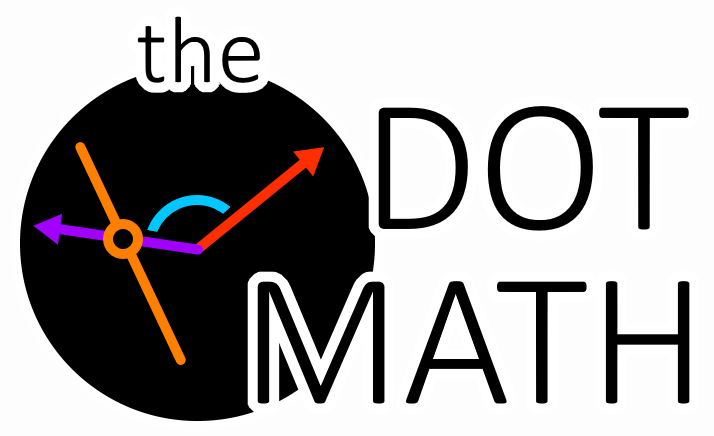
DotMath 1.1 Release

Описание функционала с примерами и картинками!





http://twitter.com/jkulvich

http://vk.com/jkulvich

http://vk.com/limide

Оглавление

001 Заголовок

002 Оглавление

003 Вступление

004 Описание методов и свойств класса «Dot»

004 Описание методов класса «DotMath»

005 Тесты производительности

006 Что из этого можно собрать?

006 Вывод двух векторов с углом в 90 градусов

007 Проверка пересечения двух отрезков

008 Проекция точки на отрезок

008 Проверка нахождения точки в фигуре

008 Ближайшая к данной точка на фигуре

Вступуление

Ну тут, типо, моё приветствие для вас!

Итак, размусоливать вступление не стану, скажу лишь что писал эту библиотеку ориентируясь в первую очередь на производительность, а во вторую, на удобство использования (в том числе на совместимость с прочими библиотеками).

Библиотека в первую очередь ориентирована для упрощения разработки 2D игр. По большей части – упрощает работу с векторами.

Надеюсь, вы уже знаете что представляет из себя данная библиотека (как минимум – из названия и того что я написал выше).

А если хотите подробнее, тогда вот:

«Математическая библиотека DotMath разрабатывалась как легковесное средство решения частых проблем и опускание рутинных операций с векторами. В первую очередь, библиотека ориентирована на разработку 2D игр. Во время разработки основной упор делался на производительность и удобство использования для программиста. Если вы хотите написать свою простенькую 2D игру с нуля и при этом не любите геометрию и не хотите особо заморачиваться по поводу чтения гигантских мануалов других библиотек – данное решение будет вам очень кстати.»

P.S. А вот дальше идёт размусоливание приветствия, можете не читать, там что-как-почему-зачем.

Скажу честно, геометрию я не очень-то и люблю… Но я устал каждый раз думать по поводу того как решить какую-либо задачу, именно поэтому я сел и… продумал на перёд.

В своё время я хотел написать 2D физический движок, на подобии Box2D, но меня остановило то что я не смог найти (или даже не пытался) формулу расчёта расстояние от точки до отрезка (позор мне, но с геометрией у меня всегда было хреново).

Надеюсь, данная библиотека позволит покрыть 90% геометрической работы в 2D играх. И вам не придётся лишний раз напрягать мозги по этому поводу и вы сможете больше сфокусироваться на качестве кода.

Описание методов и свойств класса «Dot»

|  |  |
| --- | --- |
| Именование метода / свойства | Описание |
| X | Принимает или возвращает координату X в декартовой системе |
| Y | Принимает или возвращает координату Y в декартовой системе |
| L | Принимает или возвращает расстояние от точки 0, 0 в полярной системе координат |
| P | Принимает или возвращает угол поворота точки в радианах относительно точки 0, 0 в полярной системе координат. |
| Dot GetUnitVector() | Вернёт точку являющуюся единичным вектором данной точки |
| Dot GetUnitVector(Dot A) | Вернёт точку являющуюся единичным вектором указанной точки |

Описание методов класса «DotMath»

|  |  |
| --- | --- |
| Именование метода | Описание |
| Float Distance(Dot A, Dot B) | Вернёт расстояние между двумя точками |
| Float TriangleArea(Dot A, Dot B, Dot C) | Вернёт площадь произвольного треугольника заданного вершинами |
| Float PolygonArea(Dot[] A) | Вернёт площадь произвольной выпуклой фигуры заданной точками |
| Bool IsDotInTriangle(Dot A, Dot B, Dot C, Dot D) | Вернёт значение, находится ли точка D внутри заданного вершинами ABC произвольного треугольника. |
| Bool IsDotInPolygon(Dot[] A, Dot B) | Вернёт значение, находится ли точка внутри произвольной выпуклой фигуры заданной точками |
| Dot StraightsIntersect(Dot A, Dot B, Dot C, Dot D) | Вернёт точку пересечения двух прямых, если такой точки не существует, то null |
| Dot[] Translate(Dot[] A, Dot B) | Вернёт массив точек смещённый +B вектор |
| Dot[] Rotate(Dot[] A, float B) | Вернёт массив точек повёрнутых на угол B относительно точки 0, 0 |
| Dot[] RotateFrom(Dot[] A, Dot B, float C) | Вернёт массив точек повёрнутых на угол C относительно точки B |
| Dot PolygonCenter(Dot[] A) | Вернёт центральную точку фигуры заданной вершинами массива A |
| Dot[] Scale(Dot[] A, float B) | Вернёт массив точек масштабированных относительно точки 0, 0 на коэффициент B |
| Dot[] ScaleFrom(Dot[] A, Dot B, float C) | Вернёт массив точек масштабированных относительно точки B на коэффициент C |
| Float StraightAngle(Dot A, Dot B, Dot C, Dot D) | Вернёт наименьший угол между прямыми заданными двумя точками |
| Dot DotStraightProjection(Dot A, Dot B, Dot C) | Вернёт точку, являющуюся проекцией точки C на прямую AB (Наиболее близкую к точке C точку лежащую на прямой AB) |
| Bool IsLinesIntersect(Dot A, Dot B, Dot C, Dot D) | Вернёт значение, указывающееся пересекаются ли два отрезка |
| Dot LineIntersect(Dot A, Dot B, Dot C, Dot D) | Вернёт точку являющуюся точкой пересечения двух отрезков, если отрезки не пересекаются, то null |
| Dot DotClosestSegment(Dot A, Dot B, Dot C) | Вернёт наиболее близкую к точке C точку лежащую на отрезке AB |
| Dot DotClosestSegments(Dot[] A, Dot B) | Вернёт наиболее близкую точку к точке C лежащую на ломаной кривой заданной массивом точек A |
| Dot DotClosestPolygon(Dot[] A, Dot B) | Вернёт наиболее близкую точку к точке C лежащую на замкнутой фигуре заданной массивом точек A |

Тесты производительности

Я провел некоторые тесты и скорость получилась вполне не плохая.

(проверял на AMD A8-7410 2.2GHz\*4)

Например, при игре в 60 FPS вы сможете вызвать проверку расстояния между точками более 27000 раз без потери в кадрах.

Однако, есть и довольно тяжёлые функции, как, например, проверка находится ли точка в произвольной фигуре. Для фигуры заданной 20-ю точками вы сможете вызвать данную проверку не более 319 раз без потери в кадрах.

Небольшой совет. Если вы планируете писать тяжёлые игры, советую ограничивать количество кадров 30-ю. Этого вполне достаточно для комфортной игры и при этом вы будете иметь в два раза больше вычислительного времени на кадр. Конечно, это действительно если вы не предусмотрели каких-либо хаков по кэшированию результатов вычислений или ещё чего-то подобного.

Ниже вы можете видеть почти полную таблицу скорости функций.



Что из этого можно собрать?

Ну что собрать – решает каждый сам.

- Можно без проблем определить, например, видит ли противник игрока за стеной.

- Рассчитать тень от объектов и карту света (нет, не мира, а от лампочки).

- Рассчитать угол отскока снаряда от поверхности.

Итак, на самом деле, есть некоторые моменты которые я бы хотел пояснить.

Библиотека в самом верху имеет константу:

#define SystemDrawing\_Compatibility

Если данная константа активна тогда класс Dot будет иметь полную и бесшовную совместимость с типом Point и PointF из пространства имён System.Drawing.

Если же вы данную библиотеку не используете, то советую эту константу закомментировать, в таком случае DotMath станет полностью автономной и самодостаточной библиотекой и не будет конфликтовать с другими библиотеками или требовать их наличие, но все преобразования из типа Dot в иные типы вам придётся взять на себя.

Эта штука поможет вам расширить функционал Point и не волноваться по поводу конвертации.

Например, вы можете сделать так:

Point P = new Dot(10f, 1.57f, true);

В данном случае вы задали координаты точке типа Point точкой из моей библиотеки в полярной системе координат (сейчас дойдём и до этого).

Итак, помимо восхитительных функций самой библиотеки DotMath, тип Dot тоже кое-что умеет.

Начнём с первого шага – инициализация переменной.

Инициализировать точку можно несколькими способами:

А) Dot D = new Dot();

Б) Dot D = new Dot(5f, 8f);

В) Dot D = new Dot(12f, 1.57f, true);

Г) Dot D = new Dot(dot);

Итак, что имеем. В случае А мы просто инициализируем новую точку с координатами (0, 0).

В случае Б мы инициализируем точку но в координатах (5, 8).

Далее интересней, в случае В мы инициализируем новую точку, но последний аргумент будучи установленным в true говорит что мы хотим задать координаты в полярной системе. В этом случае первый аргумент – полярное расстояние, второй – полярный градус.

Последний случай Г просто позволяет создать точную копию на другую точку (т.к. кидаться указателями не всегда удобно).

Отлично, мы создали нашу точку D, теперь какие манипуляции с ней мы можем делать?

Ну… всё то же самое что и с обычными векторами!

Т.е. мы можем умножать, делить, складывать и вычитать точки друг из друга (и не только точки, но и все совместимые типы данных, например Point и PointF).

Кроме того, мы можем делать всё те же операции не только с двумя точками, но и с точкой и числом.

А вот так, например, можно получить точку являющуюся обратной данной:

Dot AB = new Dot(1f, 0.5f);

Dot BA = -AB;

Итак, давайте попробуем что-нибудь на практике!

Для начала, обусловимся вот в чём, я буду представлять скрины результатов наших вычислений и код. Из кода я вырезаю всё что связано с выводом графики, графическое окно 400\*300px с выводом в 60+ FPS. Точкой M будем обозначать точку с текущими координатами мыши.

Итак, поехали!

Вывод двух векторов с углом в 90 градусов

Начнём с элементарного, создадим одну точку, потом возьмём копию созданной точки и повернём её на 90 градусов, ну и выведем оба вектора.



Dot A = new Dot(50f, 50f);

Dot B = new Dot(A);

B.P += 1.57f;

Вот что вышло! А теперь давайте проверим что угол верный (кстати, его мы задавали половиной от числа PI).

Вызовем метод DotMath.StraightsAngle(A, B);

Его ответ вы можете видеть на скрине рядом.

На самом деле, вы можете использовать класс Dot для лёгкой конвертации из декартовой системы координат в полярную и обратно. Просто делая так:

Dot A = new Dot();

A.X = 50f;

A.Y = 30f;

Console.Write(A.L.ToString(), A.P.ToString());

Аналогично вы можете и задать эти координаты в полярной системе и получить их в декартовой.

Проверка пересечения двух отрезков

В предыдущем случае мы вообще не использовали методы DotMath, а всего лишь обошлись стандартными свойствами класса Dot.

На этот раз давайте заюзаем функцию DotMath.LinesIntersect();



Dot A = new Dot(100f, 150f);

Dot B = new Dot(130f, 80f);

Dot C = new Dot(140f, 150f);

Dot X = DotMath.LinesIntersect(A, B, C, M);

Обратите внимание на последний случай, тут отрезки не пересекаются, а следственно, функция возвращает null, не забывайте проверять эту штуку.

Ещё хочу сделать небольшую пометку. Среди методов имеется в некотором роде похожая функция DotMath.IsLinesIntersect(); Она принимает абсолютно те же параметры, но лишь возвращает ответ пересекаются отрезки или нет. Если вам необходимо просто знать пересекаются отрезки или нет, в таком случае будет гораздо оптимальнее использовать именно последний метод, т.к. там иной алгоритм и он примерно в полтора раза быстрее делает данную проверку.

Проекция точки на отрезок

Предлагаю решить при помощи моей библиотеки ещё одну интересную задачу – нахождение проекции точки на прямую (отсетова можно получить и расстояние от точки до прямой, хотя я уже позаботился о вас и такая функция имеется).

Dot A = new Dot(100f, 150f);

Dot B = new Dot(130f, 80f);

Dot X = DotMath.DotStraightProjection(A, B, M);

В данном случае проверку на null делать не стоит, ибо обратите внимание на последний вариант, мы же находим проекцию на прямую, а не отрезок, а прямая бесконечна… прям как мои пары по матану.

Проверка нахождения точки в фигуре

Это была одна из первых задач которую могла решать моя библиотека. Штука весьма полезная для различных «хит боксов» в играх и прочего.

Что ж, как всегда, опять всё делаем ~~одной правой~~ одним методом!

Dot[] DS = new Dot[] { new Dot(40, 40), new Dot(120, 40), new Dot(80, 80), new Dot(70, 70), new Dot(50, 90), new Dot(30, 20), };

bool IN = DotMath.IsDotInPolygon(DS, M);

Здесь мы создали массив точек, а затем просто проверили его на пересечение с курсором и получили результат в виде булевого значения (true/false).

Ближайшая к данной точка на фигуре

Теперь предлагаю найти ближайшую к данной точку, которая при том будет одновременно лежать и на замкнутом полигоне. Такое можно применить для, например, определения зоны по которой может передвигаться некий объект.

Dot[] DS = new Dot[] { new Dot(40, 40), new Dot(120, 40), new Dot(80, 80), new Dot(70, 70), new Dot(50, 90), new Dot(30, 20), };

Dot X = DotMath.DotClosestPolygon(DS, M);

Возьмём всё тот же массив точек для описания фигуры и получим наиболее близкую точку к нашему курсору на фигуре.

Что ж, надеюсь, вы оцените.

С радостью приму ваши пожелания, предложения и замечания об ошибках: