Операции с векторами

Краткая справка

Http://hipnc.club

Скаляр – обычное число, имеющее только один компонент. Например, float или int.

Вектор состоит из нескольких компонентов. Чаще всего мы используем Vector3, то есть вектор, состоящий из X, Y и Z-компонент.

v[x]	v[y]	v[z]
1.28177	-0.286147	-0.353448

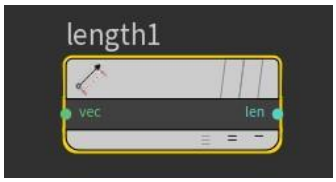
Вектор может быть координатами в пространстве (причем не обязательно в физическом, цвет – это тоже координаты в цветовом пространстве), а может быть направлением. Например, атрибут P – это координаты. Атрибуты N, v, up – это направления. Считается как направление из нуля координат в точку с координатами вектора. Это отличие чисто умозрительное, хотя некоторым функциям надо указывать, к какому типу относится вектор.

Так, если мы сделаем @N=@P; мы получим нормали, которые смотрят из центра координат.

Вектор раскладывается на две составляющие: длина и направление.

Длина вектора (модуль)

В формулах указывается как название вектора в вертикальных линиях |A| – это расстояние от нуля до координат вектора. Ее можно вычислить функцией ***length(vec)*** или нодой length в VOP.



Вектор с длиной, равной единице, называется нормированным, нормализованным, или ортом. Любой вектор можно нормализовать функцией ***normalize(vec)*** или соответствующей нодой в VOP. После этого длина вектора изменится до 1 (кроме случая, когда на вход подается вектор с длиной 0).

Если мы сделаем @P=normalize(@P); мы получим равномерное распределение точек по поверхности шара с радиусом 1. Потому что длина каждого вектора стала равной единице.

Изменить длину вектора можно, домножив его на флоат. Умножая на число больше единицы мы его удлиняем. Умножая на число от нуля до единицы мы его укорачиваем. Умножая на ноль, мы делаем длину вектора равной нулю. Умножая на отрицательное число, мы инвертируем направление вектора. Умножение на единицу ничего не меняет.

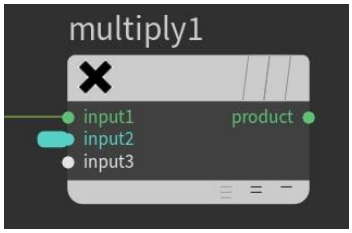
Примеры:

*v@v=@v*2; // увеличиваем скорость в два раза*

*v@v=@v*0.5; // уменьшаем скорость в два раза*

v@v=@v-1; // инвертируем скорость (направляем вектор обратно)*

Когда мы работаем в VOP важно на первый вход ноды Multiply подавать именно вектор, потому что тип данных в первом входе определяет тип данных на выходе.



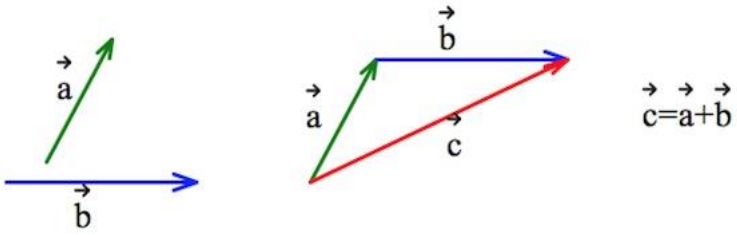
Если мы хотим сделать равномерное движение с известной скоростью, можно сделать:

*v@v=normalize(@v)*ch(“speed”);*

То есть сначала мы делаем длину вектора равной единице, а потом домножаем его на новую длину.

Сложение векторов (Add)

A+B=C

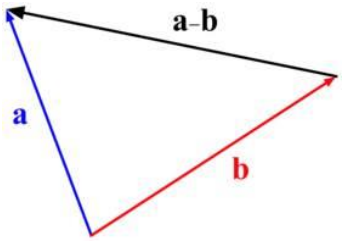


Если вектор B перенести в конец вектора A (или наоборот), то результат сложения – это вектор, проведенный из начала вектора A в конец вектора B. От перемены мест слагаемых сумма не изменится.

Стандартное использование: прибавление к позиции точки шума, прибавление к скорости векторов сил.

Вычитание векторов (Subtract)

A-B=C

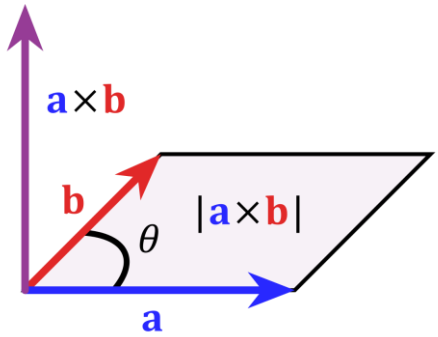


Результат вычитания векторов идет из конца второго вектора в конец первого. Если поменять аргументы местами, вектор будет идти в противоположном направлении.

Например, если вычесть координаты двух точек, то через функцию Length можно найти расстояние между ними.

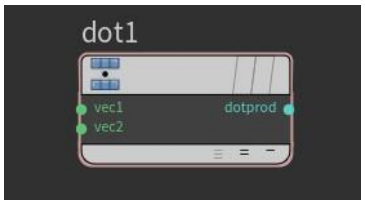
Cross Product или векторное умножение векторов

Cross(A,B) вернет вектор, перпендикулярный обоим входным векторам.



Если поменять аргументы местами, вектор поменяет направление на противоположное.

Dot Product или скалярное произведение векторов



Dot(A,B) = |A|*|B|*cos(угол между векторами)

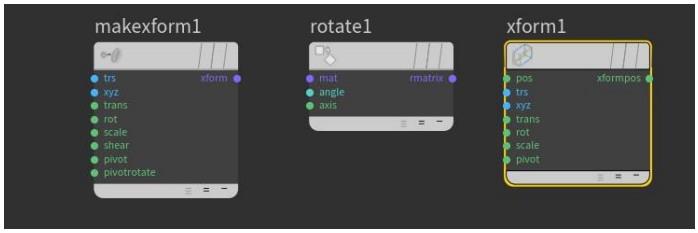
Значит, если векторы А и Б нормализованны, на выходе мы получим косинус угла между векторами. Так можно выяснить этот угол.

Результат	*Вектора*
1	Сонаправлены
-1	Противоположно направлены
0	Перпендикулярны

Или можно воспользоваться функцией acos() – в ВОП она находится в ноде Trigonometry, - чтобы найти сам угол.

Поворот векторов

Для поворота вектора надо его умножить на матрицу. Матрицы бывают Matrix3 – в них записан только поворот – и Matrix4 – в них записаны полные трансформации. К вектору также можно применять матрицу4 вместе со сдвигами и скейлом.



Для создания матриц можно использовать функции rotate или Make transform.

В rotate мы задаем ось вращения (обязательно нормализованную) и угол в радианах. В Make transform задаются обычные углы Эйлера.

ВОП-нода Transform matrix сразу домножает вектор на матрицу, так что на выходе получается измененный вектор.

Кватернионы

Кватернион – это способ однозначно задать поворот объекта в пространстве. Кватернион состоит из четырех компонентов (vector4), в которых записан вектор направления и угол поворота вокруг этого вектора.

В системе частиц кватернион представлен в виде атрибута ***orient,*** он приоритетнее всех других атрибутов поворота при работе ноды Copy to points и инстансинге.

В компоненты записываются следующие значения (вектор V и угол angle):

1. V.x*sin(angle/2)
2. V.y*sin(angle/2)
3. V.z*sin(angle/2)
4. Cos(angle/2)

При нулевом угле кватернион не будет ничего изменять, аналог матрицы идентичности.

В VOP есть некоторое количество функций для работы с кватернионами.

Quaternion позволяет задать ось и угол для создания кватерниона. Ось должна быть нормализована.

Rotate by quaternion позволяет применить кватернион к произвольному вектору.

Quaternion multiply позволяет применить вращения кватернионов последовательно.

Quaternion to vector u Vector to quaternion конвертируют в/из вектора, при этом угол поворота задается длиной вектора.

Quaternion to matrix u Matrix to quaternion – конверсия в/из матрицы поворота.

Euler to quaternion – конверсия из углов Эйлера в кватернион.