Вычисление базисных векторов касательной плоскости для произвольного меша

Eric Lengyel • March 15, 2004

Современное отображение неровностей [прим.: bump mapping] (также известное как отображение нормалей [прим.: normal mapping]) требует вычисления базисных векторов касательной плоскости для каждой вершины меша. В этой статье представлена теория, лежащая в основе вычисления касательных пространств для каждой вершины произвольной триангулирванной сетки [прим.: mesh], и приведен исходный код, реализующий соответствующую математику.

[Прим.: впервые это описание появилось в *Mathematics for 3D Game Programming & Computer Graphics*, 1я ред., 2001., Обновлённое описание появилось в *Foundations of Game Engine Development, Volume 2: Rendering*, 2019.]

Математическое описание

Мы хотим, чтобы наше касательная плоскость(пространство) располагалась так, чтобы ось x соответствовала оси u карты неровностей [прим.: bump map], а ось y - оси v. Т.е. если $\boldsymbol{Q}$ - точка внутри треугольника можно было бы записать:

$$\boldsymbol{Q} - P_0 = (u - u_0)\boldsymbol{T} + (v - v_0)\boldsymbol{B},$$

где P_0 – позиция одной из вершин треугольника и (u_0, v_0) – текстурные координаты этой вершины. Вектора $\boldsymbol{T}$ и $\boldsymbol{B}$ –

касательный(tangent) и бикасательный(bitangent) векторы, направленные соответственно текстуре, они есть то, что нам требуется найти.

Пусть имеется треугольник с вершинами P_0, P_1 и P_2, которым соответствуют текстурные координаты (u_0, v_0), (u_1, v_1) и (u_2, v_2). Наши вычисления будут проще, если будут произведены относительно точки P_0, так что пусть:

$$Q_1 = P_1 - P_0,$$
$$Q_2 = P_2 - P_0,$$

а также:

$$(s_1, t_1) = (u_1 - u_0, v_1 - v_0),$$
$$(s_2, t_2) = (u_2 - u_0, v_2 - v_0)$$

Для поиска $\boldsymbol{T}$ и $\boldsymbol{B}$ необходимо решить следующие уравнения:

$$Q_1 = s_1 T + t_1 B,$$
$$Q_2 = s_2 T + t_2 B\ .$$

Это система линейных уравнений о 6ти неизвестных (по три для $\boldsymbol{T}$ и $\boldsymbol{B}$) и из шести уравнений (компоненты x, y и z в каждом, из двух, векторных уравнений). Её можно записать в матричной форме:

$$\begin{bmatrix} (Q_1)_x & (Q_1)_y & (Q_1)_z \\ (Q_2)_x & (Q_2)_y & (Q_2)_z \end{bmatrix} = \begin{bmatrix} s_1 & t_1 \\ s_2 & t_2 \end{bmatrix} \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix}$$

Что даёт нам ненормированные вектора $\boldsymbol{T}$ и $\boldsymbol{B}$ для треугольника, заданного вершинами P_0, P_1 и P_2. Что бы вычислить касательный(tangent) вектор для отдельной вершины нам необходимо усреднить касательные вектора для всех треугольников на эту вершину опирающихся, так же как это делается в случае усреднения вектора нормали. В случае если соседние треугольники имеют разрывы в текстурировании вершины на границе текстурирования уже продублированы поскольку, в любом случае, имеют другие текстурные координаты. Мы не усредняем касательные к таким треугольникам, потому что результат не будет точно отражать ориентацию карты неровностей для любого из треугольников.

Поскольку у нас есть вектор нормали $\boldsymbol{N}$ и касательные вектора $\boldsymbol{T}$ и $\boldsymbol{B}$ к вершине, то мы можем перейти из касательного пространства в объектное используя матрицу:

$$\begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix}$$

Для совершения обратного преобразования (из объектного пространства в касательное – то, что нам необходимо для вычисления направления света), нам потребуется использовать матрицу обратную этой. Касательным векторам не всегда требуется быть перпендикулярными друг другу или вектору нормали, так что обратная матрица, в общем, не равна транспонированной. Однако можно полагать, что три вектора $\boldsymbol{N}$, $\boldsymbol{T}$ и $\boldsymbol{B}$ близки ко взаимной ортагональности, так что применение алгоритма Грамма-Шмидта для их ортагонализации не приведёт к неприемлемым искажениям. Используя этот процесс новые, всё ещё не нормализованные, вектора $\boldsymbol{T}$ и $\boldsymbol{B}$ выражаются так:

$$T' = T - (N \cdot T)N$$

$$B' = B - (N \cdot B)N - \frac{(T' \cdot B)T'}{T'^2}$$

Нормализовав эти векторы и сохранив как касательный и бикасательный вектора для вершины будем использовать матрицу:

$$\begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix} \qquad (*)$$

для преобразования направления света из объектного пространства в касательное. Взятие векторного произведения преобразованного вектора направления света со значением, взятым из карты нормалей (bump map), даёт корректное Ламбертово значение рассеянного света.

Нет необходимости выделять память и где-то хранить бикасательный вектор B для каждой вершины поскольку его всегда можно вычислить через векторное произведение $N \times T' = mB$, где $m = \pm 1$ и является указанием на произвольность ориентации системы координат (левая или правая). Значение m должно храниться для каждой вершины, потому что найденный вектор $B' = N \times T'$ может указывать в неправильном направлении. Значение $m = \det(*)$. Возможно будет удобным хранить касательный вектор как 4х мерный вектор, у которого w компонент как раз и есть это значение m. Тогда значение бикасательного вектора может быть вычислено по формуле

$$B' = T_w'(N \times T')$$

в которой векторное произведение не учитывает w компонент. Это прекрасно работает для вершинных шейдеров в силу отсутствия необходимости указывать ещё один массив для каждой вершины, содержащий значения m.

Бикасательный вектор и бинормаль

Термин бинормаль обычно используется как обозначение второго касательного вектора (перпендикулярного первому и лежащего в касательной плоскости). Это неверное название. Термин "бинормаль" появляется при изучении кривых и завершает так называемый репер Френе, касающийся определенной точки на кривой. Кривые имеют одно направление касательной и два ортогональных направления нормали, отсюда и термины "нормаль" и "бинормаль". При рассмотрении системы координат в точке на поверхности существует одно нормальное направление и два касательных направления, которые следует называть касательным(тангентом) и бикасательным(битангентом).

Исходный код

```
#include "TSVector4D.h"

struct Triangle
{
    unsigned short  index[3];
};

void CalculateTangentArray(long vertexCount, const Point3D *vertex,
        const Vector3D *normal, const Point2D *texcoord,
        long triangleCount, const Triangle *triangle,
        Vector4D *tangent)
{
    Vector3D *tan1 = new Vector3D[vertexCount * 2];
    Vector3D *tan2 = tan1 + vertexCount;
    ZeroMemory(tan1, vertexCount * sizeof(Vector3D) * 2);

    for (long a = 0; a < triangleCount; a++)
    {
        long i1 = triangle->index[0];
        long i2 = triangle->index[1];
        long i3 = triangle->index[2];

        const Point3D& v1 = vertex[i1];
        const Point3D& v2 = vertex[i2];
        const Point3D& v3 = vertex[i3];
```

```
        const Point2D& w1 = texcoord[i1];
        const Point2D& w2 = texcoord[i2];
        const Point2D& w3 = texcoord[i3];

        float x1 = v2.x - v1.x;
        float x2 = v3.x - v1.x;
        float y1 = v2.y - v1.y;
        float y2 = v3.y - v1.y;
        float z1 = v2.z - v1.z;
        float z2 = v3.z - v1.z;

        float s1 = w2.x - w1.x;
        float s2 = w3.x - w1.x;
        float t1 = w2.y - w1.y;
        float t2 = w3.y - w1.y;

        float r = 1.0F / (s1 * t2 - s2 * t1);
        Vector3D sdir((t2 * x1 - t1 * x2) * r,
                      (t2 * y1 - t1 * y2) * r,
                      (t2 * z1 - t1 * z2) * r);
        Vector3D tdir((s1 * x2 - s2 * x1) * r,
                      (s1 * y2 - s2 * y1) * r,
                      (s1 * z2 - s2 * z1) * r);

        tan1[i1] += sdir;
        tan1[i2] += sdir;
        tan1[i3] += sdir;

        tan2[i1] += tdir;
        tan2[i2] += tdir;
        tan2[i3] += tdir;

        triangle++;
    }

    for (long a = 0; a < vertexCount; a++)
    {
        const Vector3D& n = normal[a];
        const Vector3D& t = tan1[a];

        // Gram-Schmidt orthogonalize
        tangent[a] = (t - n * Dot(n, t)).Normalize();

        // Calculate handedness
        tangent[a].w =
            (Dot(Cross(n, t), tan2[a]) < 0.0F) ? -1.0F : 1.0F;
    }

    delete[] tan1;
}
```