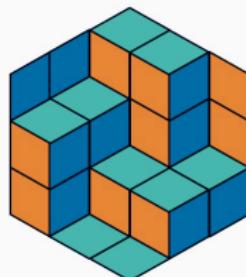


Я просто хотел писать игры

Неожиданная математика в разработке игр

Никита Лисица

2024



Факультет
математики
и компьютерных
наук
СПбГУ

Обо мне

- Работаю в Яндекс Картах

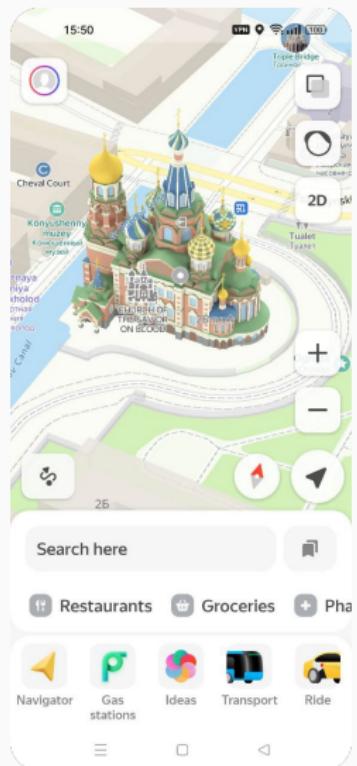
Обо мне

- Работаю в Яндекс Картах
- Веду два спецкурса о компьютерной графике на факультете МКН в СПбГУ

Обо мне

- Работаю в Яндекс Картах
- Веду два спецкурса о компьютерной графике на факультете МКН в СПбГУ
- В свободное время делаю инди-игры на собственном движке

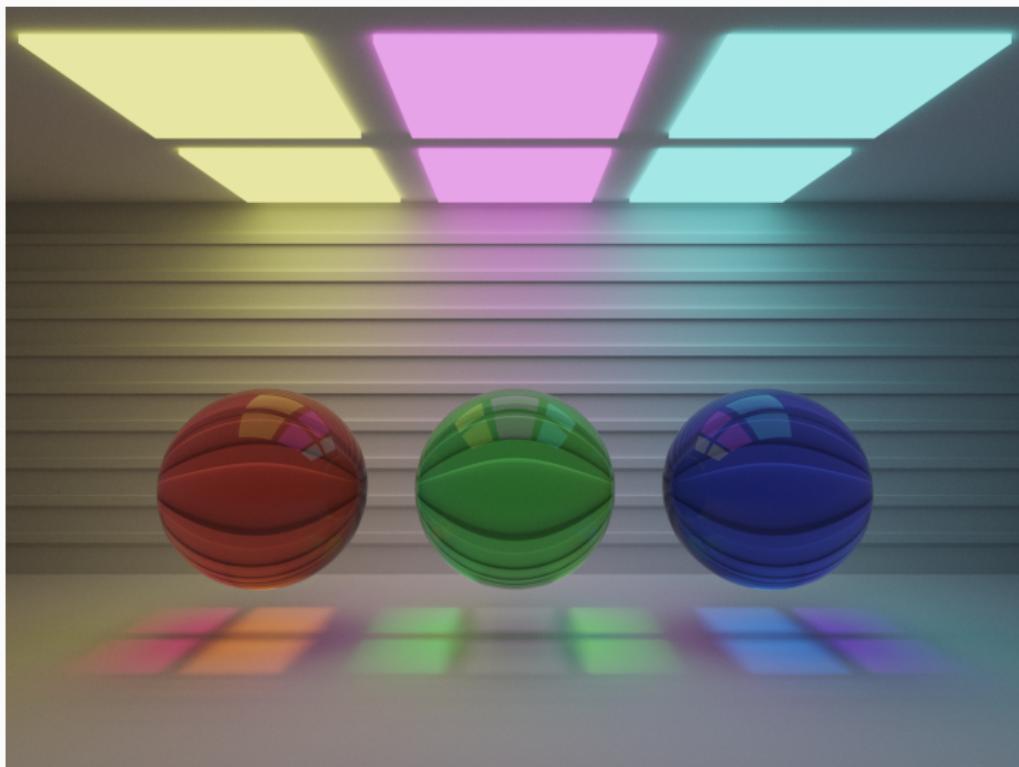
Яндекс.Карты



Курс о real-time графике



Курс о трассировке лучей



Инди-игры



Инди-игры



Из чего состоят игры?

Разработка игр – очень сложный, мультидисциплинарный вид деятельности:

Из чего состоят игры?

Разработка игр – очень сложный, мультидисциплинарный вид деятельности:

- Программирование логики

Из чего состоят игры?

Разработка игр – очень сложный, мультидисциплинарный вид деятельности:

- Программирование логики
- Графический движок

Из чего состоят игры?

Разработка игр – очень сложный, мультидисциплинарный вид деятельности:

- Программирование логики
- Графический движок
- Физика

Из чего состоят игры?

Разработка игр – очень сложный, мультидисциплинарный вид деятельности:

- Программирование логики
- Графический движок
- Физика
- Аудио

Из чего состоят игры?

Разработка игр – очень сложный, мультидисциплинарный вид деятельности:

- Программирование логики
- Графический движок
- Физика
- Аудио
- Создание контента (моделей, текстур, анимаций, эффектов)

Из чего состоят игры?

Разработка игр – очень сложный, мультидисциплинарный вид деятельности:

- Программирование логики
- Графический движок
- Физика
- Аудио
- Создание контента (моделей, текстур, анимаций, эффектов)
- Гейм-дизайн

Из чего состоят игры?

Разработка игр – очень сложный, мультидисциплинарный вид деятельности:

- Программирование логики
- Графический движок
- Физика
- Аудио
- Создание контента (моделей, текстур, анимаций, эффектов)
- Гейм-дизайн
- ...и многое другое

Из чего состоят игры?

Разработка игр – очень сложный, мультидисциплинарный вид деятельности:

- Программирование логики
- Графический движок
- Физика
- Аудио
- Создание контента (моделей, текстур, анимаций, эффектов)
- Гейм-дизайн
- ...и многое другое

Где здесь нужна математика?

Из чего состоят игры?

Разработка игр – очень сложный, мультидисциплинарный вид деятельности:

- Программирование логики
- Графический движок
- Физика
- Аудио
- Создание контента (моделей, текстур, анимаций, эффектов)
- Гейм-дизайн
- ...и многое другое

Где здесь нужна математика? Практически везде!

Перемещение персонажа



Перемещение персонажа

- Игрок нажал клавишу влево/вправо/вверх/вниз – что нужно сделать в коде игры?

Перемещение персонажа

- Игрок нажал клавишу влево/вправо/вверх/вниз – что нужно сделать в коде игры?
- Сдвинуть положение персонажа на какой-то вектор!

Перемещение персонажа

- Игрок нажал клавишу влево/вправо/вверх/вниз – что нужно сделать в коде игры?
- Сдвинуть положение персонажа на какой-то вектор!
- $\text{position}_1 = \text{position}_0 + \text{move}$

Перемещение персонажа

- Игрок нажал клавишу влево/вправо/вверх/вниз – что нужно сделать в коде игры?
- Сдвинуть положение персонажа на какой-то вектор!
- $\text{position}_1 = \text{position}_0 + \text{move}$
- Например, для перемещения влево $\text{move} = (-1, 0)$

Перемещение персонажа

- Игрок нажал клавишу влево/вправо/вверх/вниз – что нужно сделать в коде игры?
- Сдвинуть положение персонажа на какой-то вектор!
- $\text{position}_1 = \text{position}_0 + \text{move}$
- Например, для перемещения влево $\text{move} = (-1, 0)$
- Персонаж резко переместится в другое место – это некрасиво!

Плавное перемещение персонажа

- Сделаем перемещение анимированным!

Плавное перемещение персонажа

- Сделаем перемещение анимированным!
- Анимация – зависимость чего-либо от времени, часто в виде явно заданной функции

Плавное перемещение персонажа

- Сделаем перемещение анимированным!
- Анимация – зависимость чего-либо от времени, часто в виде явно заданной функции
- $\text{position}(t) = \text{position}_0 + \delta(t) \cdot \text{move}$, где

$$\delta : [0, 1] \rightarrow [0, 1]$$

$$\delta(0) = 0$$

$$\delta(1) = 1$$

Плавное перемещение персонажа

- Сделаем перемещение анимированным!
- Анимация – зависимость чего-либо от времени, часто в виде явно заданной функции
- $\text{position}(t) = \text{position}_0 + \delta(t) \cdot \text{move}$, где

$$\delta : [0, 1] \rightarrow [0, 1]$$

$$\delta(0) = 0$$

$$\delta(1) = 1$$

- Такие функции δ обычно называют *easing* функциями

Easing функции

- $\delta(t) = t$ – движение с постоянной скоростью

Easing функции

- $\delta(t) = t$ – движение с постоянной скоростью
- $\delta(t) = t^2$ – движение с постепенным ускорением, начинающееся плавно и резко заканчивающееся

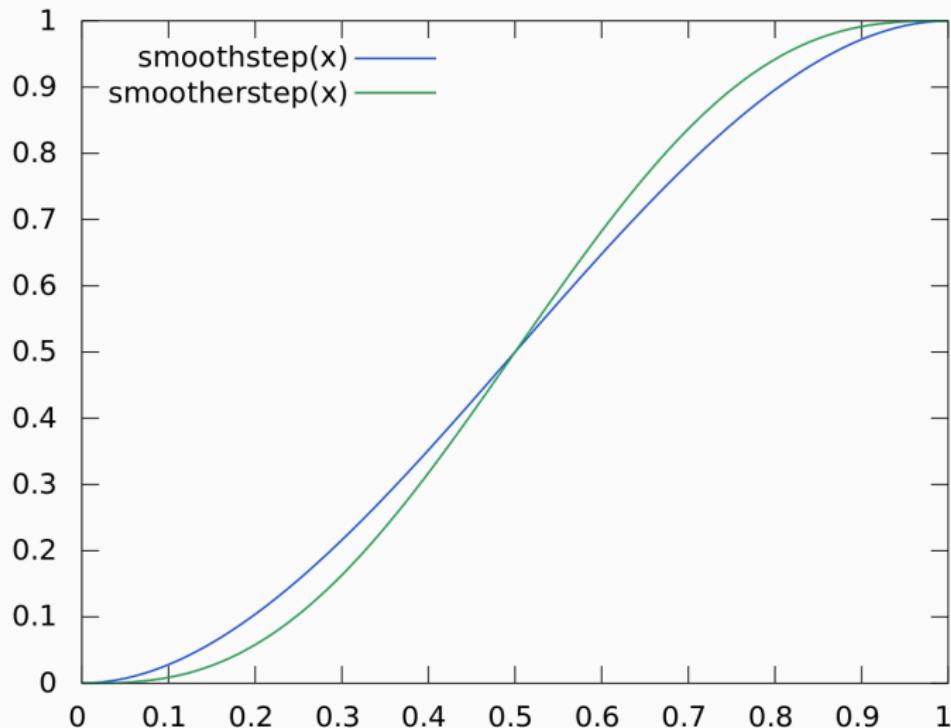
Easing функции

- $\delta(t) = t$ – движение с постоянной скоростью
- $\delta(t) = t^2$ – движение с постепенным ускорением, начинающееся плавно и резко заканчивающееся
- $\delta(t) = 1 - (1 - t)^2$ – движение с постепенным замедлением, начинающееся резко и плавно заканчивающееся

Easing функции

- $\delta(t) = t$ – движение с постоянной скоростью
- $\delta(t) = t^2$ – движение с постепенным ускорением, начинающееся плавно и резко заканчивающееся
- $\delta(t) = 1 - (1 - t)^2$ – движение с постепенным замедлением, начинающееся резко и плавно заканчивающееся
- $\delta(t) = 3t^2 - 2t^3$ – движение, начинающееся и заканчивающееся плавно (*smoothstep*)

Smoothstep



Smoothstep

- Smoothstep – $s(t) = 3t^2 - 2t^3$ – очень важная функция!

Smoothstep

- Smoothstep – $s(t) = 3t^2 - 2t^3$ – очень важная функция!
- Настолько важная, что она встроена во все языки шейдеров, и про неё есть отдельная статья на википедии

Smoothstep

- Smoothstep – $s(t) = 3t^2 - 2t^3$ – очень важная функция!
- Настолько важная, что она встроена во все языки шейдеров, и про неё есть отдельная статья на википедии
- Это многочлен минимальной степени, удовлетворяющий условиям

$$s(0) = 0 \quad s(1) = 1 \quad s'(0) = 0 \quad s'(1) = 0$$

Smoothstep

- Smoothstep – $s(t) = 3t^2 - 2t^3$ – очень важная функция!
- Настолько важная, что она встроена во все языки шейдеров, и про неё есть отдельная статья на википедии
- Это многочлен минимальной степени, удовлетворяющий условиям

$$s(0) = 0 \quad s(1) = 1 \quad s'(0) = 0 \quad s'(1) = 0$$

- Эта функция часто используется для анимаций и сглаживания

Генерация мира



Генерация мира

- Многие современные игры используют *процедурную генерацию контента (PCG)*, в том числе для генерации всего игрового мира

Генерация мира

- Многие современные игры используют *процедурную генерацию контента (PCG)*, в том числе для генерации всего игрового мира
- Есть много подходов к генерации мира, но грубо их можно разделить на два: *глобальный* и *локальный*

Генерация мира

- Многие современные игры используют *процедурную генерацию контента (PCG)*, в том числе для генерации всего игрового мира
- Есть много подходов к генерации мира, но грубо их можно разделить на два: *глобальный и локальный*
- В глобальном подходе генерируется сразу весь мир – например, целый остров с горами, реками, и поселениями

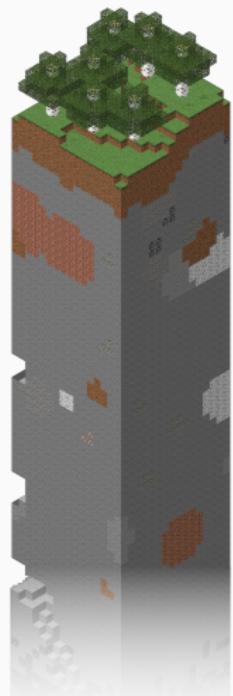
Генерация мира

- Многие современные игры используют *процедурную генерацию контента (PCG)*, в том числе для генерации всего игрового мира
- Есть много подходов к генерации мира, но грубо их можно разделить на два: *глобальный и локальный*
- В глобальном подходе генерируется сразу весь мир – например, целый остров с горами, реками, и поселениями
- В локальном подходе мир генерируется постепенно, и различные его участки могут генерироваться независимо друг от друга

Генерация мира

- Многие современные игры используют *процедурную генерацию контента (PCG)*, в том числе для генерации всего игрового мира
- Есть много подходов к генерации мира, но грубо их можно разделить на два: *глобальный и локальный*
- В глобальном подходе генерируется сразу весь мир – например, целый остров с горами, реками, и поселениями
- В локальном подходе мир генерируется постепенно, и различные его участки могут генерироваться независимо друг от друга
- Для игр с очень большим миром предпочтительнее *локальный подход*

Генерация мира



Генерация мира

- Итак, мы хотим сгенерировать небольшой кусочек нашего мира

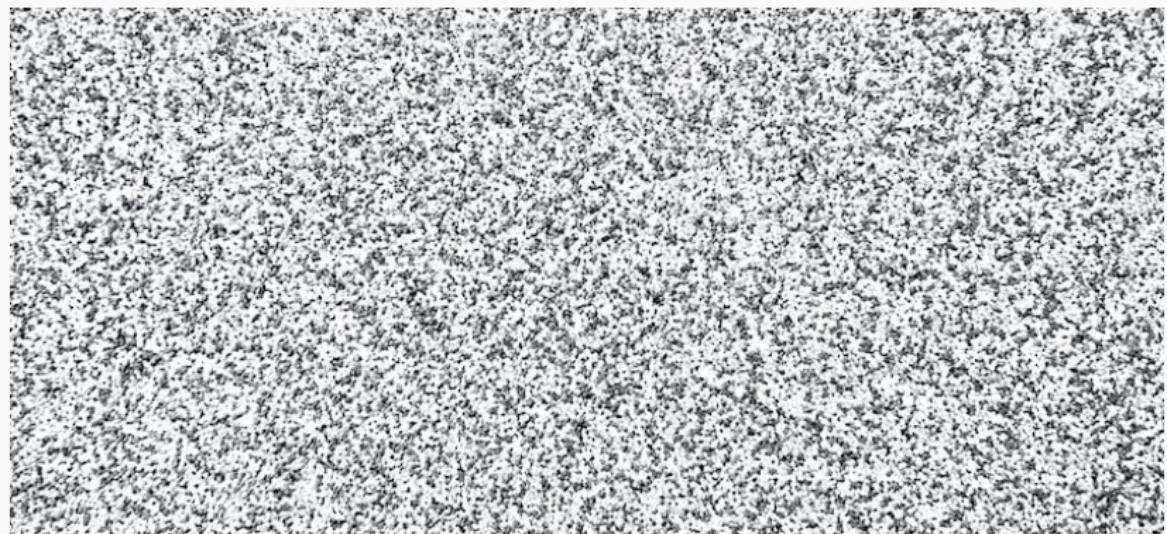
Генерация мира

- Итак, мы хотим сгенерировать небольшой кусочек нашего мира
- Для простоты будем считать, что мы хотим только сгенерировать высоту ландшафта в каждой точке, т.е. карту *высот*

Генерация мира

- Итак, мы хотим сгенерировать небольшой кусочек нашего мира
- Для простоты будем считать, что мы хотим только сгенерировать высоту ландшафта в каждой точке, т.е. карту *высот*
- Для этого нам нужен шум

Шум



Шум

- В игровой индустрии *шумом* называют любую функцию, выглядящую случайной

Шум

- В игровой индустрии *шумом* называют любую функцию, выглядящую случайной
- Часто, эта функция задана не для непрерывного аргумента, а на дискретной сетке

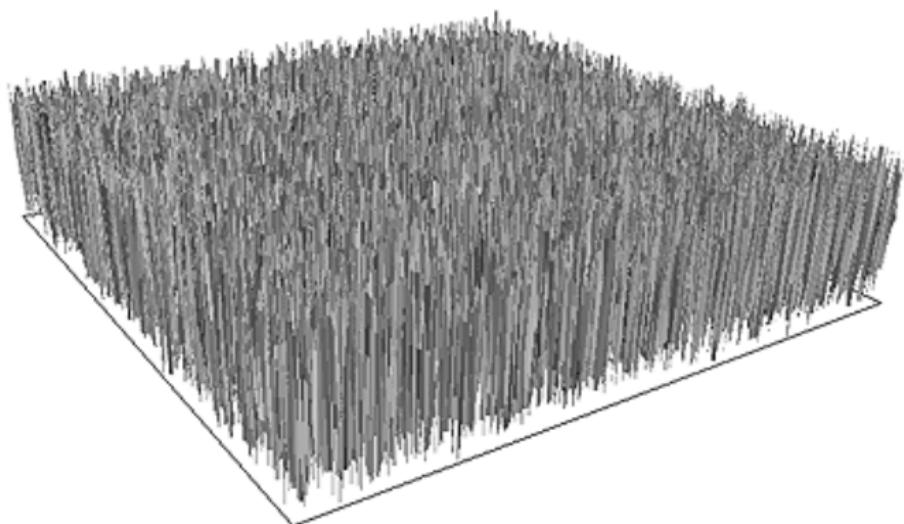
Шум

- В игровой индустрии шумом называют любую функцию, выглядящую случайной
- Часто, эта функция задана не для непрерывного аргумента, а на дискретной сетке
- *Белый шум* – шум, значения которого в разных ячейках сетки никак не зависят друг от друга

Шум

- В игровой индустрии шумом называют любую функцию, выглядящую случайной
- Часто, эта функция задана не для непрерывного аргумента, а на дискретной сетке
- *Белый шум* – шум, значения которого в разных ячейках сетки никак не зависят друг от друга
- Такой шум плохо подходит для карты высот!

Карта высот на основе белого шума



Гладкий шум

- Нам нужен какой-нибудь *непрерывный шум* – плавно меняющаяся, но всё ещё случайная функция

Гладкий шум

- Нам нужен какой-нибудь *непрерывный шум* – плавно меняющаяся, но всё ещё случайная функция
- Один из самых распространённых алгоритмов для этого называется *шумом Перлина* (по фамилии его изобретателя, Кена Перлина)

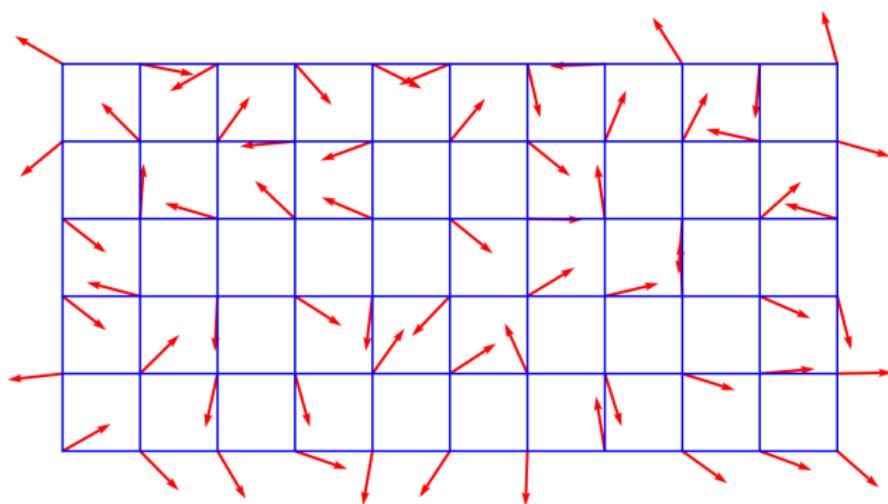
Гладкий шум

- Этот алгоритм генерирует случайные единичные векторы в вершинах квадратной сетки

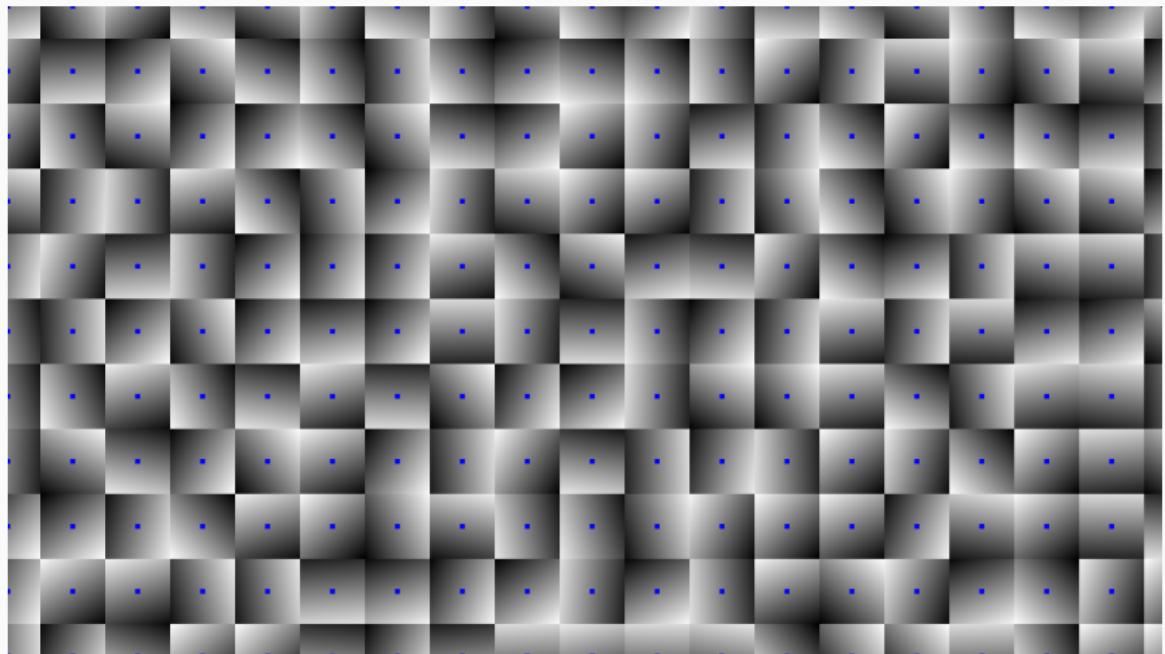
Гладкий шум

- Этот алгоритм генерирует случайные единичные векторы в вершинах квадратной сетки
- Для получения значения шума в некоторой точке, алгоритм вычисляет скалярные произведения этих векторов с векторами из точки в вершины сетки, и затем интерполирует их

Шум Перлина: векторы



Шум Перлина: скалярные произведения



Шум Перлина

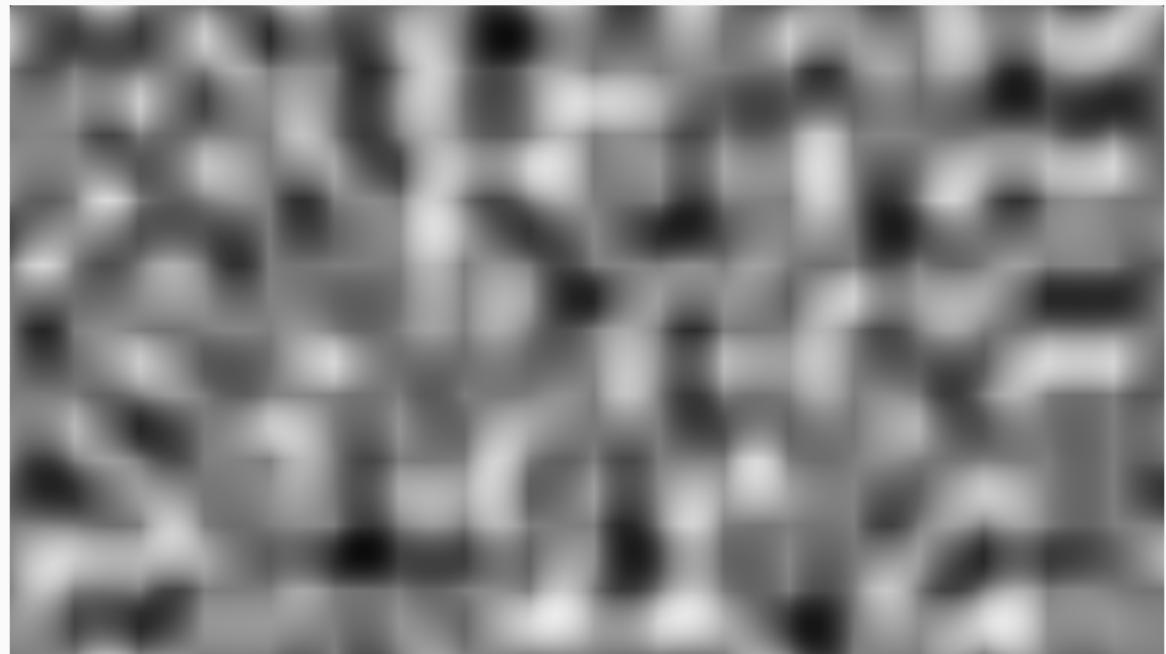
- Получив четыре скалярных произведения d_1, \dots, d_4 из ближайших вершин сетки, алгоритм интерполирует между ними, используя координаты в ячейке сетки как коэффициенты интерполяции

Шум Перлина

- Получив четыре скалярных произведения d_1, \dots, d_4 из ближайших вершин сетки, алгоритм интерполирует между ними, используя координаты в ячейке сетки как коэффициенты интерполяции
- Если точка в квадрате (ячейке сетки) имеет координаты $x, y \in [0, 1]$, то значение шума алгоритм вычисляет с помощью билинейной интерполяции

$$x \cdot y \cdot d_1 + (1 - x) \cdot y \cdot d_2 + x \cdot (1 - y) \cdot d_3 + (1 - x) \cdot (1 - y) \cdot d_4$$

Шум Перлина: билинейная интерполяция



Шум Перлина

- У полученного шума явно видна исходная сетка, и он мало пригоден для использования

Шум Перлина

- У полученного шума явно видна исходная сетка, и он мало пригоден для использования
- На помощь приходит уже известная нам функция *smoothstep*: нужно просто применить её к коэффициентам интерполяции!

Шум Перлина

- У полученного шума явно видна исходная сетка, и он мало пригоден для использования
- На помощь приходит уже известная нам функция *smoothstep*: нужно просто применить её к коэффициентам интерполяции!

$$x \leftarrow 3x^2 - 2x^3$$

$$y \leftarrow 3y^2 - 2y^3$$

Шум Перлина



Фрактальный шум

- Такой шум лучше, чем белый шум, но он всё ещё слишком плавный

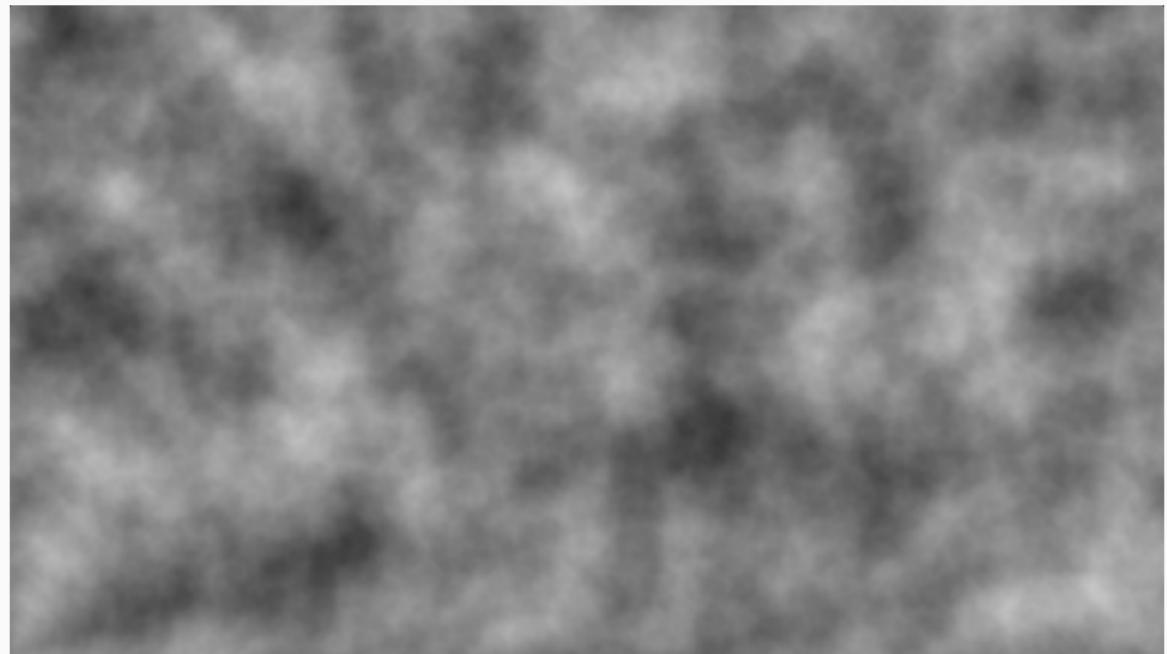
Фрактальный шум

- Такой шум лучше, чем белый шум, но он всё ещё слишком плавный
- Обычно для генерации ландшафта смешивают несколько слоёв такого шума с разными размерами сетки

Фрактальный шум

- Такой шум лучше, чем белый шум, но он всё ещё слишком плавный
- Обычно для генерации ландшафта смешивают несколько слоёв такого шума с разными размерами сетки
- Получающийся шум называют *фрактальным шумом*

Фрактальный шум



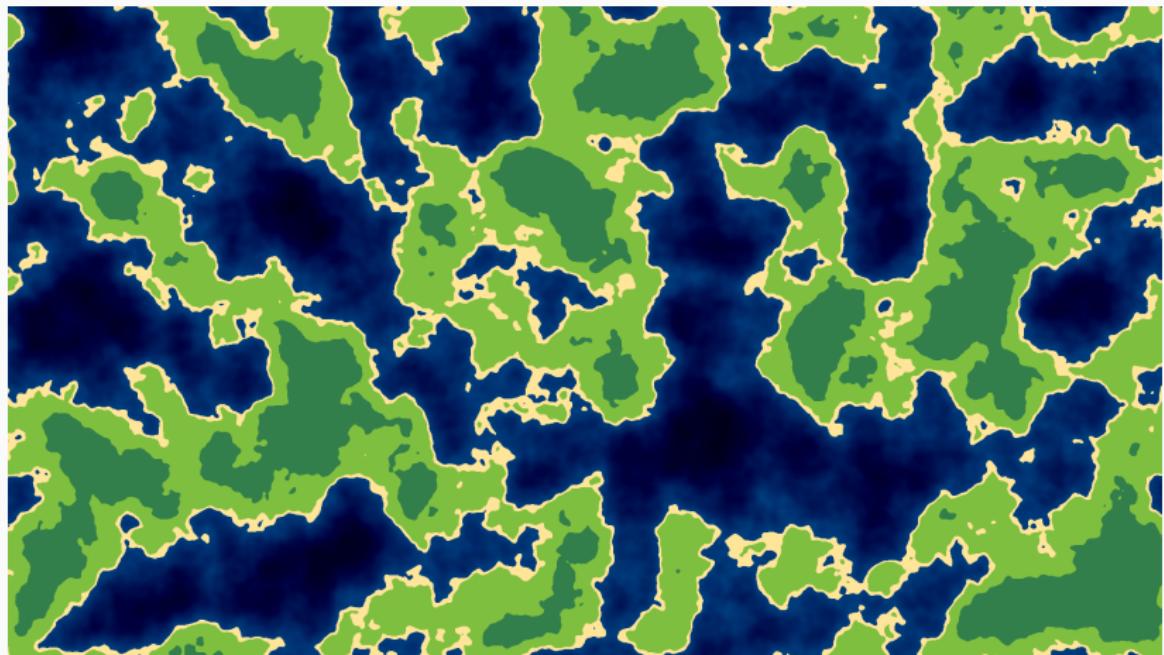
Фрактальный шум

- Значение шума можно взять в качестве высоты ландшафта в точке

Фрактальный шум

- Значение шума можно взять в качестве высоты ландшафта в точке
- На основе шума, или комбинации нескольких слоёв шума, можно построить карты температуры, количества осадков, и т.п.

Карта на основе фрактального шума



Рассчёт освещения

- Сейчас наша карта выглядит плоской

Рассчёт освещения

- Сейчас наша карта выглядит плоской
- Чтобы передать её трёхмерность, нужно применить к ней *освещение*

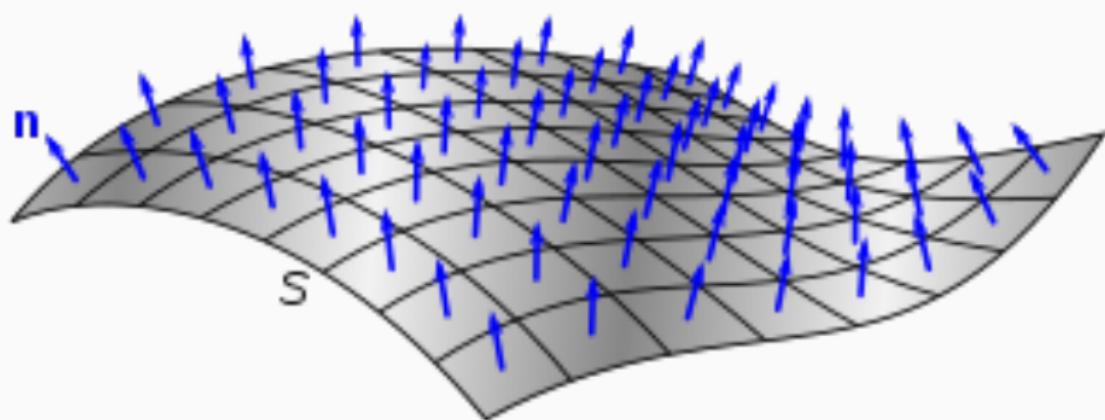
Рассчёт освещения

- Сейчас наша карта выглядит плоской
- Чтобы передать её трёхмерность, нужно применить к ней *освещение*
- Чтобы вычислить освещённость поверхности, нужно знать, насколько сильно она повернута в сторону света

Рассчёт освещения

- Сейчас наша карта выглядит плоской
- Чтобы передать её трёхмерность, нужно применить к ней *освещение*
- Чтобы вычислить освещённость поверхности, нужно знать, насколько сильно она повёрнута в сторону света
- За это отвечает вектор *нормали* n – перпендикуляр к поверхности объекта

Нормаль



Вычисление нормали

- Для поверхности, заданной своей функцией высоты от точки $z(x, y)$ вектор нормали вычисляется на основе частных производных

Вычисление нормали

- Для поверхности, заданной своей функцией высоты от точки $z(x, y)$ вектор нормали вычисляется на основе частных производных

$$n = \frac{\left(-\frac{\partial z}{\partial x}, -\frac{\partial z}{\partial y}, 1 \right)}{\sqrt{1 + \left(\frac{\partial z}{\partial x} \right)^2 + \left(\frac{\partial z}{\partial y} \right)^2}}$$

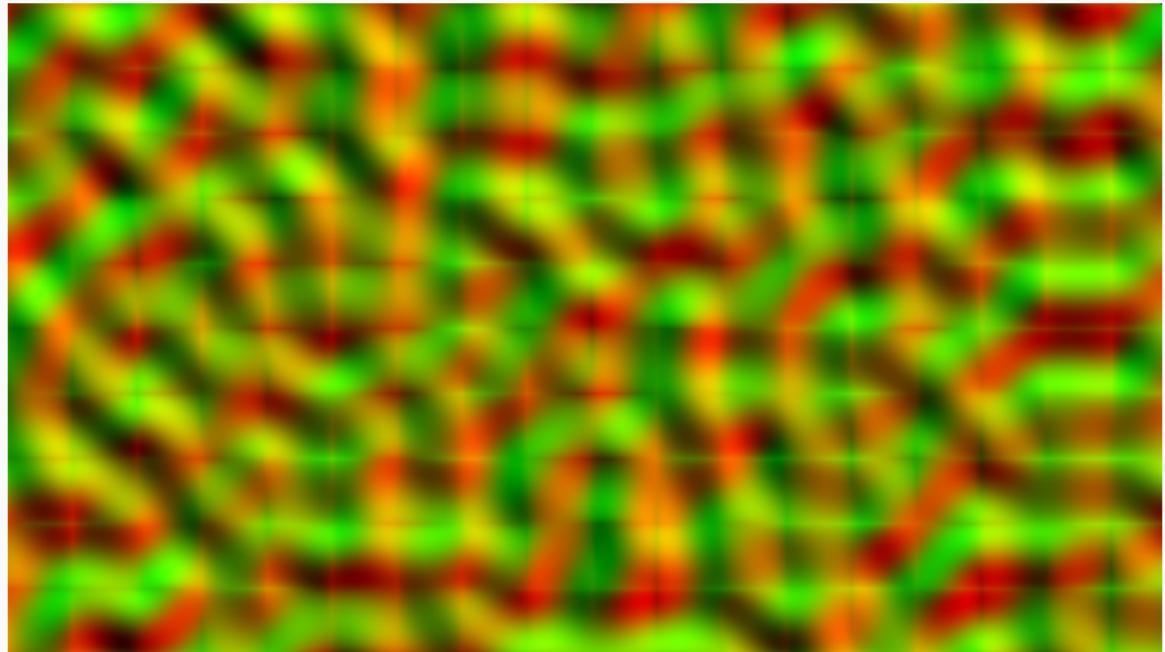
Вычисление нормали

- Для поверхности, заданной своей функцией высоты от точки $z(x, y)$ вектор нормали вычисляется на основе частных производных

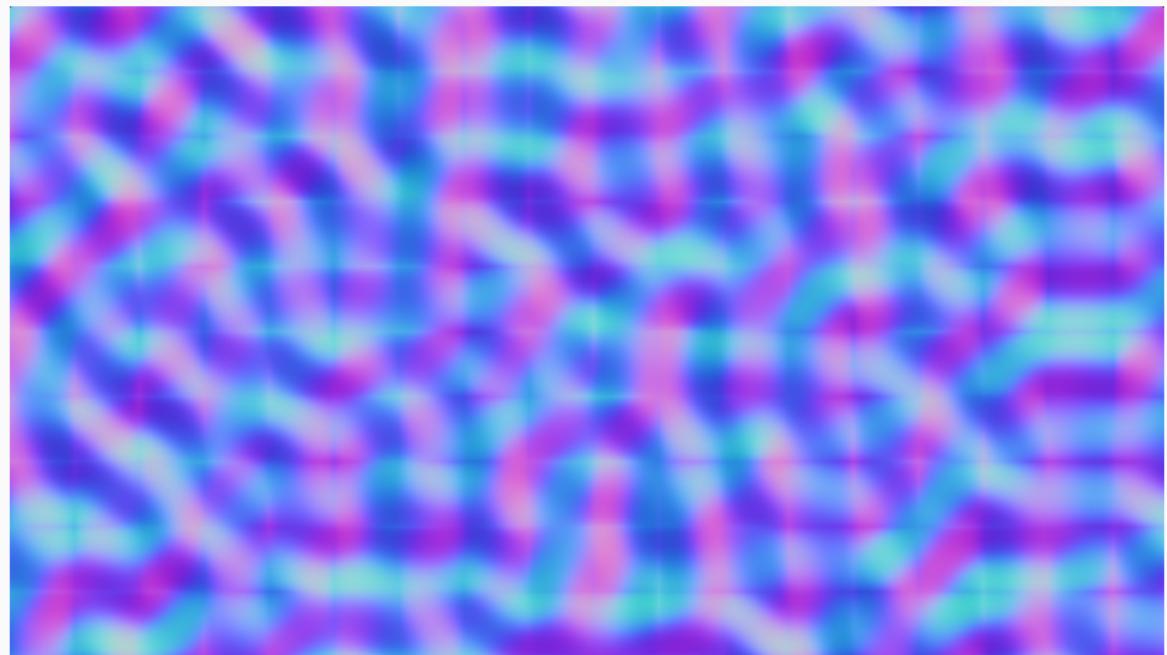
$$n = \frac{\left(-\frac{\partial z}{\partial x}, -\frac{\partial z}{\partial y}, 1 \right)}{\sqrt{1 + \left(\frac{\partial z}{\partial x} \right)^2 + \left(\frac{\partial z}{\partial y} \right)^2}}$$

- Значит, нам нужно посчитать производную от шума Перлина!

Частные производные



Карта нормалей



Нормаль к шуму Перлина

- Снова явно видна исходная сетка, по которой строился шум, но почему?

Нормаль к шуму Перлина

- Снова явно видна исходная сетка, по которой строился шум, но почему?
- Изначально мы видели сетку из-за того, что функция шума была непрерывной, но не была непрерывной её производная

Нормаль к шуму Перлина

- Снова явно видна исходная сетка, по которой строился шум, но почему?
- Изначально мы видели сетку из-за того, что функция шума была непрерывной, но не была непрерывной её производная
- Человеческий глаз очень хорошо видит разрывы в первой производной :)

Разрыв в производной



Разрыв в производной



Нормаль к шуму Перлина

- Мы избавились от видимой сетки, применив функцию *smoothstep*, у которой производные на концах отрезка $[0, 1]$ равны нулю

Нормаль к шуму Перлина

- Мы избавились от видимой сетки, применив функцию *smoothstep*, у которой производные на концах отрезка $[0, 1]$ равны нулю
- Освещение зависит от вектора нормали, а вектор нормали зависит от производной нашей функции

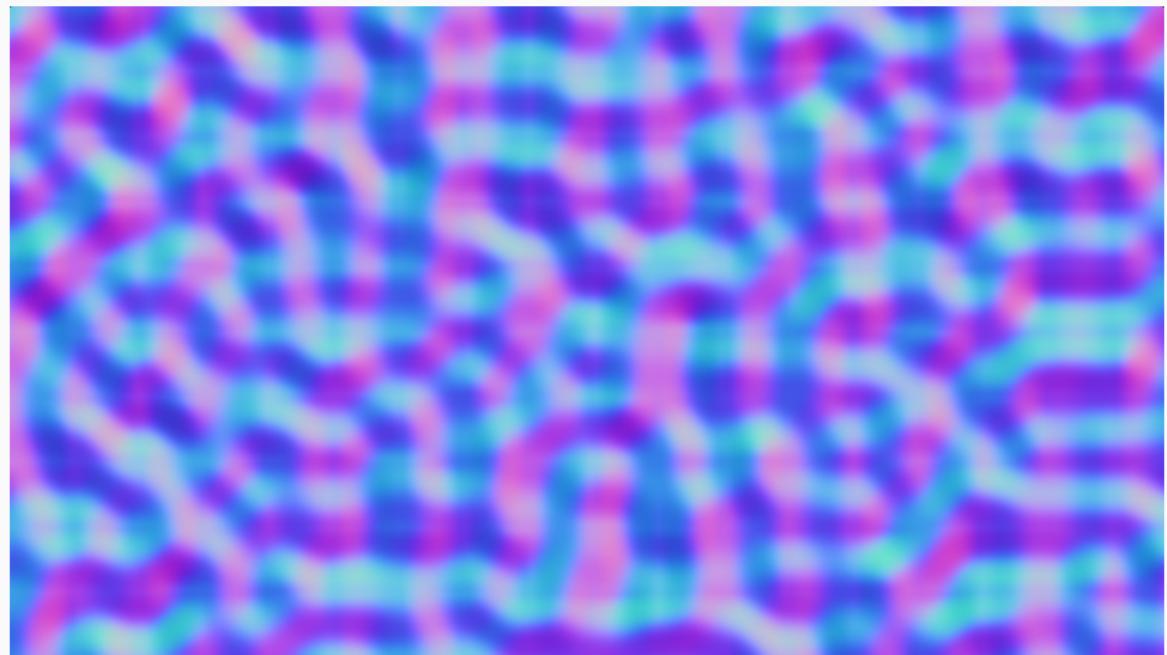
Нормаль к шуму Перлина

- Мы избавились от видимой сетки, применив функцию *smoothstep*, у которой производные на концах отрезка $[0, 1]$ равны нулю
- Освещение зависит от вектора нормали, а вектор нормали зависит от производной нашей функции
- Значит, чтобы убрать видимую сетку на нормалях, нужно сделать непрерывной вторую производную!

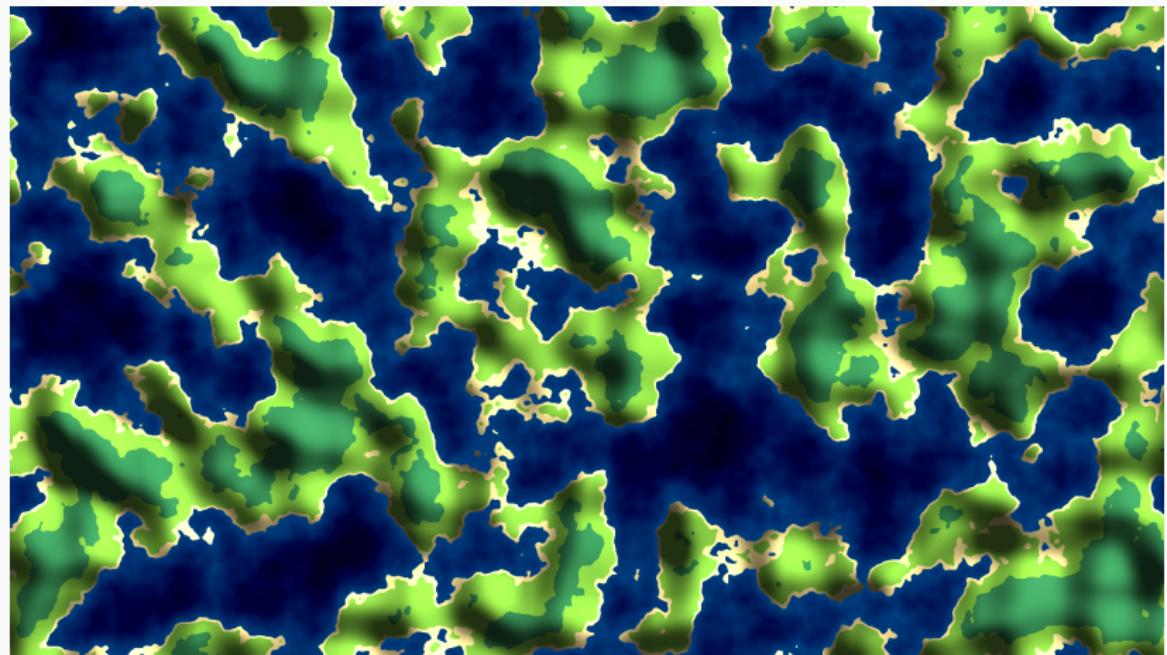
Нормаль к шуму Перлина

- Мы избавились от видимой сетки, применив функцию *smoothstep*, у которой производные на концах отрезка $[0, 1]$ равны нулю
- Освещение зависит от вектора нормали, а вектор нормали зависит от производной нашей функции
- Значит, чтобы убрать видимую сетку на нормалях, нужно сделать непрерывной вторую производную!
- Для этого есть функция *smootherstep*: $s(x) = 6x^5 - 15x^4 + 10x^3$

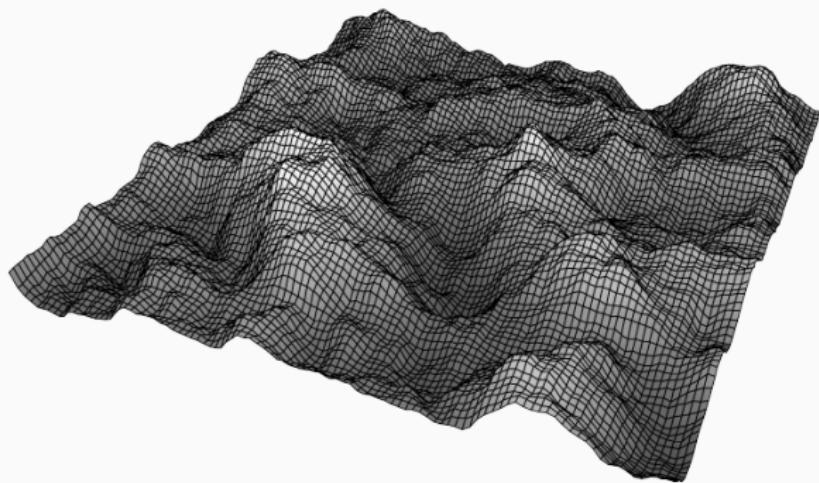
Плавные нормали



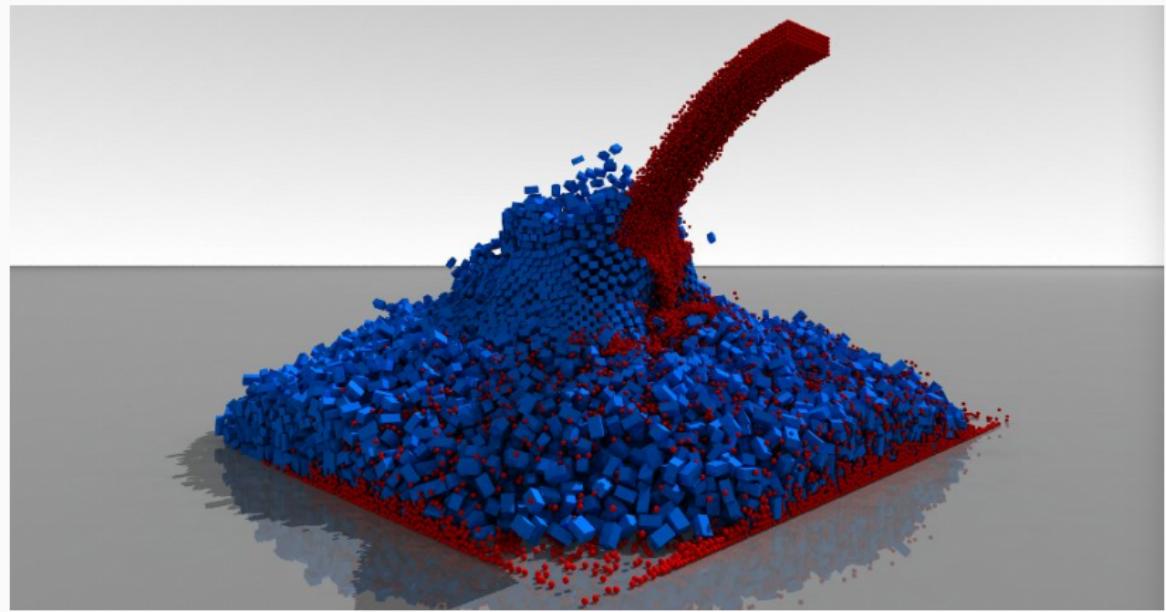
Карта с освещением



Ландшафт на основе фрактального шума



Физический движок



Физический движок

- Задача физического движка – симулировать физические взаимодействия между объектами

Физический движок

- Задача физического движка – симулировать физические взаимодействия между объектами
- Например, движение твёрдого тела описывается приложенными к нему силами F и уравнением Ньютона $F = ma$

Физический движок

- Задача физического движка – симулировать физические взаимодействия между объектами
- Например, движение твёрдого тела описывается приложенными к нему силами F и уравнением Ньютона $F = ma$
- Ускорение – это вторая производная от положения объекта $\frac{d^2}{dt^2} q$

Физический движок

- Задача физического движка – симулировать физические взаимодействия между объектами
- Например, движение твёрдого тела описывается приложенными к нему силами F и уравнением Ньютона $F = ma$
- Ускорение – это вторая производная от положения объекта $\frac{d^2}{dt^2} q$
- Физическому движку нужно решать *дифференциальное уравнение!*

Физический движок

- Задача физического движка – симулировать физические взаимодействия между объектами
- Например, движение твёрдого тела описывается приложенными к нему силами F и уравнением Ньютона $F = ma$
- Ускорение – это вторая производная от положения объекта $\frac{d^2}{dt^2}q$
- Физическому движку нужно решать *дифференциальное уравнение!*

$$\frac{d^2}{dt^2}q = \frac{F}{m}$$

Физический движок

- Обычно силы, приложенные к объектам, зависят от действий игрока

Физический движок

- Обычно силы, приложенные к объектам, зависят от действий игрока
- Обычно объектов в мире довольно много

Физический движок

- Обычно силы, приложенные к объектам, зависят от действий игрока
- Обычно объектов в мире довольно много
- Нет смысла решать физические уравнения аналитически (в виде явной формулы), лучше решать их численно и приближённо, используя дискретные шаги во времени

Физический движок

- Есть много разных методов численного решения уравнений движения: явный и неявный методы Эйлера, симплектический метод Эйлера, методы Рунге-Кутты, и т.д.

Физический движок

- Есть много разных методов численного решения уравнений движения: явный и неявный методы Эйлера, симплектический метод Эйлера, методы Рунге-Кутты, и т.д.
- Например, симплектический метод Эйлера выглядит так:

$$v_1 = v_0 + \frac{F}{m} \cdot \Delta t$$

$$p_1 = p_0 + v_1 \cdot \Delta t$$

Физический движок

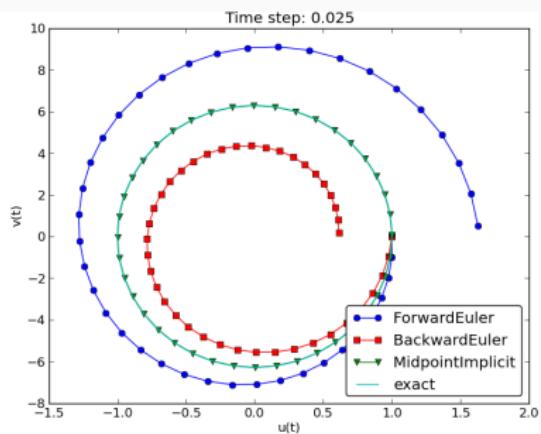
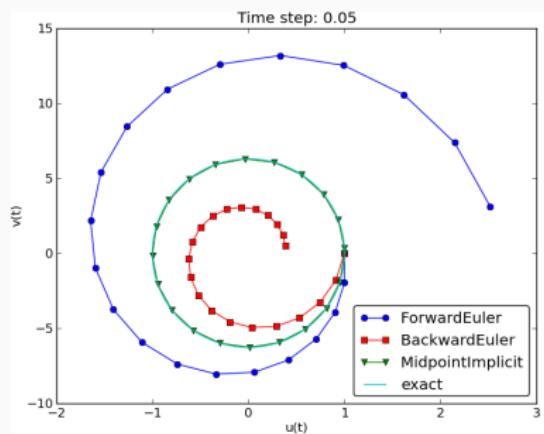
- Есть много разных методов численного решения уравнений движения: явный и неявный методы Эйлера, симплектический метод Эйлера, методы Рунге-Кутты, и т.д.
- Например, симплектический метод Эйлера выглядит так:

$$v_1 = v_0 + \frac{F}{m} \cdot \Delta t$$

$$p_1 = p_0 + v_1 \cdot \Delta t$$

- Он прост в реализации и хорошо сохраняет энергию физической системы

Методы решения уравнений движения



Вращения

- Хочется, чтобы объекты не только двигались, но и вращались!

Вращения

- Хочется, чтобы объекты не только двигались, но и вращались!
- Как описать вращение объекта?

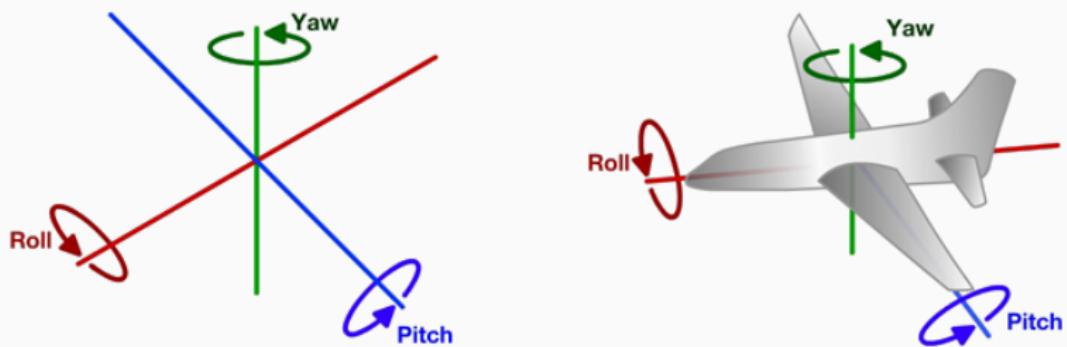
Вращения

- Хочется, чтобы объекты не только двигались, но и вращались!
- Как описать вращение объекта?
- В 2D это сделать несложно: вращение можно описать одним углом поворота

Вращения

- Хочется, чтобы объекты не только двигались, но и вращались!
- Как описать вращение объекта?
- В 2D это сделать несложно: вращение можно описать одним углом поворота
- В 3D всё становится куда веселее!

Трёхмерные вращения



Трёхмерные вращения

- Можно описать трёхмерное вращения тремя углами Эйлера

Трёхмерные вращения

- Можно описать трёхмерное вращения тремя углами Эйлера
- С этими углами очень неудобно работать!

Трёхмерные вращения

- Можно описать трёхмерное вращения тремя углами Эйлера
- С этими углами очень неудобно работать!
- Можно описать вращение матрицей 3×3

Трёхмерные вращения

- Можно описать трёхмерное вращения тремя углами Эйлера
- С этими углами очень неудобно работать!
- Можно описать вращение матрицей 3×3
- С матрицами очень удобно работать, но они крайне избыточны (нужно запомнить 9 значений вместо 3), и не любая матрица является вращением

Кватернионы

- Решение – кватернионы!

Кватернионы

- Решение – кватернионы!
- Кватернионы – это четвёрки вещественных чисел (x, y, z, w) с покомпонентным сложением и хитрым, некоммутативным законом умножения

Кватернионы

- Решение – кватернионы!
- Кватернионы – это чётвёрки вещественных чисел (x, y, z, w) с покомпонентным сложением и хитрым, некоммутативным законом умножения
- *Что-то вроде* четырёхмерных комплексных чисел

Кватернионы

- Решение – кватернионы!
- Кватернионы – это чётвёрки вещественных чисел (x, y, z, w) с покомпонентным сложением и хитрым, некоммутативным законом умножения
- *Что-то вроде* четырёхмерных комплексных чисел
- Очень интересный и важный математический объект

Кватернионы

- Через кватернионы можно выразить любое вращение трёхмерного вектора v с помощью некоторого кватерниона q и формулы

$$v \mapsto q \cdot v \cdot q^{-1}$$

Кватернионы

- Через кватернионы можно выразить любое вращение трёхмерного вектора v с помощью некоторого кватерниона q и формулы

$$v \mapsto q \cdot v \cdot q^{-1}$$

- Абсолютно все игровые движки используют кватернионы!

Распространение света

- Основная задача компьютерной графики – описать и смоделировать *распространение света*

Распространение света

- Основная задача компьютерной графики – описать и смоделировать *распространение света*
- Свет взаимодействует с объектами по очень сложным законам

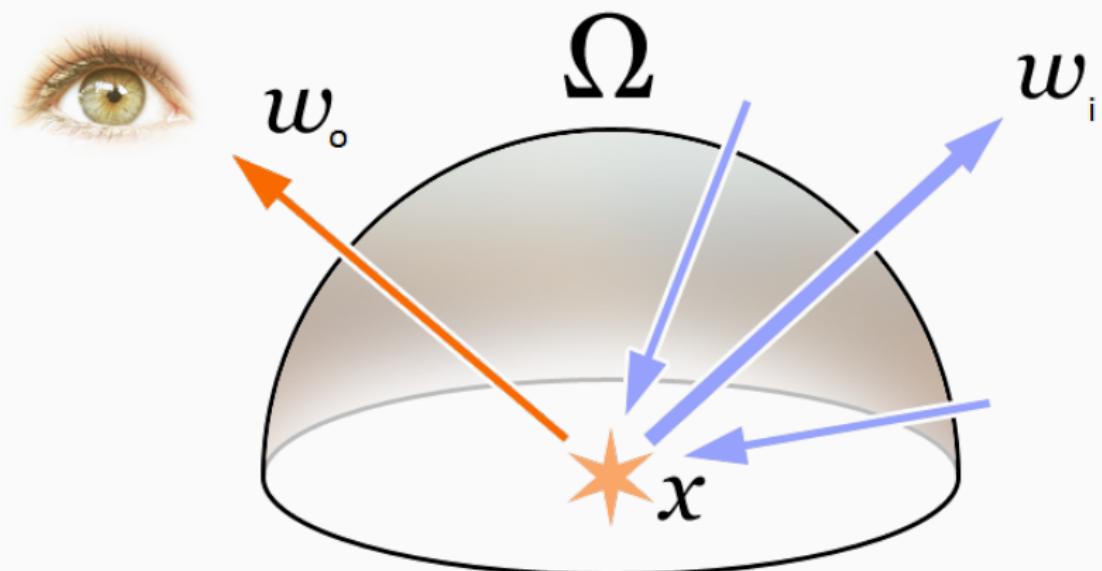
Распространение света

- Основная задача компьютерной графики – описать и смоделировать *распространение света*
- Свет взаимодействует с объектами по очень сложным законам
- В общем случае свет, выходящий из какой-то точки объекта в каком-то направлении, складывается из света, пришедшего из *всех возможных направлений!*

Распространение света

- Основная задача компьютерной графики – описать и смоделировать *распространение света*
- Свет взаимодействует с объектами по очень сложным законам
- В общем случае свет, выходящий из какой-то точки объекта в каком-то направлении, складывается из света, пришедшего из *всех возможных направлений!*
- Чтобы это смоделировать, нужно использовать *интегральный уравнения*

Распространение света



Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}, \lambda) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}, \lambda) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

- I_{out} – количество света, выходящего из точки

Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}, \lambda) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

- I_{out} – количество света, выходящего из точки
- I_e – количество излучаемого света

Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}, \lambda) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

- I_{out} – количество света, выходящего из точки
- I_e – количество излучаемого света
- I_{in} – количество света, приходящего в точку

Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}, \lambda) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

- I_{out} – количество света, выходящего из точки
- I_e – количество излучаемого света
- I_{in} – количество света, приходящего в точку
- f – функция, описывающая материал объекта

Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}, \lambda) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}, \lambda) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

- Описывает многие визуальные эффекты: тени, отражение, преломление, рассеяние света

Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}, \lambda) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

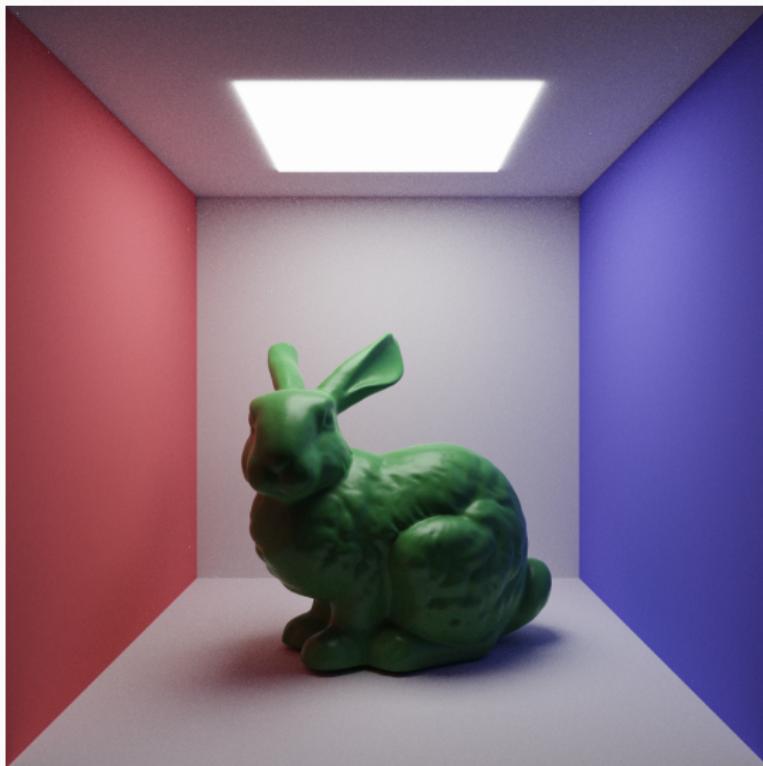
- Описывает многие визуальные эффекты: тени, отражение, преломление, рассеяние света
- Очень сложно решать!

Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}, \lambda) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

- Описывает многие визуальные эффекты: тени, отражение, преломление, рассеяние света
- Очень сложно решать!
- Один из моих курсов целиком про это уравнение :)

Уравнение рендеринга



Уравнение объёмного рендеринга

- Обычное уравнение рендеринга описывает только взаимодействие света с *поверхностью* объектов

Уравнение объёмного рендеринга

- Обычное уравнение рендеринга описывает только взаимодействие света с *поверхностью* объектов
- Многие эффекты требуют взаимодействия света с *объёмом*: туман, лучи света в пыльной комнате, и даже цвет самого неба

Уравнение объёмного рендеринга

- Обычное уравнение рендеринга описывает только взаимодействие света с *поверхностью* объектов
- Многие эффекты требуют взаимодействия света с *объёмом*: туман, лучи света в пыльной комнате, и даже цвет самого неба
- Для этого нужно ещё более сложное *уравнение объёмного рендеринга!*

Уравнение объёмного рендеринга

$$\begin{aligned} I(p + L\omega, \omega) &= I(p, \omega) \exp \left(- \int_0^L k_a(p + t\omega) dt \right) + \\ &+ \int_0^L k_e(p + t\omega) \exp \left(- \int_t^L k_a(p + s\omega) ds \right) dt \end{aligned}$$

Уравнение объёмного рендеринга

- Часто используется в медицине – например, для визуализации МРТ-обследований

Уравнение объёмного рендеринга

- Часто используется в медицине – например, для визуализации МРТ-обследований
- Используется в играх для ‘объёмных’ лучей света, тумана, облаков, или для реалистичного неба

Лучи света



Небо

