Рендеринг псевдотрёхмерного пространства методом бросания лучей

Шорников Александр Евгеньевич, группа 05230

Бурятский государственный университет
Институт математики и информатики
Кафедра прикладной математики

Научный руководитель — асс. **Брагин Александр Фёдорович**

Улан-Удэ
2016г.

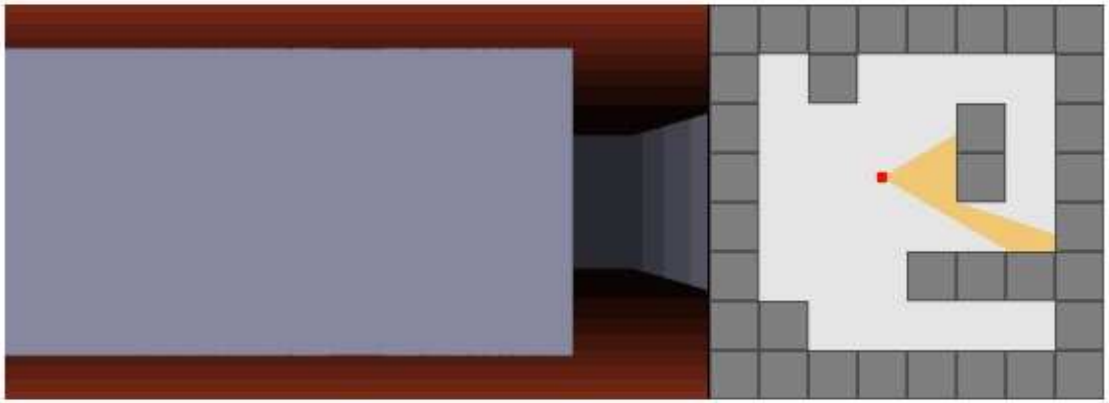
Введение

Цель данной курсовой работы:
- проанализировать и обработать теоретические и экспериментальные данные по теме «Метод бросания луча»;
- анализ собранной информации;
- иллюстрация методов;
- разработка программы, реализующая данное семейство методов.

Объектом исследования данной курсовой работы является задача построения псевдотрёхмерной картинки.

Технология рейкастинга

Метод бросания лучей(англ. Raycasting, "Рейкастинг") - это технология получения изображения по модели с помощью компьютерной программы, позволяющая создавать 3D перспективу в 2D картах. По сути, это метод преобразования ограниченной формы данных (очень простая карта этажа) в трёхмерную проекцию с помощью трассировки лучей из точки обзора в объём обзора.

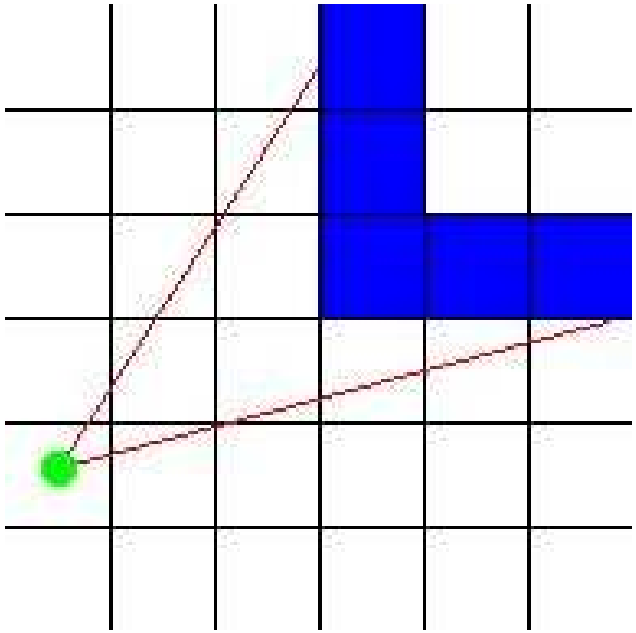


Описание метода

Основная идея рейкастинга состоит в следующем: карта представляет собой 2D-решетку с квадратными ячейками (двухмерный массив), где значением каждой ячейки может быть равно 0, что означает отсутствие стены, либо положительное число, означающее стену определенного цвета или текстуры.
Каждому значению x на экране (для каждой вертикальной линии на экране) соответствует луч, который исходит из местонахождения игрока и направление которого зависит от двух критериев: направление взгляда игрока и координата x на экране. Затем данный луч начинается двигаться вперед по 2D карте до тех пор, пока не упрется в ячейку карты, которая является стеной. Если он пересечётся со стеной, то будет рассчитываться расстояние от этой точки соприкосновения со стеной до игрока, которое поможет определить, насколько высоко стену нужно будет переместить на экране: чем дальше распложена стена, тем меньше будет ее изображение на экране, и наоборот.

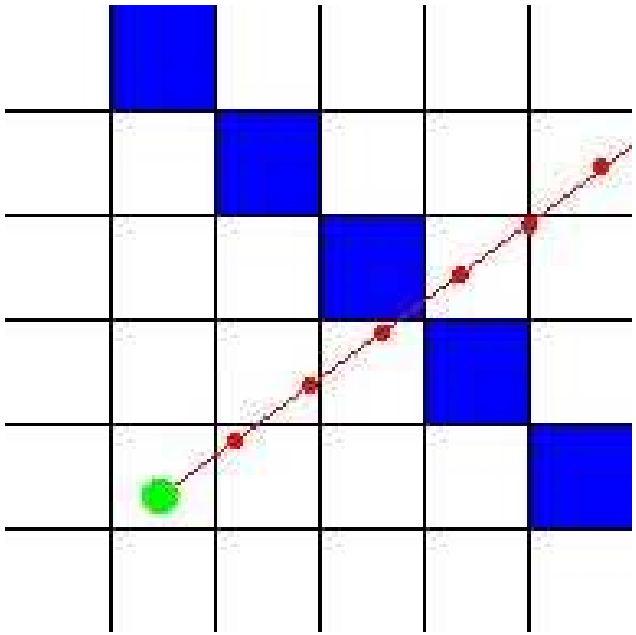
Описание метода

На рисунке в ракурсе «сверху вниз» представлены два луча (выделены красным), которые исходят от игрока зеленя точка) и упираются в синюю стену.



Описание метода

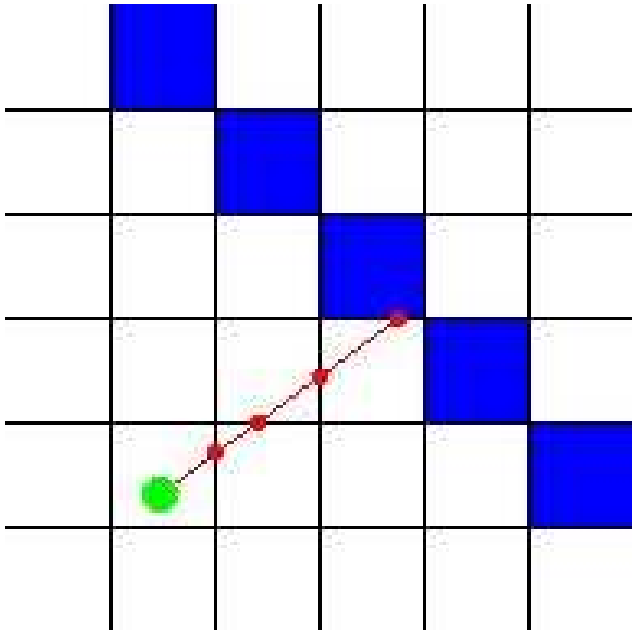
Чтобы обнаружить первую стену, которую луч встречает на своем пути, необходимо, чтобы он исходил из точки местоположения игрока, а затем нужно все время проверять, не находится ли луч внутри стены. Если он оказывается внутри стены (упирается в неё), цикл можно завершить, рассчитать расстояние и нарисовать стену правильной высоты. Если же луч не упирается в стену, необходимо продолжать вести его: добавьте определённую величину к его положению, в направлении направления данного луча, и проверьте, не находится ли луч в новом положении внутри стены. Проделывать данные действия необходимо до тех пор, пока луч не коснется стены.



Описание метода

Человек может сразу же увидеть, касается ли луч стены, но невозможно сразу же рассчитать, какой именно ячейки луч касается, используя всего одну формулу, поскольку компьютер может осуществить проверку ограниченного количества положений луча. Многие рейкастовые движки добавляют постоянную величину к лучу на каждом этапе, но в этом случае существует вероятность того, что луч может "промахнуться"и не коснуться стены.
Но существует более подходящая методика, предполагающая выполнение лишь нескольких вычислений и позволяющая определить каждую из стен. Идея состоит в следующем: проверять факт наличие луча на каждой из сторон стены. При ширине каждой ячейки, равной 1, каждая из сторон стены будет целым числом, а места между стенами будут равны некоему числу с цифрами после запятой. В этом случае размер шага не является постоянной величиной, он зависит от расстояния до следующей стороны ячейки.

Описание метода



Как видно на рисунке, луч касается стены именно там, где нам это необходимо. В способе используется алгоритм, основанный на цифровом дифференциальном анализе. Цифровой Дифференциальный Анализ - скоростной алгоритм, обычно применяемый при использовании квадратной решётки, позволяющий определить, какие ячейки задевает луч (например, чтобы нарисовать линию на экране, состоящем из решётки квадратных пикселей). Таким образом, мы также можем использовать этот метод, чтобы определить, какие ячейки решётки на нашем экране оказываются задеты лучом, и приостановить алгоритм, как только луч коснется ячейки, являющейся стеной.

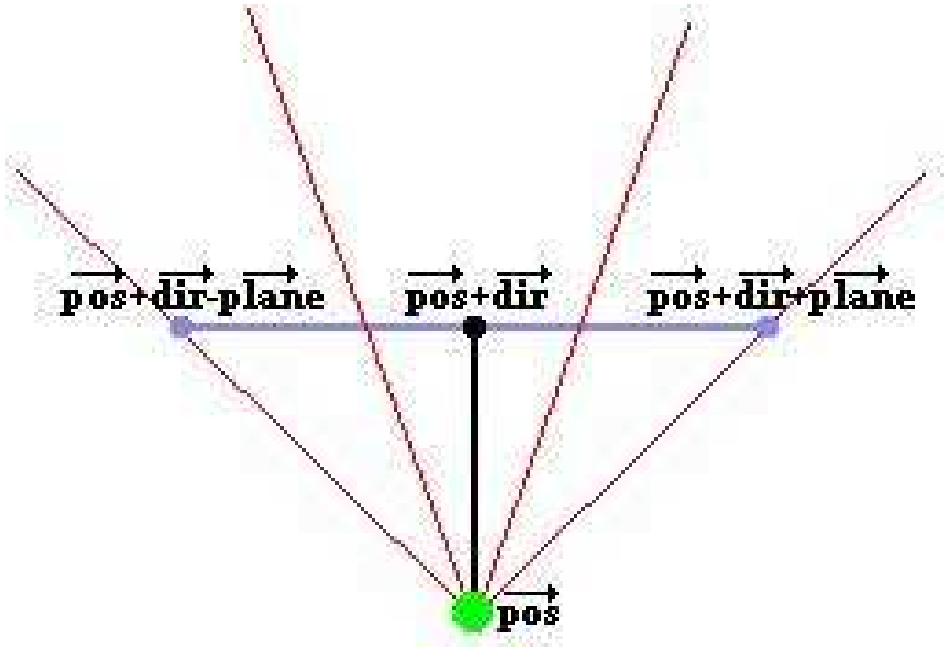
Описание метода

Рейкастинг работает с векторами для задания угла обзора и камеры: положение игрока всегда является вектором (с координатами x и y), но теперь мы можем также определить направление вектора: его направление определяется с помощью двух величин (координат x и y направления). Вектор направления можно визуализировать следующим образом: если вы нарисуете линию в направлении взгляда игрока, проходящую через точку нахождения игрока, то каждая точка данной линии будет являться суммой показателей положения игрока и будет кратна направлению вектора (показателям направления вектора). Длина вектора направления не имеет особого значения, важно лишь его направление. Умножение показателей x и y на одну и ту же величину изменяет длину, но сохраняет направление вектора.

Данный векторный метод также требует наличие дополнительного вектора, представляющего собой вектор плоскости камеры. В настоящем 3D движке также имеется плоскость камеры, и там она является настоящей 3D плоскостью, которую должны представлять два вектора (u и v). Рейкастинг рассчитан на 2D карты, потому в этом случае плоскость камеры не является плоскостью, это скорее линия, представленная одним вектором. Плоскость камеры всегда должна быть перпендикулярна вектору направления. Плоскость камеры представляет собой плоскость на компьютерном экране, в то время как вектор направления расположен перпендикулярно по отношению к этой плоскости и направлен внутрь экрана. Положение игрока представлено одной точкой, расположенной перед плоскостью камеры. Определённый луч определённых x-координат на экране является, в таком случае, лучом, который начинается в точке нахождения игрока и проходит через это положение или через плоскость камеры.

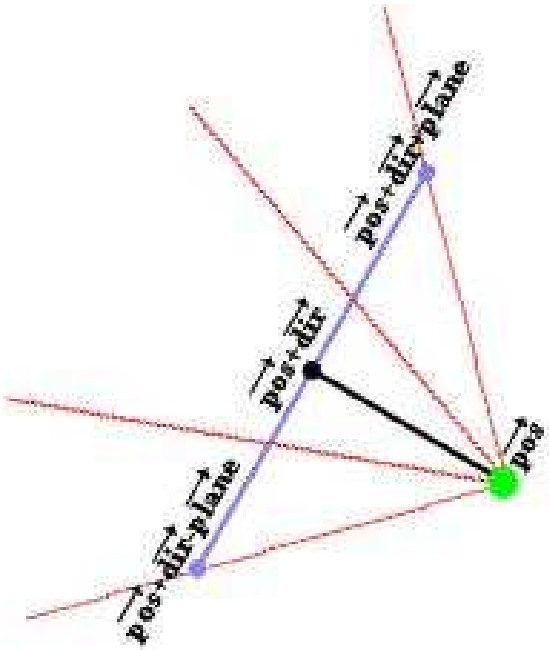
Описание метода

На рисунке представлена такая 2D камера. Зеленая точка - это положение (вектор $\overrightarrow{pos}$). Чёрная линия, оканчивающаяся чёрной точкой, представляет вектор направления (вектор $\overrightarrow{dir}$). Таким образом, положение чёрной точки - это вектор $\overrightarrow{pos} + \overrightarrow{dir}$. Синяя линия представляет полную плоскость камеры. Вектор, проходящий от чёрной точки к правой синей точке представляет вектор $\overrightarrow{plane}$. Таким образом, положение правой синей точки - $\overrightarrow{pos} + \overrightarrow{dir} + \overrightarrow{plane}$, а положение левой синей точки - $\overrightarrow{pos} + \overrightarrow{dir} - \overrightarrow{plane}$.



Описание метода

При вращении игрока камера также должна вращаться(рис. 2.9), следовательно, и вектор направления, и вектор плоскости камеры также должны поворачиваться вместе с ними. Далее все остальные лучи будут вращаться автоматически.



Чтобы повернуть вектор, необходимо рассчитать его по следующему шаблону вращения:

$$\begin{vmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\beta) & cos(\beta) \end{vmatrix}$$

Реализация нетекстурированного движка

Для реализации данного движка я использовал язык C++ и графические библиотеки SDL. Работоспособность была проверена в среде разработки *Code* :: *Blocks* 16.01 с компилятором *gcc* под операционными системами *Linux Mint* 18 и *Windows* 7.