Публикация

slupoke не публиковался

История о том как не столкнулись два куба, или теорема о разделяющей оси

Разработка игр, C#, Математика

Из песочницы

Теорема о разделяющей оси

Обратил внимание, что на хабре мало публикаций по вычислительной геометрии, поэтому я решил поделиться своим опытом в решении задач связанных с определением коллизий двух выпуклых геометрий.

Примечание 1: в статье будет приведен пример с 2 параллелепипедами(далее — кубы), но идея для других выпуклых объектов будет сохранена.

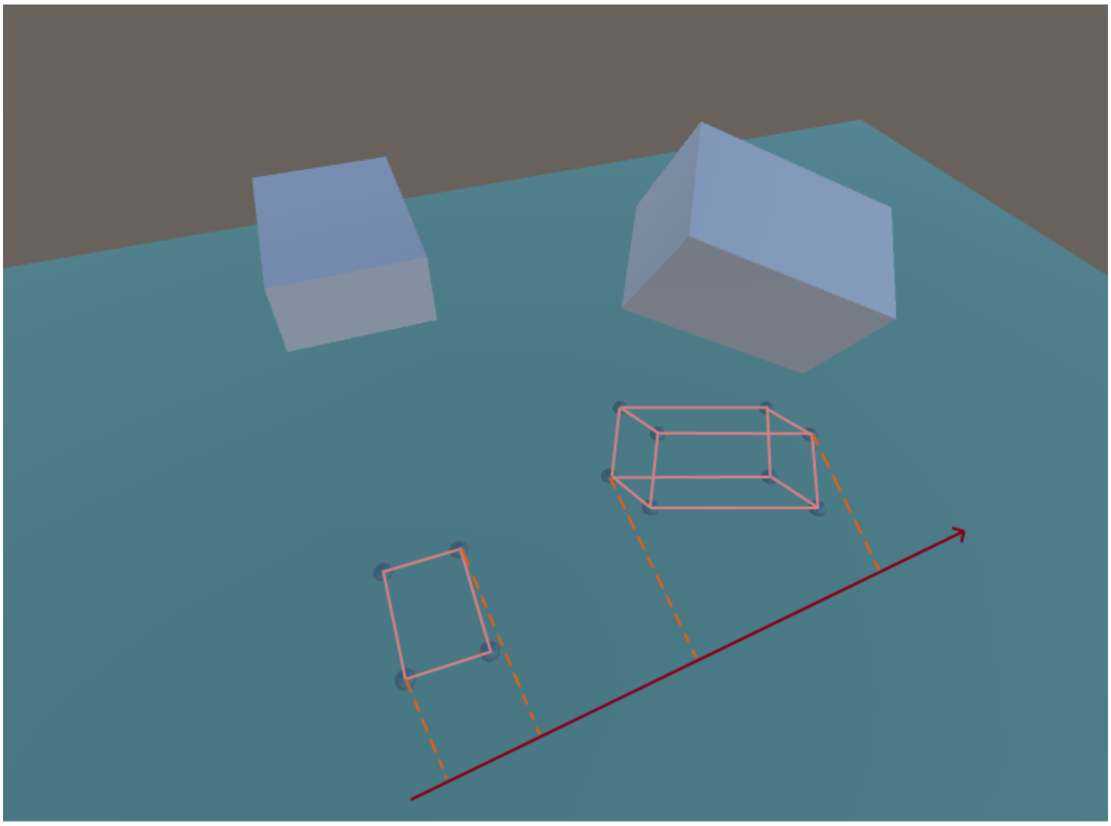
Акт 0. Теорема о разделяющей плоскости

Для начала нужно познакомиться с "теоремой о разделяющей гиперплоскости".
Именно она будет **лежать в основе алгоритма**.

Для общего случая, теорема звучит так: **Две выпуклые геометрии не пересекаются, тогда и только тогда, когда существует гиперплоскость, которая их разделяет, тогда проекции фигур на ортогональную разделяющей гиперплоскость — не пересекаются.**

Для нашего случая, теорему можно озвучить следующим образом: **Два куба не пересекаются, тогда и только тогда, когда существует ось, проекции геометрий на которую не пересекаются**. Данная ось называется разделяющей.

Можно заметить, что если проекции на плоскость не пересекаются, тогда в этой плоскости существует вектор, на который проекции тоже не пересекаются.



Потенциальная разделяющая ось будет находиться в следующих множествах:
• Нормы плоскостей каждого куба(красные)
• Векторное произведение ребер кубов $\{(\vec{x}, \vec{y}) : x \in X, y \in Y\}$,
где X — ребра первого куба(зеленые), а Y — второго(синие).

Для того, чтобы определить плоскости и ребра, нам понадобиться получить координаты вершин каждого куба.

Каждый куб мы можем описать следующими входными данными:
• Координаты центра куба
• Размеры куба(высота, ширина, глубина)
• Кватернион куба

Акт 1. Нахождение вершин куба

Как часто бывает, объект может вращаться в пространстве, для того, чтобы найти координаты вершин, с учетом вращения куба, необходимо понять, что такое **кватернион**.

Кватернион — это гиперкомплексное число, которое определяет вращения объекта в пространстве.

$$w + xi + yj + zk$$

Мнимая часть(x,y,z) представляет вектор, который определяет направление вращения
Вещественная часть(w) определяет угол на который будет совершено вращения.

Его основное отличие от всем привычных углов Эйлера в том, что нам достаточно иметь **один** вектор, который будет определять направление вращения, чем **три** линейно независимых вектора, которые вращают объект в 3 подпространствах.

Рекомендую две статьи, в которых подробно рассказывается о кватернионах:
Раз
Два

Теперь, когда у нас есть минимальные представления о кватернионах, давайте поймем, как вращать вектор, и опишем функцию вращение вектора на кватернион.

Формула вращения вектора

$$\vec{v}' = q * \vec{v} * \overline{q}$$

$\vec{v}'$ — искомый вектор
$\vec{v}$ — исходный вектор
q — кватернион
$\overline{q}$ — обратный кватернион

Для начала, дадим понятие обратного кватерниона в **ортонормированном базисе — это кватернион с противоположным по знаку мнимой частью**
$q = w + xi + yj + zk$
$\overline{q} = w - xi - yj - zk$

Теперь посчитаем $\vec{v} * \overline{q}$
Если вектор $\vec{v}$ представить в виде кватерниона с нулевой действительной частью
$\vec{v} = 0 + xi + yj + zk$
то мы получим обычное произведение кватернионов
$M = \vec{v} * \overline{q} = (0 + v_x i + v_y j + v_z k)(q_w - q_x i - q_y j - q_z k) =$
$= v_x q_w i + v_x q_x - v_x q_y k + v_x q_z j +$
$+ v_y q_w j + v_y q_x k + v_y q_y - v_y q_z i +$
$+ v_z q_w k - v_z q_x j + v_z q_y i + v_z q_z$

$M = u_w + u_x i + u_y j + u_z k$
$u_w = v_x q_x + v_y q_y + v_z q_z$
$u_x i = (v_x q_w - v_y q_z + v_z q_y) i$
$u_y j = (v_x q_z + v_y q_w - v_z q_x) j$
$u_z k = (-v_x q_y + v_y q_x + v_z q_w) k$

Посчитаем оставшуюся часть, в которой мы должны будем получить искомый вектор. Вспомним определение кватерниона и из результата данного произведения уберем вещественную часть, тем самым останется только векторная(мнимая) часть.
$\vec{v}' = q * M = (q_w + q_x i + q_y j + q_z k)(u_w + u_x i + u_y j + u_z k) =$
$= q_w u_x i + q_w u_y j + q_w u_z k +$
$+ q_x u_w i + q_x u_y k - q_x u_z j +$
$+ q_y u_w j - q_y u_x k + q_y u_z i +$
$+ q_z u_w k + q_z u_x j + q_z u_y i$

Соберем компоненты вектора
$v'_x = q_w u_x + q_x u_w + q_y u_z + q_z u_y$
$v'_y = q_w u_y - q_x u_z + q_y u_w + q_z u_x$
$v'_z = q_w u_z + q_x u_y + q_y u_x + q_z u_w$

$v' = v'_x + v'_y + v'_z$
Таким образом необходимый вектор получен

Напишем код

```
public static Vector3 QuanRotation(Vector3 tmp,Quaternion q)
{
        //формула поворота
        // a = q*tmp*q^(-1)
        //a - новый вектор
        //q - кватернион
        //q^(-1) - обратный кватернион
        //tmp - начальный вектор

        float u0 = tmp.x * q.x + tmp.y * q.y + tmp.z * q.z;
        float u1 = tmp.x * q.w - tmp.y * q.z + tmp.z * q.y;
        float u2 = tmp.x * q.z + tmp.y * q.w - tmp.z * q.x;
        float u3 = -tmp.x * q.y + tmp.y * q.x + tmp.z * q.w;
        //M = tmp*q^(-1)
        Quaternion M = new Quaternion(u1,u2,u3,u0);

        //a = v*M
```

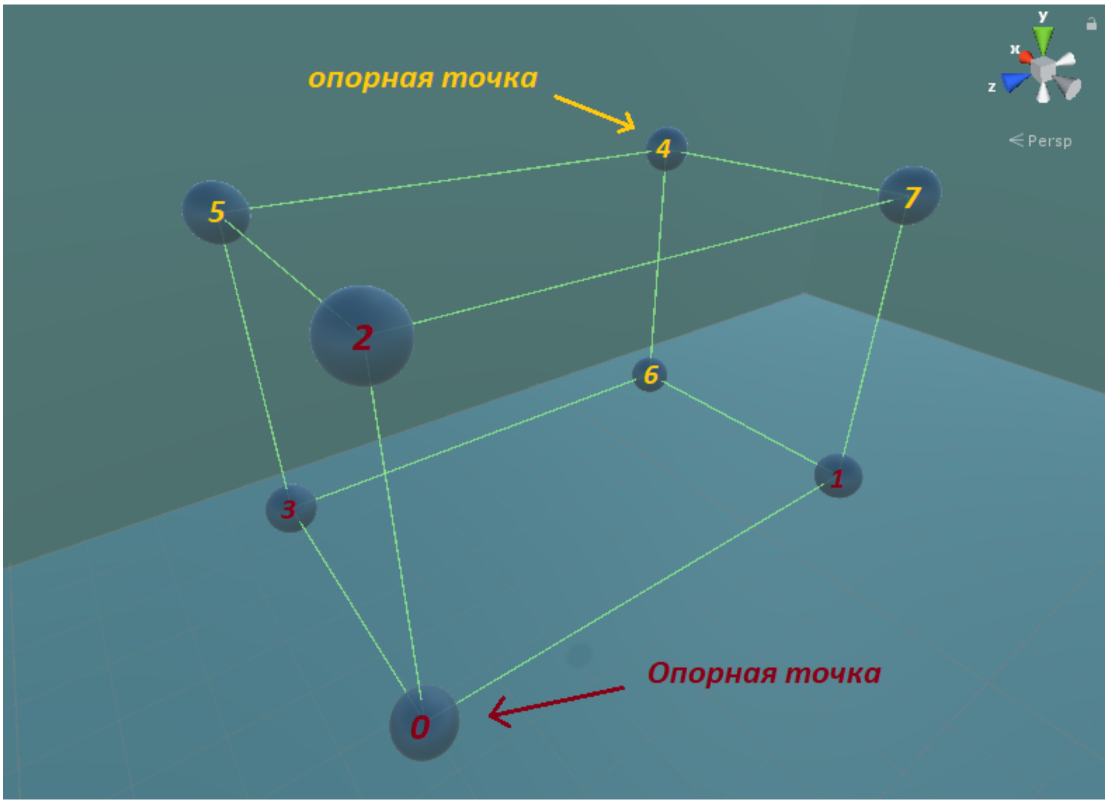
```
        Vector3 a;
        a.x = q.w * M.x + q.x * M.w + q.y * M.z - q.z * M.y;
        a.y = q.w * M.y - q.x * M.z + q.y * M.w + q.z * M.x;
        a.z = q.w * M.z + q.x * M.y - q.y * M.x + q.z * M.w;

        return a;
}
```

Перейдем к функцию нахождении вершин куба. Определим базовые переменные

```
public Vector3[] GetPoint(GameObject p)
{
        Vector3 center = p.transform.position;//центр куба
        Quaternion q = p.transform.rotation;//кватернион куба
        //размеры куба
        //где x,y,z соответствуют ширине, высоте и глубине
        Vector3 size = p.transform.localScale;

        //Тут будут храниться координаты вершин
        Vector3[] point = new Vector3[8];

        //получаем координаты
        //....

        return point;
}
```

Далее необходимо найти такую точку(опорную точку), от которой будет легче всего найти другие вершины.
Из центра покоординатно вычитаем половину размерности куба
Далее к опорной точке прибавляем по одной размерности куба

```
//...
        point[0] = center - size/2;
        point[1] = point[0] + new Vector3(size.x , 0, 0);
        point[2] = point[0] + new Vector3(0, size.y, 0);
        point[3] = point[0] + new Vector3(0, 0, size.z);

        //таким же образом находим оставшееся точки
        point[4] = center + size / 2;
        point[5] = point[4] - new Vector3(size.x, 0, 0);
        point[6] = point[4] - new Vector3(0, size.y, 0);
        point[7] = point[4] - new Vector3(0, 0, size.z);
//...
```



Можем видеть, как сформированы точки

После нахождения координат вершин, необходимо повернуть каждый вектор на соответствующий кватернион

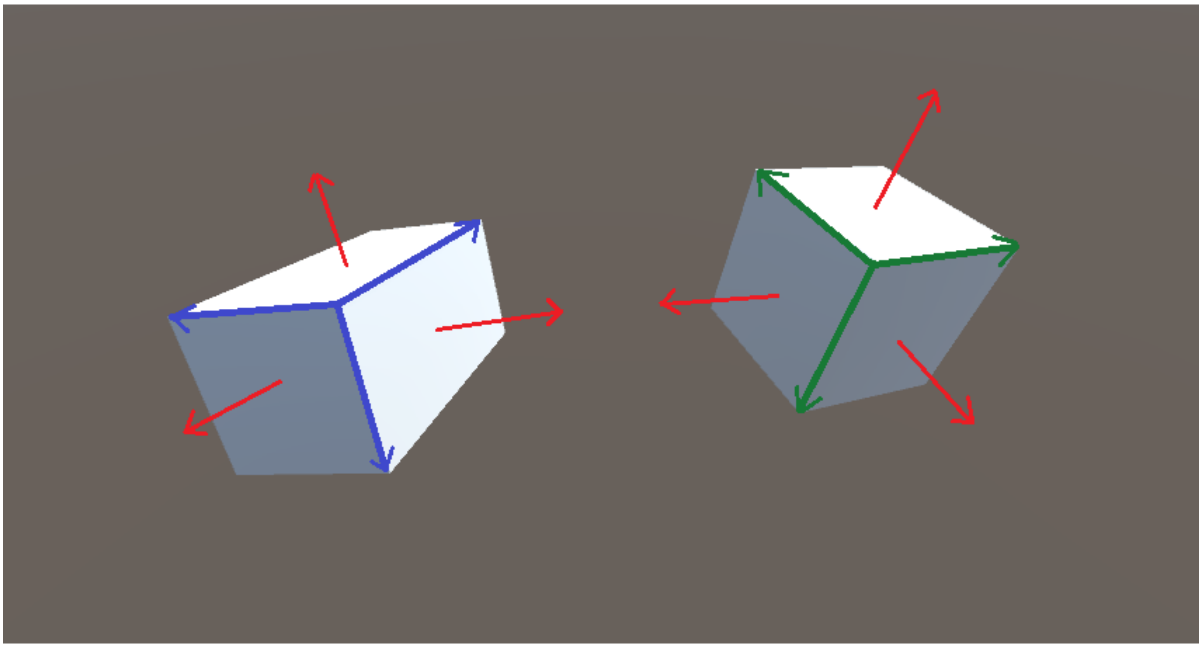
```
//...

        for (int i = 0; i < 8; i++)
        {
                //выполняем составное преобразование
                //чтобы поворот осуществлялся относительно центра координат
                point[i] -= center;//перенос центра в начало координат
                point[i] = QuanRotation(point[i], q);//поворот
                point[i] += center;//обратный перенос
        }

//...
```

▸ полный код получения вершин

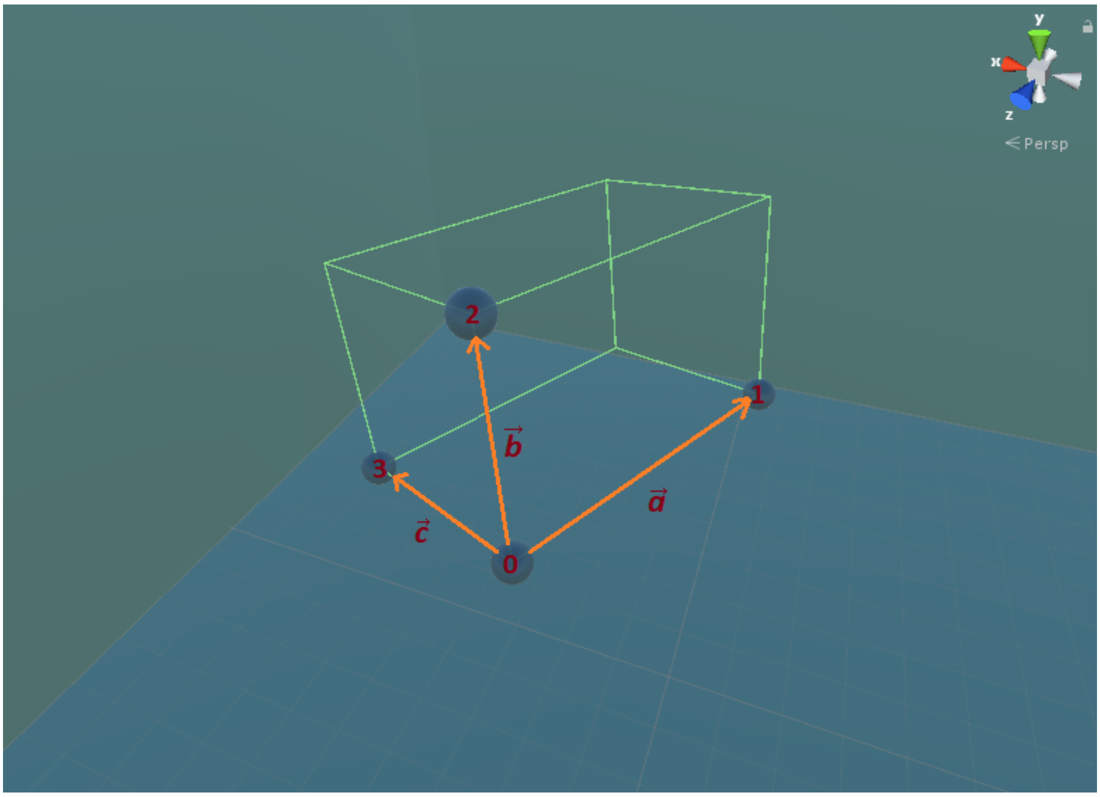
Перейдем к проекциям.

Акт 2. Поиск разделяющих осей

Следующим шагом необходимо найти оси, претендующие на разделяющую.
Вспомним, что ее можно найти в следующих множествах:
• Нормы плоскостей каждого куба(красные)
• Векторное произведение ребер кубов $\{(\vec{x}, \vec{y}) : x \in X, y \in Y\}$,
где X — ребра первого куба(зеленые), а Y — второго(синие).



Для того, чтобы получить необходимые оси, достаточно иметь первые четыре вершины куба



(удалить)

Заметим, что вектора $\vec{a}$, $\vec{b}$ и $\vec{c}$ формируют ортогональную систему векторов, они и будут определять наши плоскости и ребра.

(удалить)

Напишем функцию нахождения векторного произведения и по совместительству нормали:

```
public static Vector3 VectProduct(Vector3 a, Vector3 b)
{
        Vector3 result;
        result.x = a.y * b.z - a.z * b.y;
        result.y = a.z * b.x - a.x * b.z;
        result.z = a.x * b.y - a.y * b.x;
        return result;
}
```

Необходимо найти нормы плоскостей, порожденные векторами:

- $\vec{a}$ и $\vec{b}$
- $\vec{b}$ и $\vec{c}$
- $\vec{a}$ и $\vec{c}$

Для этого надо перебрать пары ребер куба так, что бы каждая новая выборка образовывала плоскость, ортогональную всем предыдущим полученным плоскостям. Для меня невероятно тяжело было объяснить как это работает, поэтому я предоставил два варианта кода, которые помогут понять.

▸ такой код позволяет получить эти вектора и найти нормы к плоскостям для двух кубов

Но можно сделать проще:

```
public static List<Vector3> GetAxis(Vector3[] a, Vector3[] b)
{
        //ребра формирующие плоскость
        Vector3 A;
        Vector3 B;

        //список, хранит потенциальные разделяющие оси
        List<Vector3> Axis = new List<Vector3>();

        //нормы плоскостей первого куба
        for (int i = 1; i < 4; i++)
        {
            A = a[i] - a[0];
            B = a[(i+1)%3+1] - a[0];
            Axis.Add(VectProduct(A,B).normalized);
        }

        //нормы второго куба
        for (int i = 1; i < 4; i++)
        {
            A = b[i] - b[0];
            B = b[(i+1)%3+1] - b[0];
            Axis.Add(VectProduct(A,B).normalized);
        }

        //...
}
```

Еще мы должны найти все векторные произведения ребер кубов. Это можно организовать простым перебором

```
public static List<Vector3> GetAxis(Vector3[] a, Vector3[] b)
{
        //...
        //получение норм
        //...


        //Теперь добавляем все векторные произведения
        for (int i = 1; i < 4; i++)
        {
            A = a[i] - a[0];
            for (int j = 1; j < 4; j++)
            {
                B = b[j] - b[0];
                //векторное произведение параллельных векторов игнорируем.
                if (VectProduct(A,B).magnitude != 0)
                {
                    Axis.Add(VectProduct(A,B).normalized);
                }
            }
        }
}
```

▸ Полный код

Акт 3. Проекции на оси

Мы подошли к самому главному моменту. Здесь мы должны найти проекции кубов на все потенциальные разделяющие оси. У теоремы есть одно важное следствие: если объекты пересекаются, то ось на которую проекции пересечения кубов минимальна — является направлением коллизии, а длинна отрезка пересечения — глубиной проникновения.

Но для начала напомним формулу проекции вектора ***v*** на единичный вектор ***a***:

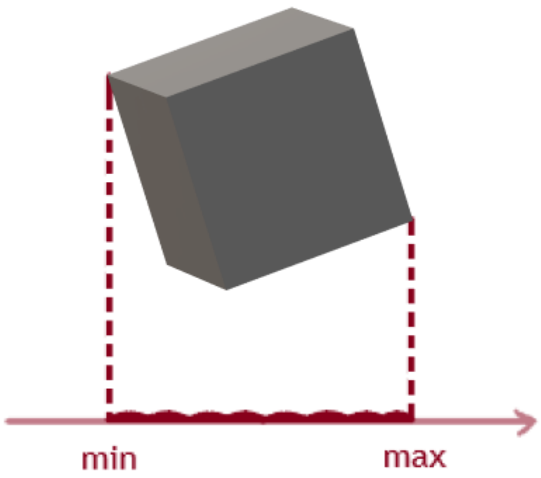
$$proj_{\langle\vec{a}\rangle}\vec{v} = \frac{(\vec{v},\vec{a})}{|\vec{a}|}$$

```
public static Vector3 ProjVector3(Vector3 v, Vector3 a)
{
        a = a.normalized;
        float alpha = Vector3.Dot(v, a) / a.magnitude;
        return alpha;

}
```

Теперь опишем функцию будет определять пересечение проекций на оси-кандидаты.
На вход получаем вершины двух кубов, и список потенциальных разделяющих осей

```
public static Vector3 ProjAxis(Vector3[] a, Vector3[] b, List<Vector3> Axis)
{
        for (int j = 0; j < Axis.Count; j++)
        {
           //в этом цикле проверяем каждую ось
           //здесь будем определять разделяющие оси из списка кандидатов
        }
        //Если мы в цикле не нашли разделяющие оси, то кубы пересекаются, и нам нужно
        //определить глубину и нормаль пересечения.
}
```

Проекция на ось задается двумя точками, которые имеют максимальные и минимальные значения на самой оси

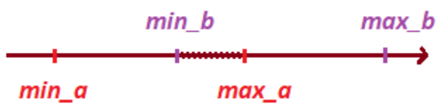


Используя написанную выше функцию нахождения проекции вектора на ось, найдем проекционные точки двух кубов.

```
            //...

            //проекции куба a
            float max_a = ProjVector3(a[0], Axis[j]);//значение вершины
            float min_a = ProjVector3(a[0], Axis[j]);//значение
```

```
            for (int i = 0; i < b.Length; i++)
            {
                float tmp = ProjVector3(a[i], Axis[j]);
                if (tmp > max_a)
                {
                    max_a = tmp;
                }

                if (tmp < min_a)
                {
                    min_a = tmp;
                }
            }

            //проекции куба b
            float max_b = ProjVector3(b[0], Axis[j]);
            float min_b = ProjVector3(b[0], Axis[j]);
            for (int i = 0; i < b.Length; i++)
            {
                float tmp = ProjVector3(b[i], Axis[j]);
                if (tmp > max_b)
                {
                    max_b = tmp;
                }

                if (tmp < min_b)
                {
                    min_b = tmp;
                }
            }

            //...
```

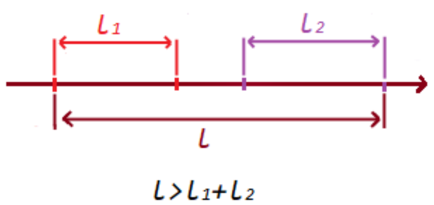
Получив проекционные точки каждого куба, мы должно определить пересечение проекций



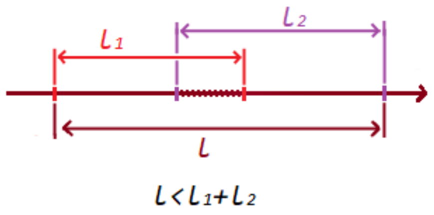
Для этого давайте поместим наши точки в массив и отсортируем его, такой способ поможет нам определить не только пересечение, но и глубину пересечения.

```
            float[] p = {min_a, max_a, min_b, max_b};
            Array.Sort(p);
```

Заметим следующее свойство:
1)Если отрезки **не пересекаются**, то сумма отрезков будет меньше, чем отрезок сформированными крайними точками



2)Если отрезки **пересекаются**, то сумма отрезков будет больше, чем отрезок сформированными крайними точками



Вот таким простым условием мы проверили **непересечение** отрезков, тем самым вернем глубину пересечения, которая равна нулю.

```
            //Сумма отрезков
            float sum = (max_b - min_b) + (max_a - min_a);
            //Длина крайних точек
            float len = Math.Abs(p[3] - p[0]);

            if (sum <= len)
            {
                // Результат: Непересек
                return Vector3.zero;
            }
```

Таким образом, нам достаточно иметь хоть один вектор, на котором проекции не пересекаются, тогда кубы не пересекаются. Поэтому когда мы нашли разделяющую ось, мы можем не проверят оставшееся вектора, и завершить работу алгоритма.

В случае пересечения кубов, все немного интереснее: проекции кубов на **все** вектора будут пересекаться, и мы должны определить вектор с минимальным пересечением.

Для этого, высчитываем вектор который будет иметь направление этого пересечения, а его длина будет глубиной пересечения. Создадим этот вектор перед циклом, и в нем будет храниться вектор с минимальной длинной. Тем самым в конце цикла получим искомый вектор.

```
    public static Vector3 ProjAxis(Vector3[] a, Vector3[] b, List<Vector3> Axis)
    {
        Vector3 norm = new Vector3(1000,1000,1000);
        //простым нахождение мин. и макс. точек куба по заданной оси
        for (int j = 0; j < Axis.Count; j++)
        {
            //...
        }
        //в случае, когда нашелся вектор с минимальным пересечением, возвращаем его
        return norm;
    {
```

И каждый раз когда мы находим ось, на которой проекции пересекаются, проверяем является ли она минимальной по длине среди всех. В случае успеха текущую ось умножаем на длину пересечения, и результатом будет искомая нормаль пересечения.
Так же я добавил определение ориентации нормали по отношению первого куба.

```
            if (sum <= len)
            {
                // Результат: Непересек;
                return new Vector3(0,0,0);
            }
            else
            {
                float dl = Math.Abs(p[2] - p[1]);//длина пересечения
                if (dl < norm.magnitude)
                {
```

```
                norm = Axis[j] * dl;
                //нахождение ориентации нормали
                if(p[0] != min_a)
                    norm = -norm;

            }
        }
```

Весь код

Теги: Вычислительная геометрия, разработка игр, определение коллизий

Хабы: Разработка игр, C#, Математика