Кубический мир на JavaScript

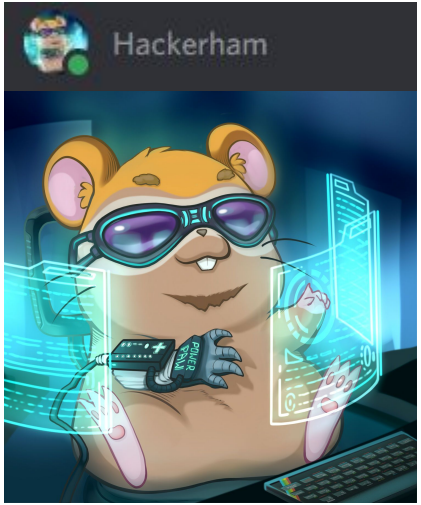
by Иван Попельшев

GameDev for Web

1 107 members

white frame

Немножко о себе



Три раза на HolyJS:

- Подборка WebGL/2d/SVG проблем
- Воркшоп как делать PixiJS приложение: 1, 2
- Кошмар с рисованием линий

В молодости выиграл много компьютерных олимпиад

Создатель gameofbombs.com , одной из MMO на заре HTML5-игр

Некромант, предпоследний человек изучавший Adobe Flash (обучил последнего)

Консультант Wargaming не уехавший на кипр

madcraft.io

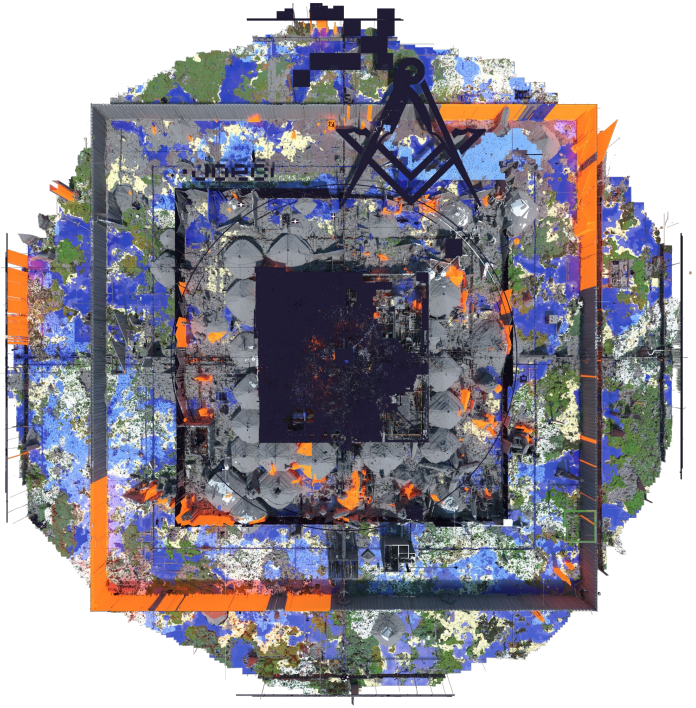

MAD
CRAFT
HolyJS 2022 Spring
@Hackerham
Change skin
Textures: Kenney
Generate mipmap
Use light

План доклада

1. Факты о Minecraft
2. Откуда взялся наш проект, цели
3. Ресурсы игры
4. Хардкор: как хранить воксели в свободном мире
5. Передышка: аналогичные проекты
6. Хардкор: основы рендеринга
7. Некоторые выводы о браузере + MC

Minecraft: масштаб игры

- Первая успешная 3D песочница
- 1.000.000.000$ от M$ чтобы отправить автора игры на пенсию
- Миллионы игроков, многие годы youtube контента
- HashiCorp (terraform, packer, vault) обожают эту игру, они бывали на jugru для devoops
- Хорошая семейная активность
- Прото - Metaverse



Проект MadCraft: краткая история

Для помощи по простым и трудным вопросам по геймдеву и в частности по WebGL у нас есть [сообщество в телеграмме]

Появляется нуб который пытается сделать что-то похожее на Minecraft, приходиться помогать с нетривиальными задачами.

В какой-то момент люди начинают путать скриншоты прототипа и реальной игры, а заходя внутрь замечают разницу только через какое-то время.

Оказывается что у автора есть своя web-студия и можно превратить это в реальный проект.

Цель: переписать игру

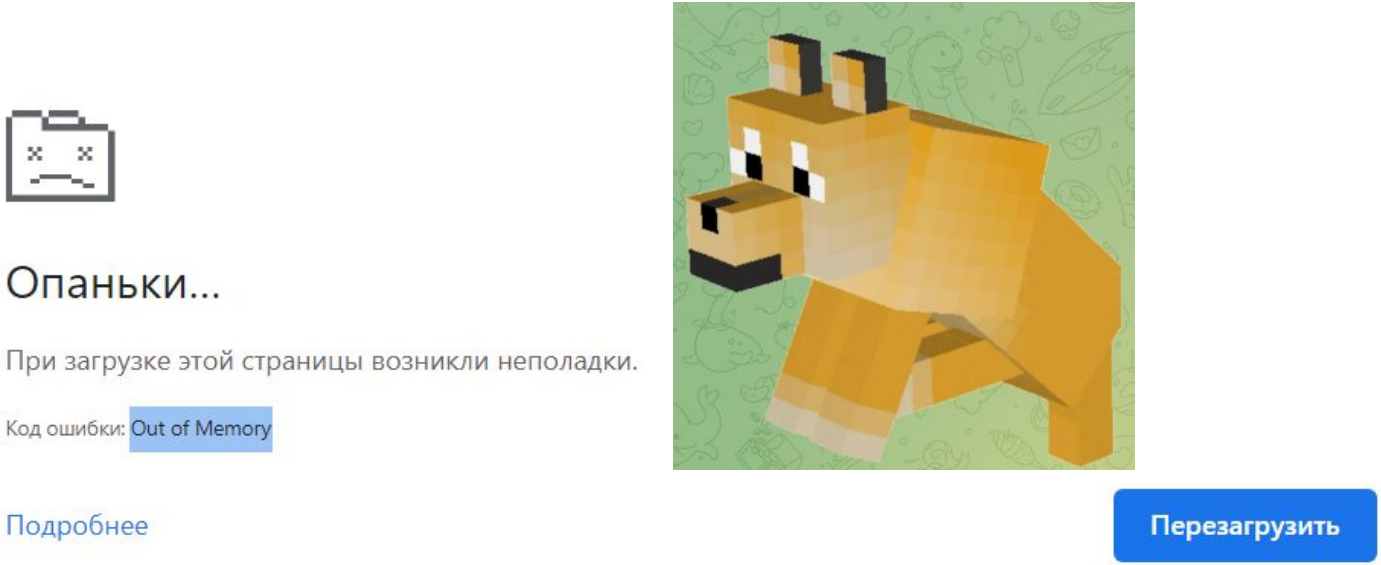
и лучшие моды

Ресурсы игры

Ресурсы MC: память

2 байта на блок x 256 (высота) x 400 x 400 = 80 мегабайт область видимости

Реальность: Out of memory



Ресурс МС: память, идеальный расчёт

Клиент

- 2 байта на блок, 80 МБ
- 1 байта на свет, 40 МБ
- геом: 72 байта на каждую видимую сторону куба, 1м сторон = 72 МБ

Минимум 192 МБ

Сервер

- 2 байта на блок, 80 МБ
- 1 байта на свет, 40 МБ
- x 100 игроков

Итого 12 гигабайт

Суровая реальность MadCraft :(

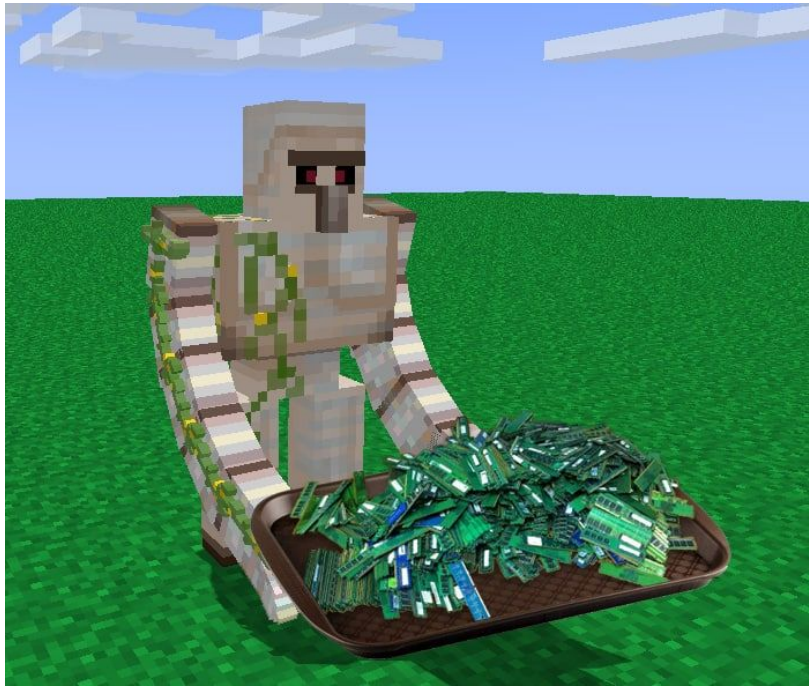
- Оно не single-copy, и это не статичные данные
- 6 байт на блок, 8 байт на свет
- Временные выделения памяти под алгоритмы могут быть достаточно большими

на самом деле
2 гига любви на клиенте

Менеджер памяти как паттерн проектирования

- Облегчить задачу Javascript GC и постараться чтобы нужные для алгоритмов данные хранились рядом а не чёрти как
- У нас не C / C++, но это не значит что мы ничего не можем сделать
- Примитивы для страницы памяти - Array, TypedArray, SharedArrayBuffer

Использовать знание внутреннего устройства компьютера не зная параметров конкретного оборудования из языка высокого уровня вполне возможно - Spectre как пример.



Менеджер памяти: что умеет

Должен уметь

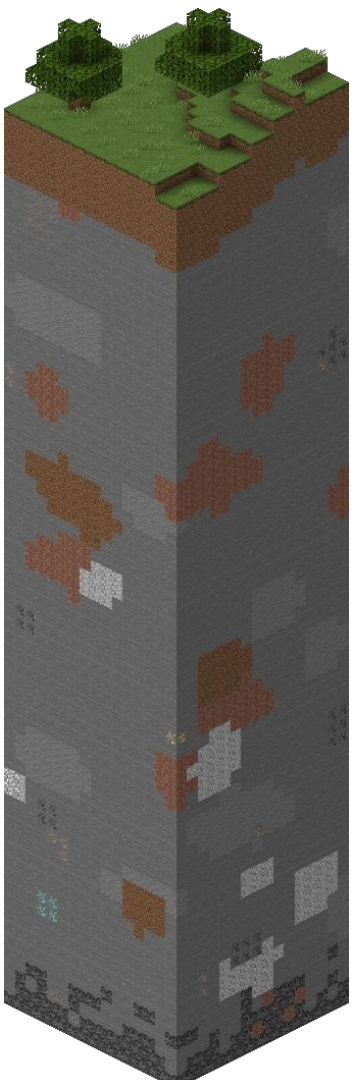
- Отбирать память у системы
- Аллоцировать память под нужды игры
- Освобождать память с ненужными объектами
- Иметь запас памяти чтобы избежать лагов
- Следить за балансом разных типов памяти под разные объекты

- Тут картинка

Воксели: разбиение мира

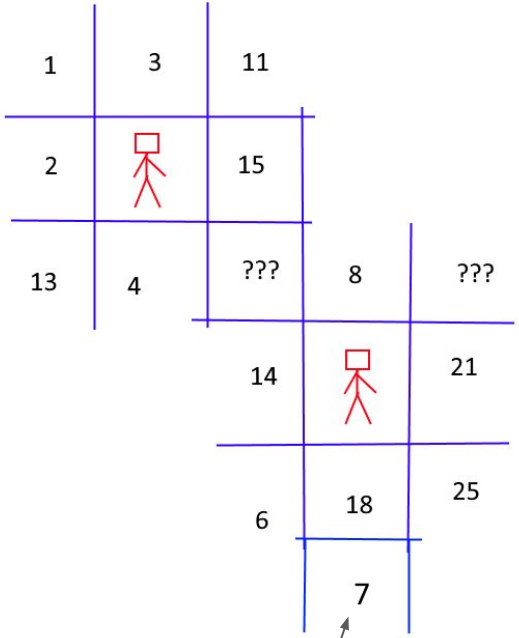
Идея в общем-то простая - надо изобрести свою модель памяти, как будто мы делаем на более низком уровне.

- Мир делится на куски одинакового размера - Chunk. В майнкрафте это обычно 16x16x128 или 16x16x16, *наверное чтобы насытить код битовыми операциями*
- Внутри каждого куска можно как-то пронумеровать блоки. *не все знают как 1d массив превращается в 3d но я это объясню*
- Текущие загруженные чанки можно тоже как-то пронумеровать, их не должно быть много.



Воксели: менеджер памяти для подгруженных чанков

- Для простоты смотрим на 2D
- Храним массив чанков, часть клеток свободна, часть занята
- При выделении проходимся по массиву начиная с предыдущего выделенного
- В примере новые чанки займут места 9 и 10
- Каждый чанк это объект с выделенным TypedArray внутри



прошлый выделенный (allocated)

	1	2	3	4		6	7	8	*9*
10	11		13	14	15			18	

[object Object]

Uint16Array[4096]

скоро удалят

Воксели: как сжать 3D в 1D

Для простоты рассмотрим 2D случай: пусть нужен чанк размера **width** x **height** сжать в 1D

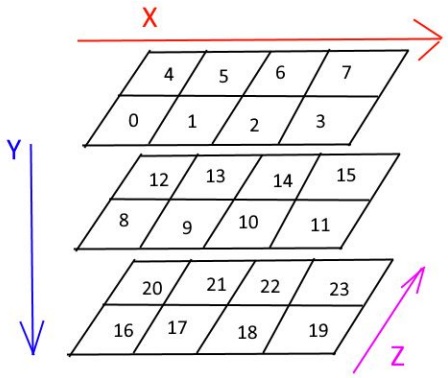
Тогда индекс ячейки (**x**, **y**) будет **width * y + x**

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

4 * y + x

Аналогично для 3D **width** x **depth** x **height**

Ячейка (**x**, **y**, **z**) индекс **(depth * width) * y + width * z + x**

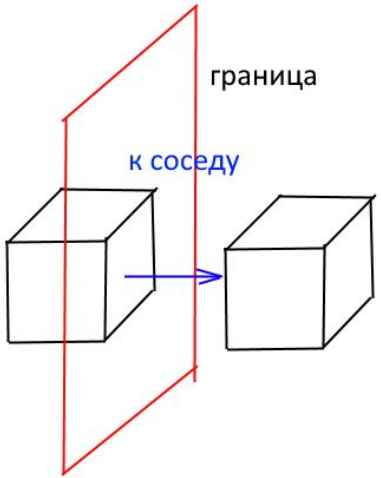


4 * 2 * y + 2 * z + x

Воксели: проблема соседей

Довольно часто в алгоритмах надо быстро узнать значения не в ячейке, а ещё в её соседях

```
const ax = pos.x + p.x, ay = pos.y + p.y, az = pos.z + p.z;
if(ax >= 0 && ay >= 0 && az >= 0 && ax < CHUNK_SIZE_X
    && ay < CHUNK_SIZE_Y && az < CHUNK_SIZE_Z) {
    // чанк тот же самый
    v.x = ax; v.y = ay; v.z = az;
} else {
    // другой чанк
    v.x = (ax + CHUNK_SIZE_X) % CHUNK_SIZE_X;
    v.y = (ay + CHUNK_SIZE_Y) % CHUNK_SIZE_Y;
    v.z = (az + CHUNK_SIZE_Z) % CHUNK_SIZE_Z;
    // дальше спрашиваем менеджера а где этот чанк
    chunk = chunkManager.getChunk(chunk.x + pos.x, chunk.y + pos.y, chunk.z + pos.z);
}
// нам ведь ещё индексы считать...
```



Воксели: поля

Попробуем хранить данные "с полями" (margin), скопируем данные из соседних чанков по краям. Тогда формула индекса поменяется:

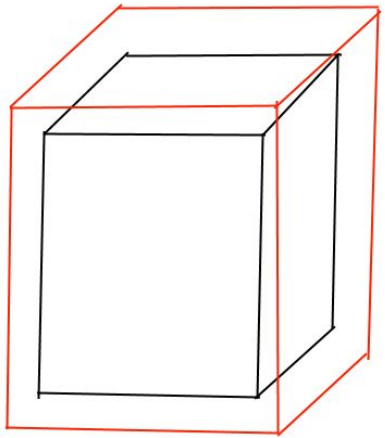
(width + 2) * (depth + 2) * (y + 1) + (width + 2) * (z + 1) + (x + 1)

В коде это будет выглядеть кошмарно.

Чтобы всё упростить, введём общую формулу хранения 3D - массива внутри большего 3D - массива расплющенного в 1D

cx * X + cy * Y + cz * Z + cw

где (**cx**, **cy**, **cz**, **cw**) какие-то рассчитанные коэффициенты зависящие от 1. порядка осей 2. 3D-размера 3. размер полей



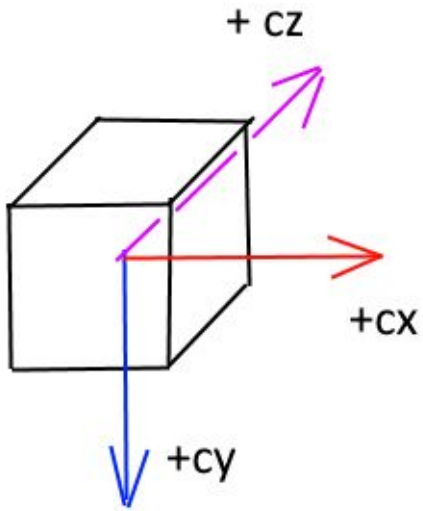
Воксели: общая формула для индексов

При использовании формулы

cx * X + cy * Y + cz * Z + cw

взятие соседей по индексу становится совсем простым

```
index = cx * x + cy * y + cz * z + cw;
i_here =  data[index]
i_up =    data[index + cy];
i_down =  data[index - cy];
i_north = data[index + cz];
i_south = data[index - cz];
i_east =  data[index + cx];
i_west =  data[index - cx];
```



Воксели: объектная часть, проблема

В начале объяснения я сделал большое допущение, которое теперь надо исправить.

Есть много блоков с дополнительными данными, которые никак не влезут в 2 байта, например сундуки, хранящие предметы.

Кроме того, с чистыми данными неприятно работать, хочется объекты.

Воксели: объектная часть, решение

Допустим что у разных типов блоков иногда бывают дополнительные поля, в полях хранятся числа или объекты.

Внутри объекта чанка, для каждого поля можно хранить отдельный объект, ключом в котором будет номер блока в чанке, а значением какой-то объект.

```
index = cx * x + cy * y + cz * z + cw;
chunk.extra_data[index] = { // сундучки
    items: [ /* ... */ ],
}
chunk.power[index] = 15; // для лампочек, воды
```

Воксели: объектная часть, структура для поля

Array

- умеет хранить разреженные данные где много пропусков
- В качестве ключей - числа больше нуля, нам подходит

Интересно

Map

- Фиксированный налог памяти на каждый элемент
- Тоже любит числа в качестве ключей, но не оптимизирует конкретно под них

Стандартно

Object

- Фиксированный налог памяти на каждый элемент
- Ключи - строки, каждый раз JS может конструировать их по числам

Отстой

Воксели: работа как с объектами

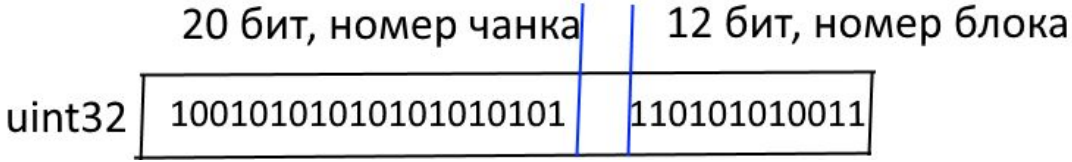
Логику всё-таки хочется писать с использованием объектов.

Пример временного View для блока:

```
export class BlockView {
    constructor(chunk, vec, index) { this.chunk = chunk; this.vec = vec; this.index = index; }
    get worldPos() {
        const {chunk, vec} = this;
        return new Vector( x: chunk.x + vec.x,  y: chunk.y + vec.y,  z: chunk.z + vec.z);
    }
    get data() { return this.chunk.data[this.index]; }
    get extra_data() { return this.chunk.extra_data.get(this.index); } // если поле - Map
    get power() { return this.chunk.power[this.index]; } // если поле - Array
    set data(value) { /* кошмар переходящий в соседние чанки */ }
}
```

Воксели: адресация в uint32

- В виду переиспользования кода структуры данных, ~~извр~~ игр с порталами, размер чанков может быть разным, но допустим максимально чанк хранит 4096 блока
- Пусть макс зона видимости игрока занимает 1000 чанков, и максимум 1000 игроков на сервере, максимум миллион чанков
- При других числах надо использовать другое разбиение



Воксели: распаковка координаты из uint32

Код распаковки достаточно простой

```
BLOCK_BITS = 12;
blockId = blockIndex & ((1<<BLOCK_BITS) - 1) ;
chunkId = blockIndex >> BLOCK_BITS;
localY = Math.floor( x: blockId / cy);
localZ = Math.floor( x: (blockId - localY * cy) / cx);
localX = blockId % cx;
worldX = chunkManager.chunks[chunkId].x + localX;
worldY = chunkManager.chunks[chunkId].y + localY;
worldZ = chunkManager.chunks[chunkId].z + localZ;
```

Возможный тормоз

Воксели: ускорение адресации для доп полей

- Объект Chunk может разрастись из-за реализации, а быстро данные получать хочется
- Продублируем главную информацию чанках в отдельном Uint32Array

x0	y0	z0	L0	x1	y1	z1	L1	x2	y2	z2	L2	…

- Это не единственные важные данные о чанке, например при переменном размере придётся добавить cx, cy, cz, cw
- Может добавиться адрес в другой структуре данных, например для света
- Такой массив можно загрузить текстурой в WebGL, и тогда на шейдерах можно запрограммировать те же методы, например распаковку из uint32

Менеджер памяти: tradeoff

Не бывает идеальных паттернов. Где тут подвох?

- Используя свою адресацию вместо обычных ссылок, мы рискуем оказаться в ситуации когда по нужному адресу уже живёт другой чанк
- Конечно, есть обходы: удалять чанки в течение некоторого времени, чтобы плохие ссылки ушли из текущих итераций алгоритмов на воркерах
- На самом деле, используя воркеры мы уже создаём себе garbage collecting problems, потому что ссылок между воркерами не существует

За счёт чего код самого MM легче чем на C/C++ ?

- Для управляющих структур используются существующие структуры JS и используется Garbage Collector из виртуальной машины V8. Управляющих объектов мало, v8 GC легко с ними справится.

Аналогичные проекты

передышка

Creativerse (Unity)

Boundless (C++/JS)

Worlds FRVR (html5)

4D Miner

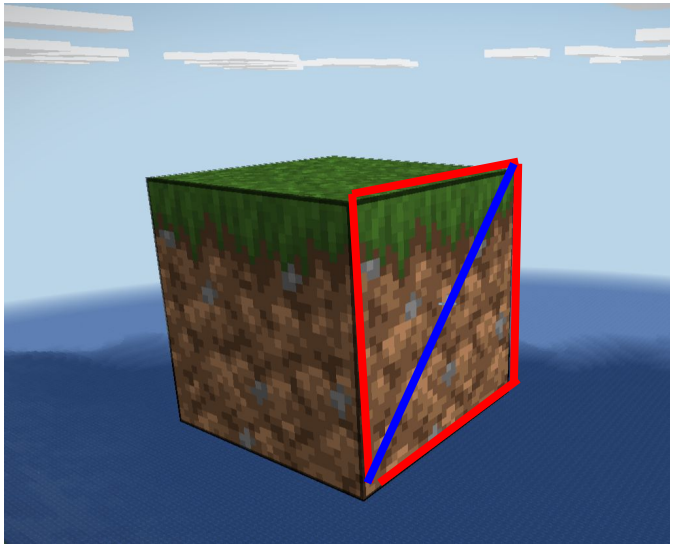
Рендеринг

сейчас снова будет жесть

Rendering: основной элемент сцены

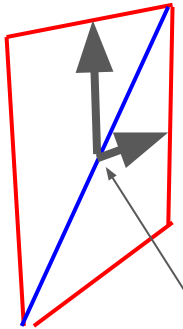
ChunkID	X	Y	Z	uX	uY	uZ	vX	vY	vZ	tX	tY	flags	…	…

координаты чанка, полезные данные



tX, tY - текстурные координаты

vX, vY, vZ - ось U



uX, uY, uZ - ось V

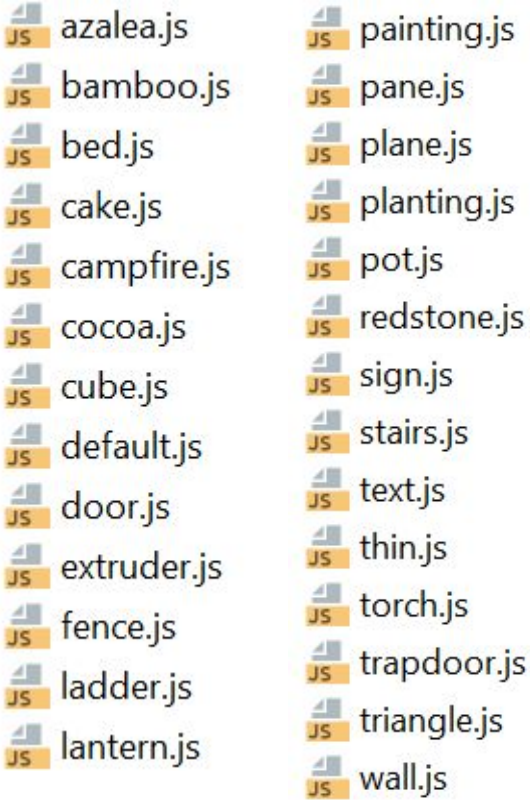
X, Y, Z - центр

Rendering: генерация quad-ов по блоку

Каждый блок имеет несколько соседей.

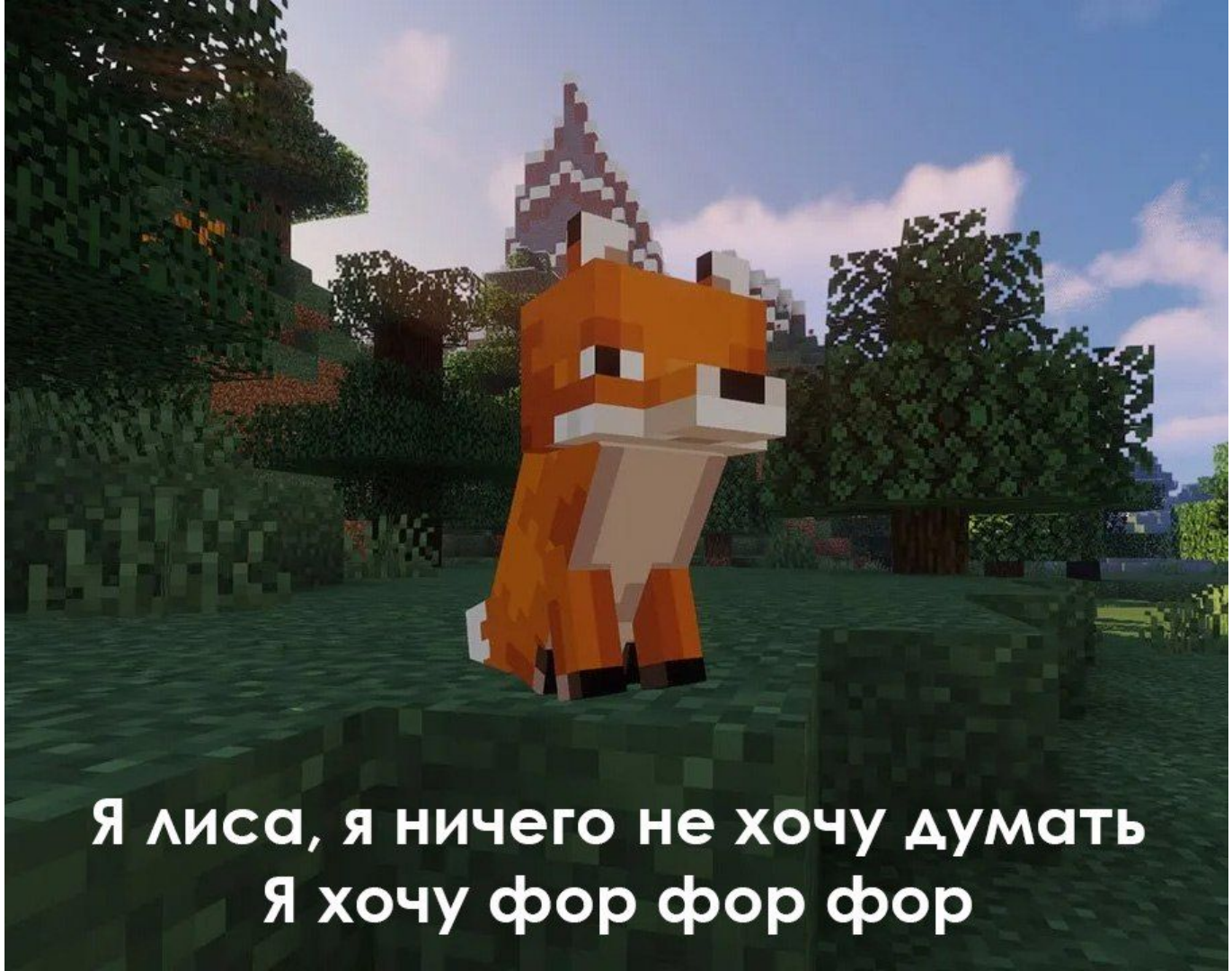
- Для обычных блоков, если блок непрозрачный, а один из соседей прозрачный - между ними надо нарисовать квад

- Для сложных типов блоков можно вызывать модель которая нарисует квады (двери, сундуки, и.т.п.)



Rendering: Как нарисовать все блоки?

```
for (let chunk of chunks)
    for (let Y = 0; Y < chunk.size.y; Y++)
        for (let Z = 0; Z < chunk.size.z; Z++)
            for (let X = 0; X < chunk.size.x; X++) {
                let block = chunk.getBlock(X, Y, Z);
                let neib = chunk.getNeighbours(block);
                let model = models[block.style];
                model.draw(renderer, block, neib);
            }
```


Я лиса, я ничего не хочу думать
Я хочу фор фор фор

Quad instance: расчёт памяти

- где-то 1.000.000 кроликов в стандартной области видимости
- из них рисуются объекты - блоки, двери, кровати, факела, и.т.д.
- в общем виде занимает где-то 18 float , x 4 = 72 байта
- "хорошие" квады, строго грани блоков, можно упаковать - ещё лучше

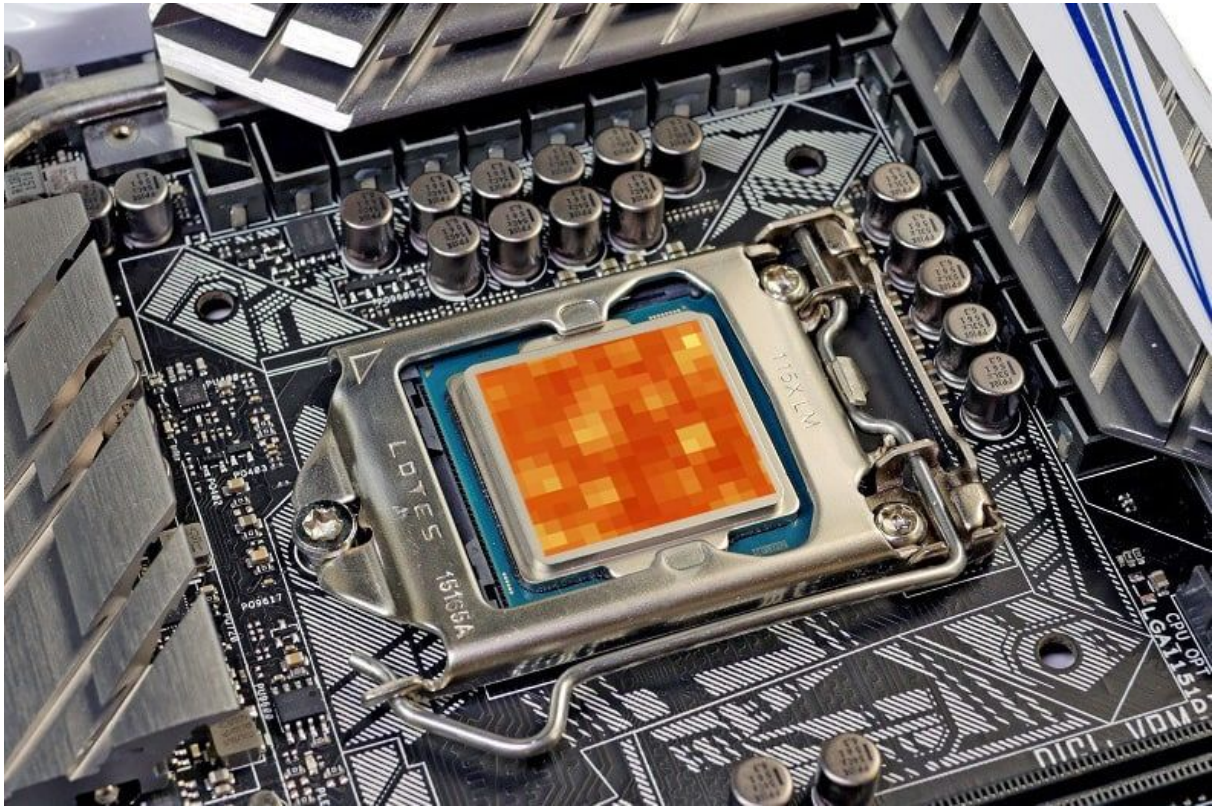
Получается память уже не такая проблема.

Так что будет если проходиться по всем блокам и отдавать команду на отрисовку каждого квада?

40 миллионов блоков в области видимости x **60FPS** = ???

ответ на следующем слайде

Ресурсы MC: процессор на 40M блоков при 60FPS



Chunk & Quad: re-mesh, запоминание

- Для каждого чанка, для каждого типа материала (прозрачный, непрозрачный, анимированный) можно генерировать по буферу с квадами, это называют Mesh.
- Если в чанке ничего не менялось - переиспользовать старый буфер
- Нужна очередь на генерацию Mesh-эй, чтобы не создать дикие лаги когда поменялось много чанков
- Генерация меша - это FOR FOR FOR по чанку с вызовом моделей блоков

Chunk & Quad: Менеджер памяти

Из-за запоминания у нас опять начинают выделяться и освобождаться большие куски памяти под буфера, память фрагментируется и выделяется много излишков, нужен свой менеджер.

640 quad 16k bytes	640 quad 16k bytes	640 quad 16k bytes	-	640 quad 16k bytes	-	640 quad 16k bytes	640 quad 16k bytes
Chunk 1	Chunk 1	Chunk 2		Chunk 2		Chunk 1	Chunk 1

Команды WebGL: draw-call

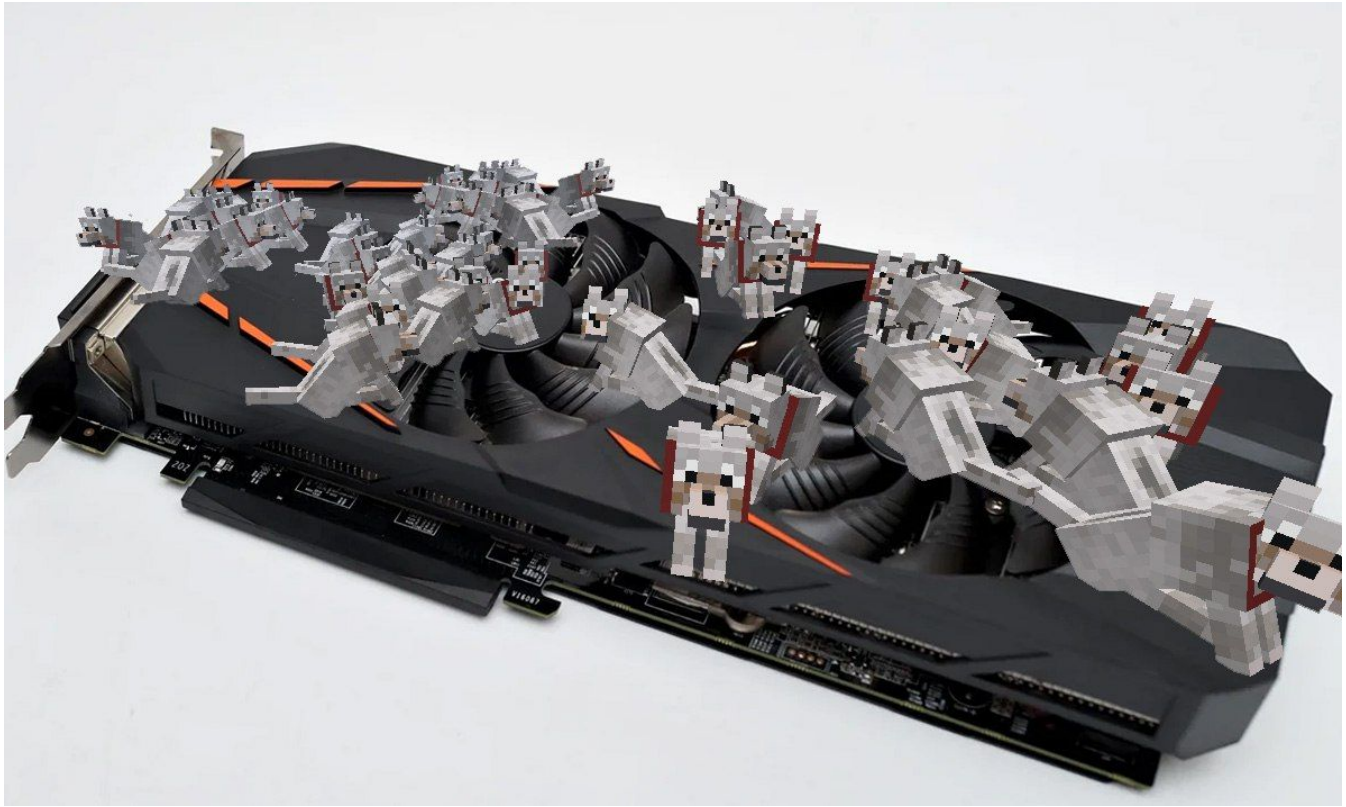
На каждый буфер надо отдельно вызывать bindBuffer() и drawElementsInstanced().

Получается из-за экономии памяти мы увеличили количество webgl-вызовов, их итак много (200 - 1000 чанков?), а мы ещё самым жирным добавили разделение на страницы.

Плюс по 1 drawcall на каждое животное может быть…

Можно ли это как-то "забатчить" (соединить drawcalls)?

Ресурсы MC: видеокарта и тормозная прослойка WebGL



Команды WebGL: multi-draw-call

2021 год, draft нового метода: multiDrawArraysInstancedBaseInstanceWEBGL

Настоящее, 2022 год, экспериментальный флаг в Chromium и Safari

Текущая неделя: битва с Mozilla за то чтобы добавить это в FF

Можно подать массив из отрезков (first, count) рисуемых из буфера instance-ов.

Решение:

- Выделять огромный буфер заранее, и делить его на страницы
- Вызывать multiDrawArraysInstancedBaseInstanceWEBGL с данными по каждой странице: порядок отрисовываемых страниц и сколько инстансов рисовать с каждой отдельно.

PR для дебага принят в SpectorJS


```
uniform1f: WebGLUniformLocation - ID: 16, 0.03125
uniform1f: WebGLUniformLocation - ID: 17, 0.0009765625
uniform1f: WebGLUniformLocation - ID: 20, 4
bindVertexArray: WebGLVertexArrayObject - ID: 211
bindBuffer: ARRAY_BUFFER, WebGLBuffer - ID: 423
bufferSubData: ARRAY_BUFFER, 67615128, [..(20687418)..], 16903782,
bufferSubData: ARRAY_BUFFER, 67696848, [..(20687418)..], 16924212,
uniform3f: WebGLUniformLocation - ID: 10, -6.548999999999978, -6.99
multiDrawArraysInstancedBaseInstanceWEBGL: TRIANGLES, drawCount=317
uniform4fv: WebGLUniformLocation - ID: 29, [..(4)..]
disable: CULL_FACE
uniform1f: WebGLUniformLocation - ID: 28, 0.5
bindVertexArray: WebGLVertexArrayObject - ID: 211
uniform3f: WebGLUniformLocation - ID: 10, -6.548999999999978, -6.99
multiDrawArraysInstancedBaseInstanceWEBGL: TRIANGLES, drawCount=262
uniform4fv: WebGLUniformLocation - ID: 29, [..(4)..]
enable: CULL_FACE
uniform1f: WebGLUniformLocation - ID: 28, 0.5
activeTexture: TEXTURE4
bindTexture: TEXTURE_2D, WebGLTexture - ID: 5
uniform1f: WebGLUniformLocation - ID: 16, 0.03125
uniform1f: WebGLUniformLocation - ID: 17, 0.015625
uniform1f: WebGLUniformLocation - ID: 20, 4
```


Render: webgl
Renderer: ANGLE (NVIDIA, NVIDIA GeForce RTX 3080 Laptop GPU Direct3D11 vs_5_0 ps_5_0, D3D11)
FPS: 139 / 2.31 ms
MAT: NULL
Game mode: Spectator
LAG: 43ms
Day: 0, Time: 11:11
Chunks drawed: 366 / 3607 (12)
Quads: 415026 / 852566 / 55Mb
Lightmap: 0 / 0Mb
Packets: 190/42
Particles: 0
Drawcalls: 2 + 15(multi)
XYZ: 2882, 109, 2783 / 0 km/h
CHUNK: 180, 2, 173
Online:
Hackerham ← YOU
17 multi-drawcalls
Get oak logs
Mine 5 oak logs ... 0/5
Нажми [TAB], чтобы увидеть подробности

Выводы

приехали

Чем плох браузер для МС

1. дополнительная прослойка для взаимодействия с видяхой в WebGL приводит к потере производительности, нужно максимально снижать drawcalls и надо знать как это делать.

2. перф у JS ниже других языков, готовьтесь сильнее оптимизировать алгоритмическую часть или выносить низкий уровень на WASM

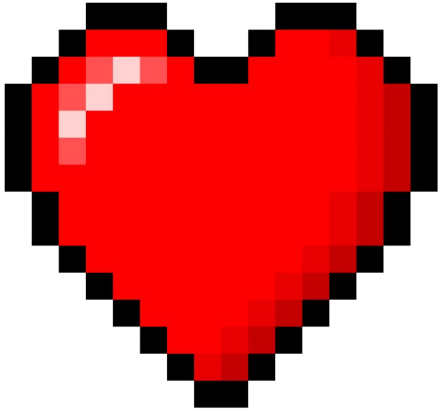
3. Многие массивы нужны в двойном экземпляре - один на главном потоке, другой на соответствующем воркере. Мы платим памятью за производительность, а памяти на самом деле меньше чем у игр для платформы x64.

4. рефреш по F5 и ctrl+W. Игроки привыкли что на F есть определенный функционал, например вид от третьего лица. W для бега, ctrl используется для автобега.

Чем хорош JavaScript для MC

- Структура данных для хранения чанка из коробки - sparse array
- Это скрипт, игровую логику на нём набить легко
- shared-логика очень удобная, высокая степень связи сервера и клиента, даже техническая часть - общие веб-воркеры.
- моддинг и поддержка версионирования - можно перезагрузить клиент игрока с нужными настройками прямо при подключении к серверу
- динамическая типизация JS упрощает создание GL рендерера, кода нужно гораздо меньше

MC и web подходят друг для друга



Спасибо за просмотр!

Хотите помощи - опишите свою проблему в
https://t.me/gamedevforweb
Хотите поиграть (можно с друзьями) - идите на
https://madcraft.io
Хотите пообщаться о игре - присоединяйтесь к
https://discord.gg/g9qZT5w2aA

white frame

Оформление слайдов кроликами:

https://vk.com/animactiondecks