кто захочет не только получить результат, но и узнать, что лежит между интерфейсом графической библиотеки и готовой картинкой на экране. Для них и предназначена эта статья, в которой мы постараемся изложить суть одного метода построения трехмерных изображений, быть может, не самого эффективного.

**Терминология**

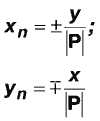
Прежде поясним некоторые математические понятия, которыми будем пользоваться в дальнейшем.

Мировая система координат – в нашем случае пространственная прямоугольная система координат (СК), две оси которой (X и Y) направлены по сторонам экрана монитора, а третья – от наблюдателя.

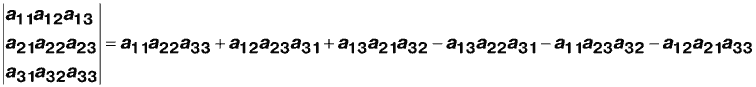
Экранная система координат – СК в плоскости экрана, ее оси совпадают с осями X и Y мировой СК.

Система координат модели – СК, относительно которой в файле заданы координаты всех вершин модели, изображение которой строится.

Вектор – направленный отрезок, его положение будем задавать либо координатами начала и конца, либо их разностями (собственно координатами вектора). Длина (модуль) вектора рассчитывается как квадратный корень из суммы квадратов его координат – это следствие теоремы Пифагора. Скалярное произведение векторов – число p, определяемое следующим образом: или, где |A| и |B| - длины векторов A и B, x, y, z – их координаты, t – угол между ними. Коллинеарные векторы – два или более вектора, лежащие на одной прямой или параллельных прямых. Компланарные векторы – три или более вектора, которые при отложении из одной точки оказываются лежащими в одной плоскости. Если векторы A, B, C компланарны, то вектор C можно разложить по векторам A и B, то есть C=aA+bB, где a и b – некоторые коэффициенты. Нормаль к вектору – вектор единичной длины, перпендикулярный данному. На плоскости координаты нормали к вектору P(x; y) определяются по формулам:



Определитель – алгебраическое выражение, записанное в особой форме. Мы будем использовать определители 3-го порядка:



Существует мнемоническое правило вычисления определителей 3-го порядка – так называемое правило Саррюса, с которым можно ознакомиться в специальной литературе.

**Используемые данные и их представление**

Возможно, вы обратили внимание на слово «полигональный» в заголовке статьи. Поясним его смысл. «Полигон» в переводе на русский язык означает «многоугольник», а «полигональный» – «составленный из многоугольников». В применении к машинной графике это означает, что для построения изображения произвольного тела сначала создается его модель – сложный многогранник, все грани которого представляют собой многоугольники, как правило, простейшие, - треугольники.

В файле с информацией о модели должны быть каким-либо образом заданы координаты всех вершин (их число может достигать нескольких тысяч) и порядок их соединения. Если предполагается наложение текстур, то каждой вершине должны быть приписаны еще два числа – текстурные координаты u, v. Их смысл в следующем. Текстура представляет собой плоское растровое изображение, которое должно быть наложено на пространственную модель без разрывов. Это предполагает неравномерную деформацию текстуры – ее сжатие и растяжение. Но одновременно требуется, чтобы текстура не «сползла», то есть во всех вершинах модели оказались строго определенные точки растра. Эти точки и задаются координатами u, v в системе координат, связанной с текстурой. Хорошей механической аналогией может послужить кусок резины, натягиваемый на каркас и прикрепляемый булавками в вершинах каркаса.

С технической точки зрения хранить все эти данные удобнее всего в двоичном файле, содержащем три массива:

|  |
| --- |
| //Информация о вершинах  struct TVertex  {  float x; //координаты в СК модели  float y;  float z;  } Vertices[NUM\_VERTICES];  //Информация о гранях  struct Ttriangle  {  int i1; //номера вершин, составляющих грань, в массиве Vertices  int i2;  int i3;  float u1; //текстурные координаты вершин  float v1;  float u2;  float v2;  float u3;  float v3;  } Triangles[NUM\_TRIANGLES];  //Текстура (256 цветов)  unsigned char Texture[TEXTURE\_SIZE]; |

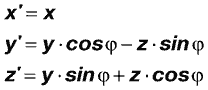
Отдельно требуется указать ракурс, под которым будет видна модель. Наиболее удобным для пользователя было бы задание оси вращения в виде вектора и угла поворота вокруг нее. Однако значительно проще реализовать последовательные повороты по трем углам: вокруг оси X, вокруг оси Y’, в которую перешла ось Y при первом повороте, вокруг оси Z’’, в которую перешла ось Z’ при втором повороте.

**Алгоритм построения изображения**

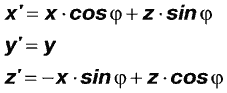
Изображение модели строится по отдельным граням, а изображение грани – по отдельным точкам, для каждой из которых определяется цвет. При этом, во-первых, закрашены должны быть все точки внутренней области изображения, во-вторых, цвет точки должен рассчитываться только один раз, что накладывает некоторые ограничения на выбор алгоритма.

Первым этапом построения изображения треугольной грани будет определение координат ее вершин в мировой СК и, в частности, их положения на экране, для чего требуется повернуть СК модели на заранее заданные углы (см. выше). Наиболее изящно такой поворот осуществляется умножением радиуса-вектора вершины на матрицу поворота. Мы же опишем его в терминах обычной координатной геометрии с применением формул поворота «плоской» (!) СК. Пусть x, y, z – начальные, а x’, y’, z’ - конечные координаты вершины, ТАУ – угол поворота, тогда эти формулы приобретают вид:

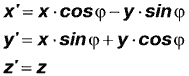
вокруг оси X:



вокруг оси Y:



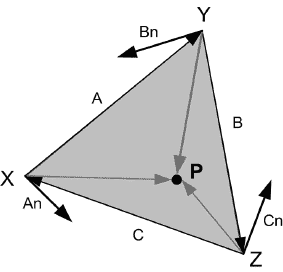
вокруг оси Z:



Следует помнить, что наша задача требует осуществлять все три поворота последовательно и при каждом новом повороте использовать в качестве начальных координат те, что получены при предыдущем.

Перейдем ко второму, не менее важному этапу. После того, как контур грани на экране определен, нужно найти все точки (пиксели), лежащие внутри него, иными словами, решить классическую задачу о принадлежности точки внутренней области треугольника. Один из вариантов ее решения (найденный автором статьи) таков. Представим контур грани составленным из векторов, а не отрезков (см. рисунок 1). К каждому из них проведем нормаль. Знаки координат вектора нормали выберем так, чтобы он был направлен в сторону противоположной вершины. Тогда внутри треугольника будут находиться те и только те точки, для которых все три скалярных произведения вектора, проведенного из какой-либо вершины в эту точку, и нормали, проведенной из той же вершины, положительны.

Например, на приведенном рисунке точка P лежит внутри треугольника, поскольку выполняются соотношения (здесь и далее заглавными латинскими буквами будем обозначать точки и векторы, а строчными – координаты):

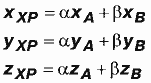


Для каждой найденной таким образом точки нужно определить ее видимость. Для этого используем широко распространенный метод z-буфера. Буфер представляет собой массив вида

|  |
| --- |
| float ZBuffer[SCREEN\_WIDTH][SCREEN\_HEIGHT]; |

Каждой точке на экране (пикселю) соответствует один элемент массива, а его значение трактуется как «глубина» этой точки, иными словами, ее координата z в мировой СК. Перед выводом точки ее «глубина» сравнивается с текущим значением в массиве и, если оказывается меньше его, записывается на его место и точка выводится на экран. Таким образом, видимой среди всех точек с одинаковыми координатами x и y оказывается та, у которой координата z минимальна.

Внимательный читатель заметит, что на предыдущем этапе задачу о взаимном расположении точки и треугольника мы решали в плоскости экрана и ни для одной из проверяемых точек координата z вообще неизвестна. Зато известны координаты z вершин треугольной грани, а кроме того, тот очевидный факт, что любая из точек (пусть это будет все та же точка P на рисунке вверху) лежит в плоскости грани. Следовательно, векторы A, C, XP (можно выбрать и другие тройки) компланарны и

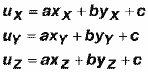


Эта система с неизвестными a, b, Zxp легко решается методом подстановки. Сложив Zx и Zxp, мы получим координату z точки P.

Теперь, если мы убедились, что точка находится внутри треугольника и она не заслонена другими точками, можно приступить к последнему этапу – определению ее цвета исходя из текстурных координат вершин грани, то есть, по сути, отысканию текстурных координат этой точки. При наложении на грань текстура деформируется – растягивается или сжимается – но так, что при этой деформации прямые линии остаются прямыми. Такое преобразование плоскости называется аффинным и задается уравнениями вида

|  |
| --- |
| x’ = ax + by + c;  y’ = dx + ey + f. |

Приведенные уравнения справедливы для координат любой точки, в том числе и для вершин, а значит, если мы, например, хотим найти координату u точки P, то должны сначала определить a, b, c, решив систему уравнений



Для решения таких систем часто применяют так называемое правило Крамера. Пусть F11 - определитель, полученный выписыванием коэффициентов перед неизвестными в правой части так, как они расположены в системе, а F12 - определитель, полученный из F11 заменой i-го столбца на столбец свободных членов (левая часть системы). Тогда i-е неизвестное рассчитывается по формуле



Так находятся числа a, b, c и, аналогично, d, e, f, которые затем применяются для расчета текстурных координат точки P. Далее цвет точки текстуры с этими координатами переносится в точку P на экране. Построение точки завершено.

**Недостатки концепции**

Рассмотренный нами метод работоспособен и вполне надежен. Но у него есть и существенные недостатки, относящиеся, в первую очередь, к скорости построения изображения. Так, изображение модели, состоящей из 1250 граней, на компьютере с процессором Celeron с тактовой частотой 1,3 ГГц строится за 1,5 секунды. Ясно, что для применения в практических задачах метод требуется оптимизировать, прежде всего, уменьшением количества операций умножения и деления чисел.

Настораживают также и требования к объему оперативной памяти: один только z-буфер, являющийся, по сути, вспомогательной структурой, в графическом режиме 640х480 точек потребует 1,2 Мб.

Замечу, что на настоящий момент описанный алгоритм реализован на языке высокого уровня, применение ассемблера смогло бы несколько увеличить его быстродействие.