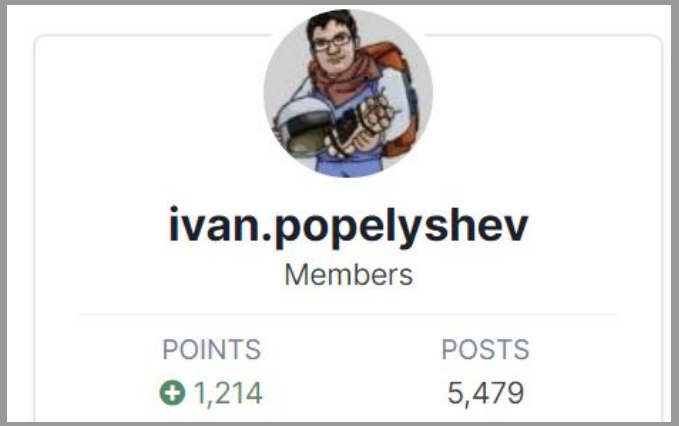
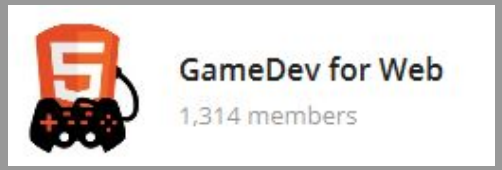
HTML 5 GAME DEVS

https://t.me/hackerham



https://t.me/gamedevforweb



PixiJS

Цель

Я делаю крутую штуку и хочу про это рассказать

Тема слишком сложная и перенасыщена кодом

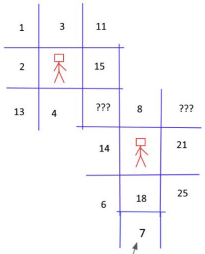
Хочу объяснить понятно, и пропускаю весь кошмар

В предыдущих частях

https://www.youtube.com/watch?v=4WMk5krAVL0

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

https://www.youtube.com/watch?v=w8wQcsZEvpw

https://www.youtube.com/watch?v=ZvhQ8qi6jwY

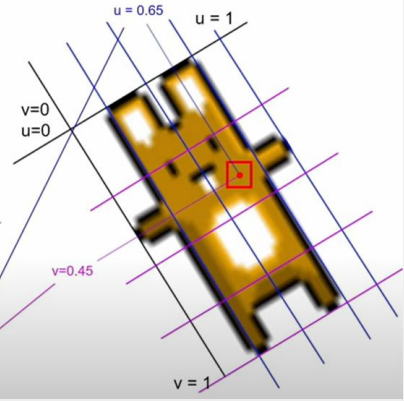
Интерполяция: финал

1. Для каждой вершины рисуемого треугольника задаём U, V - координаты на текстуре кролика. Обычно это 3 угла из 4 (0, 0) (0, 1) (1, 0) (1, 1)
2. WebGL интерполяция по этим значениям восстанавливает формулы функции U(x, y), V(x, y)
3. Ура, для нас нахаляву посчитают значения U, V для центра пикселя!
4. Цвет кролика у нас один на всех вершинах - он не изменится при интерполяции

А вот тут надо дописать, как после интерполяции рассчитать цвет пикселя

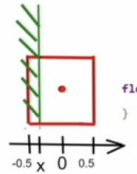
Fragment (pixel) Shader

```
vec4 textureColor = texture(bunny_sampler, in_uv);
gl_FragColor = textureColor * in_color;
```

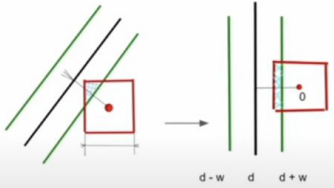


Вот это поворот!

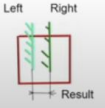
- Забудьте о повороте, ошибка где-то +-4% альфы
- Напишем функцию для пересечения пикселя и полуплоскости
- Покрытие пикселя - это разница между двумя функциями пересечения полуплоскостей




```
float pixelLine(float x) {
    return clamp(x + 0.5, 0.0, 1.0);
}
```




```
in vec4 vColor; //color style
in float vType; //type of segment/joint
in float d; //varying signed distance
in float w; //varying half-width

void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    float result = 0.0;
    if (vType == 0.0) {
        float left = pixelLine(d - w);
        float right = pixelLine(d + w);
        result = right - left;
    }
    fragColor = vColor * result;
}
```



Приём (принцип?) математики: деление с остатком

Работа = GPU x шейдер + остаток на JS

работа = индусы x рутина + прораб с образованием

Статья = Суть x опыт + Факты

Инфа = Архив x словарь + не_пожалось_JPG

Я не боюсь того, кто изучает 10 000 различных ударов.

Я боюсь того, кто изучает один удар 10 000 раз.

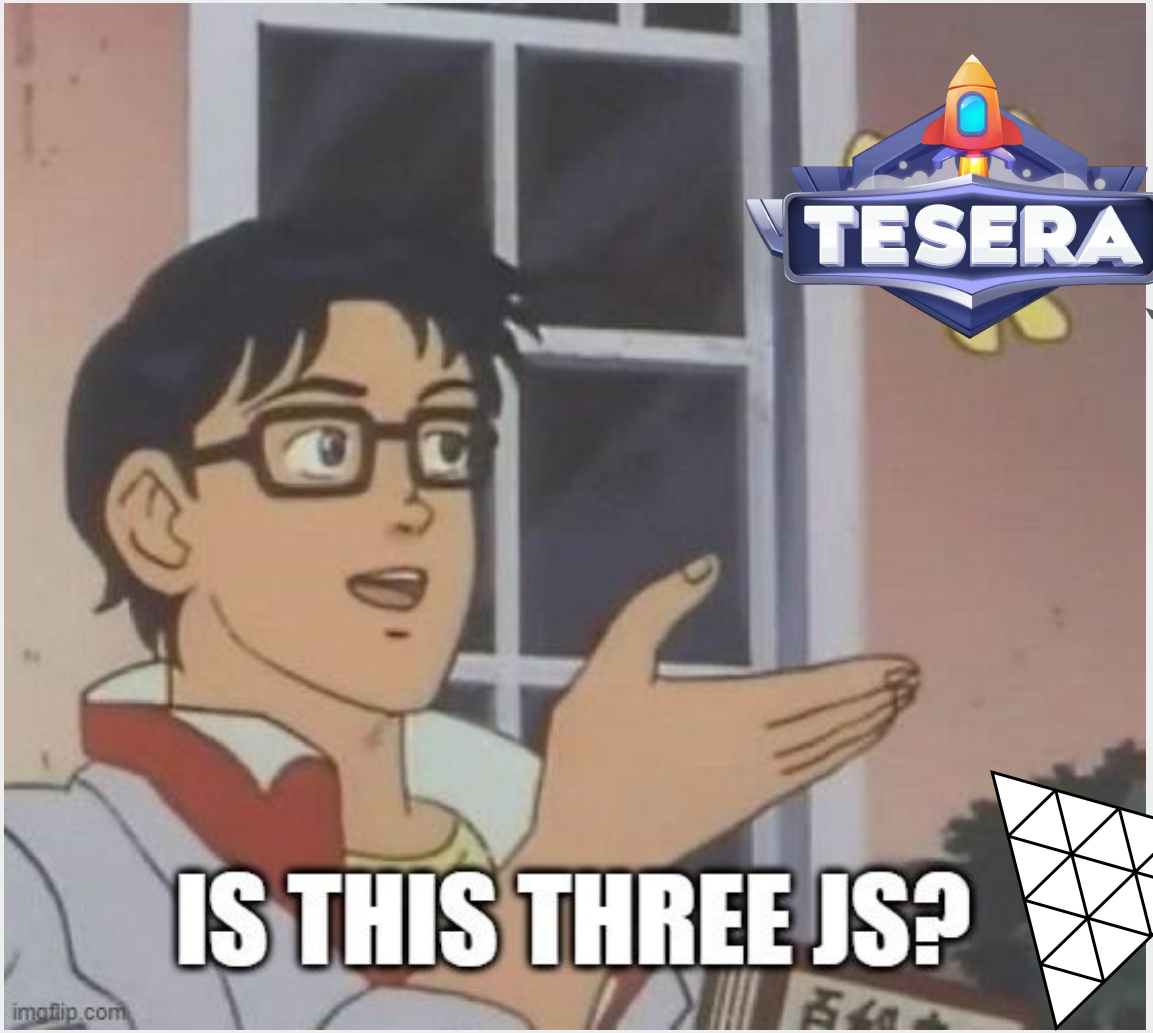
А как делить

работу, память, вычисления

доклад с намёком на прикладную математику

Внимание, вопрос:

Что такое текстура?



Blockbench

A low-poly 3D model editor

PixiJS

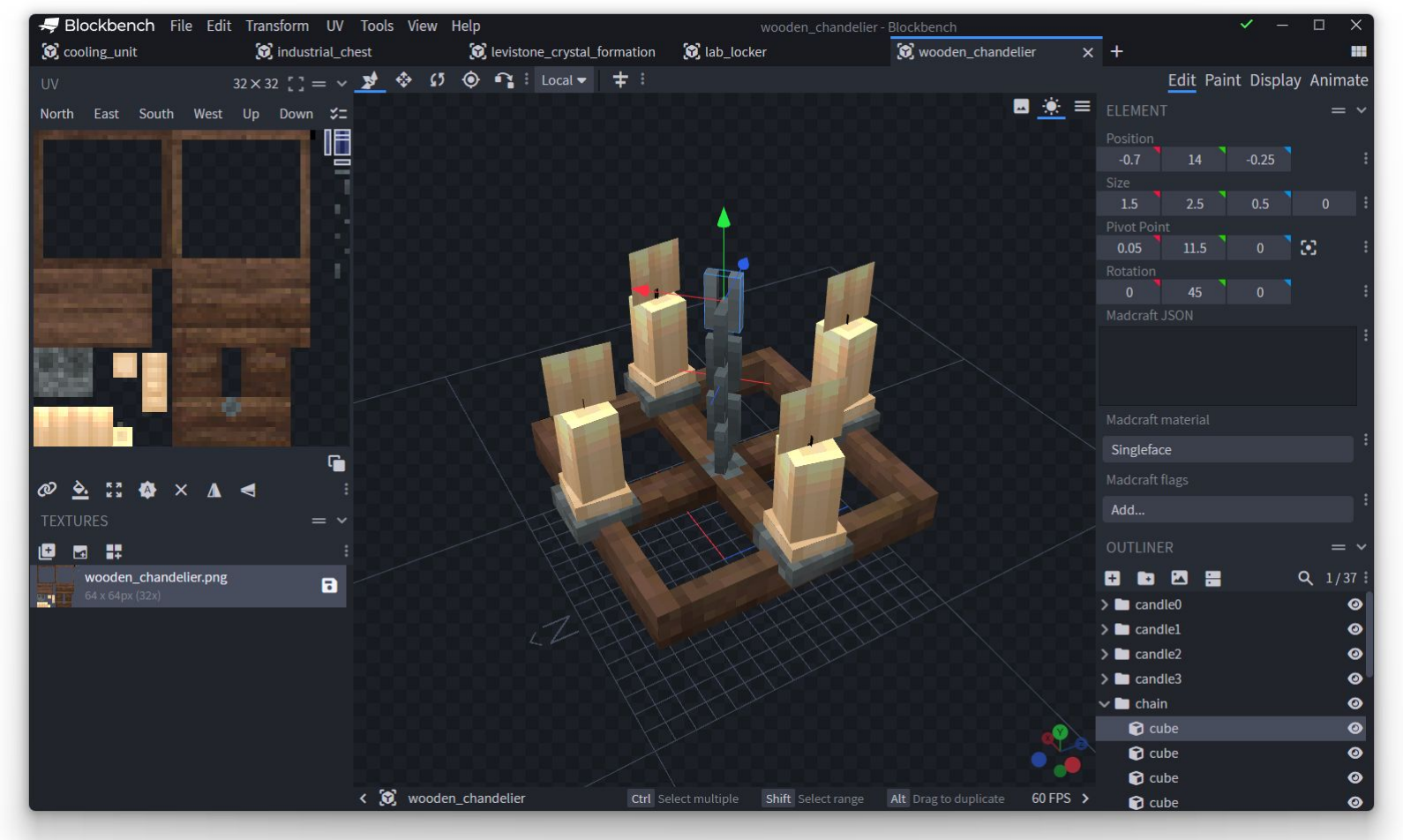
glMatrix

Javascript Matrix and Vector library for High Performance WebGL apps

3D движок состоит из

- Математика: можно просто взять gl-matrix
- Абстракция от WebGL: можно взять oframe/OGL, REGL, я взял PixiJS (она лучше всех, и халявное 2D)
- Shader-Lib: шейдера пишутся на C-подобном языке, собирать разные функции в один шейдер надо concat,
  решение в Three JS нас не удовлетворяло
- Самое жирное: поддержка моделей, анимация и их ускорение : нам нужен BlockBench

Основа 3D-движка: Формат BlockBench



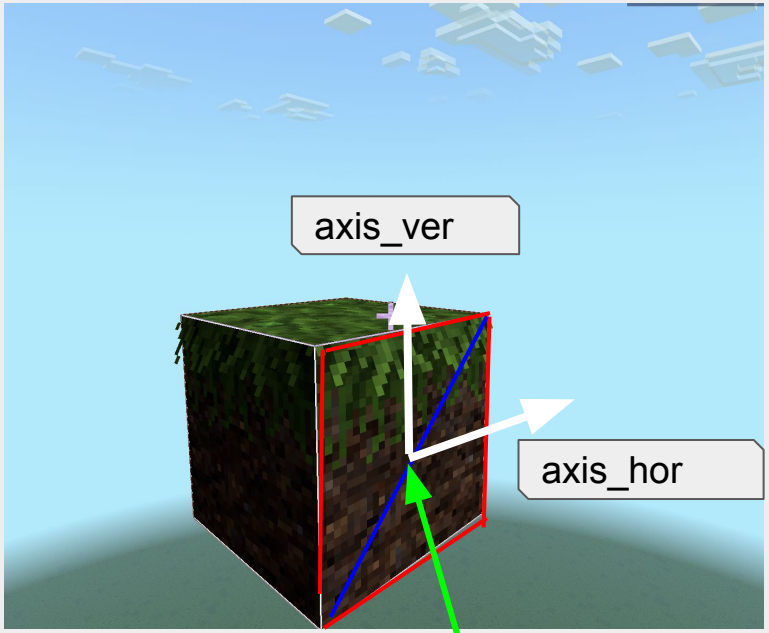
Quad: основной примитив графики

```
{
    a_chunk_id   : int
    a_position   : vec3
    a_axis_hor   : vec3
    a_axis_vert  : vec3
    a_uv         : ivec4
    a_color      : int
    a_flags      : int
}
```

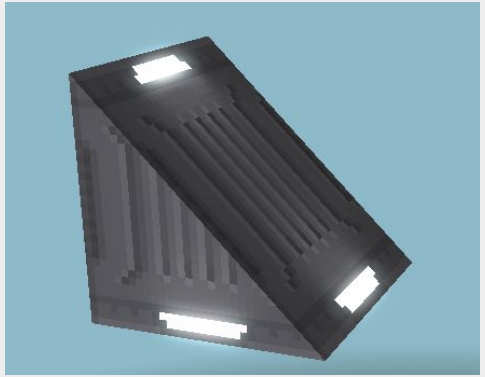
номер чанка, меша, кости, или что ещё надо будет

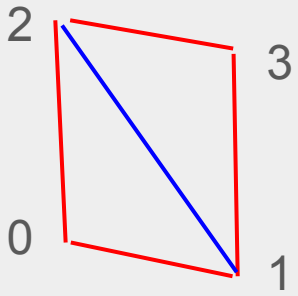
набор флагов можно из enum

uv = cX, cY, sX, sY - текстурные координаты и размер

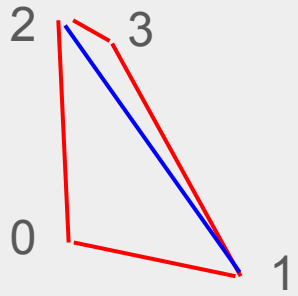


Треугольники = Квады x 2 триангла + уголки



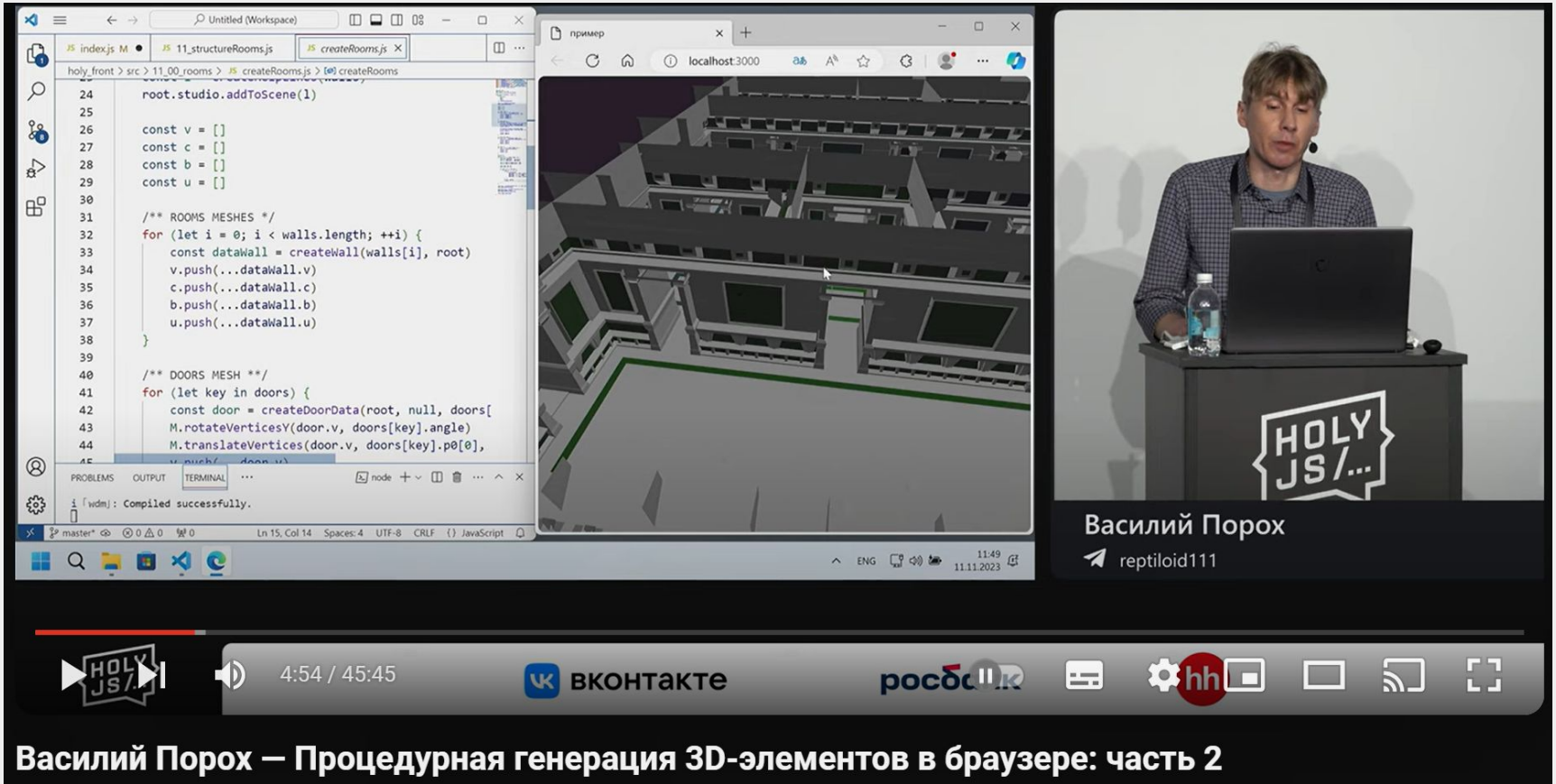


gl_VertexID



Генерация большой карты из примитивов

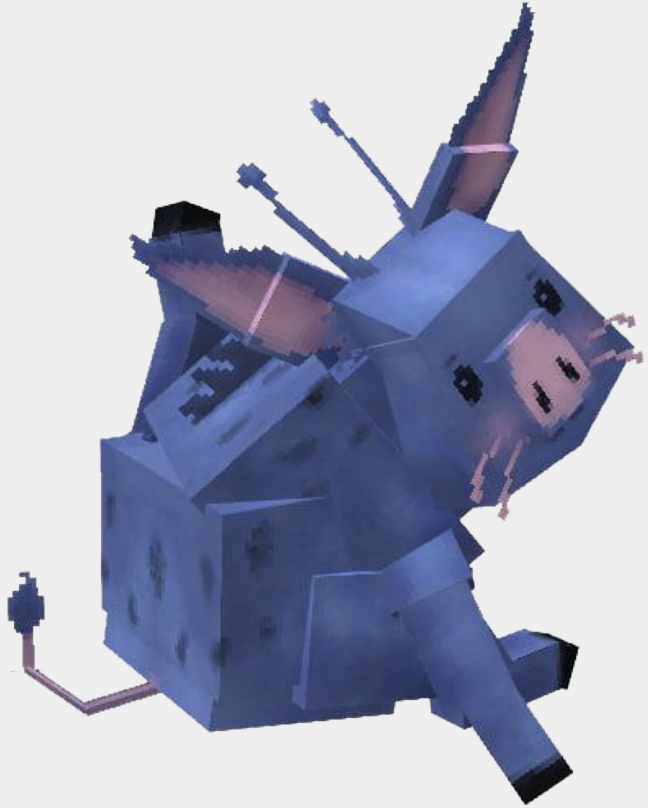
https://www.youtube.com/watch?v=qE5Y_vC_SRk

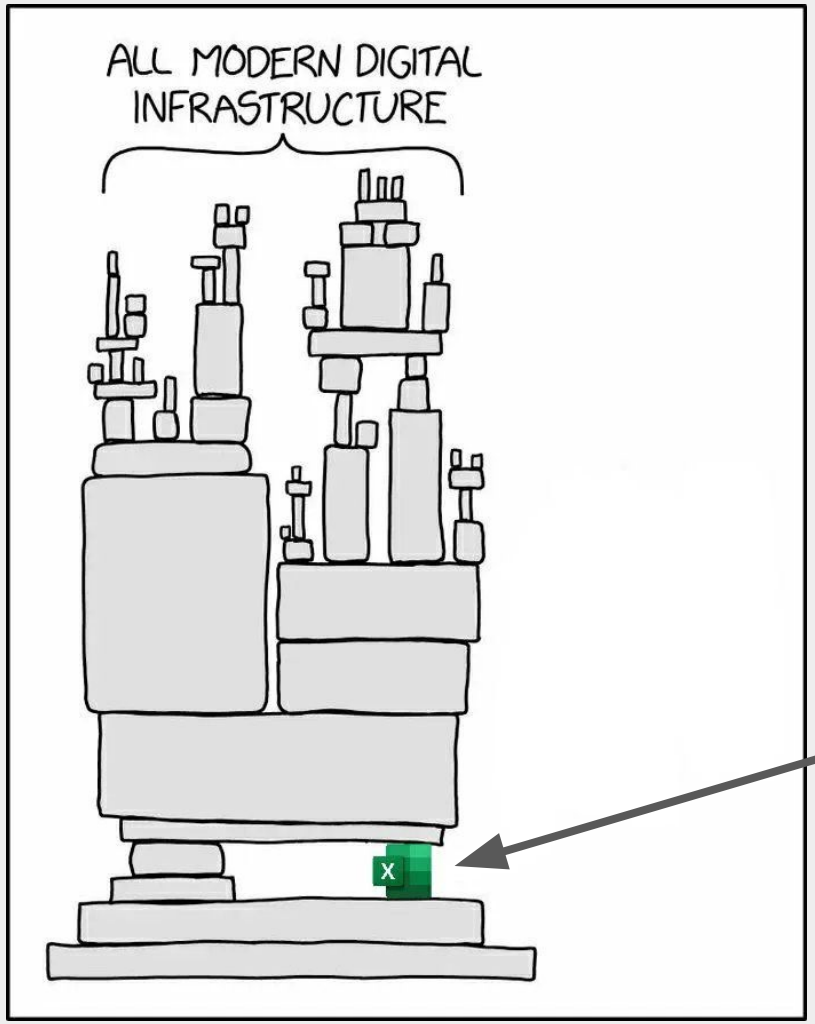


Массивы и буфера

```
export class QuadArray  {
    static strideFloats = 16

    vertices: float[] = []
    count: int = 0

    constructor(public mat: QuadMaterial) {
    }

    pushQuad(quad: Quad) {
        this.vertices.push(quad.center_x)
        this.vertices.push(quad.center_y)
        // ...
        this.count++
    }
}
```

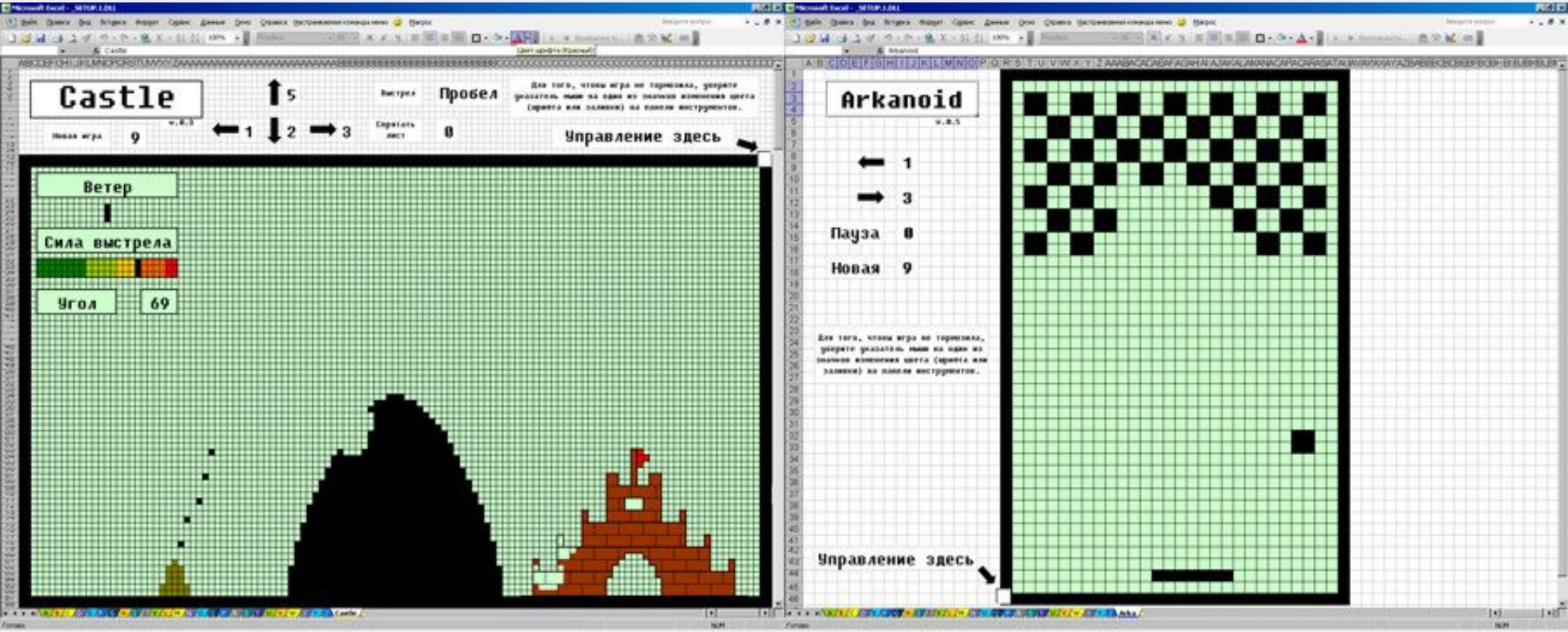
```
export class QuadBuffer  {
    data: Float32Array;
    int32_data: Int32Array;
    count: int = 0

    constructor(public mat: QuadMaterial,
                public size: int) {
        this.data = new Float32Array(
            size * QuadArray.strideFloats);
        this.int32_data = new Int32Array(this.data.buffer);
    }

    pushQuad(quad: Quad) {
        const ind = this.count * QuadArray.strideFloats
        this.data[ind] = quad.center_x
        // ...
        this.int32_data[ind + 15] = quad.flags
        this.count++
    }
}
```



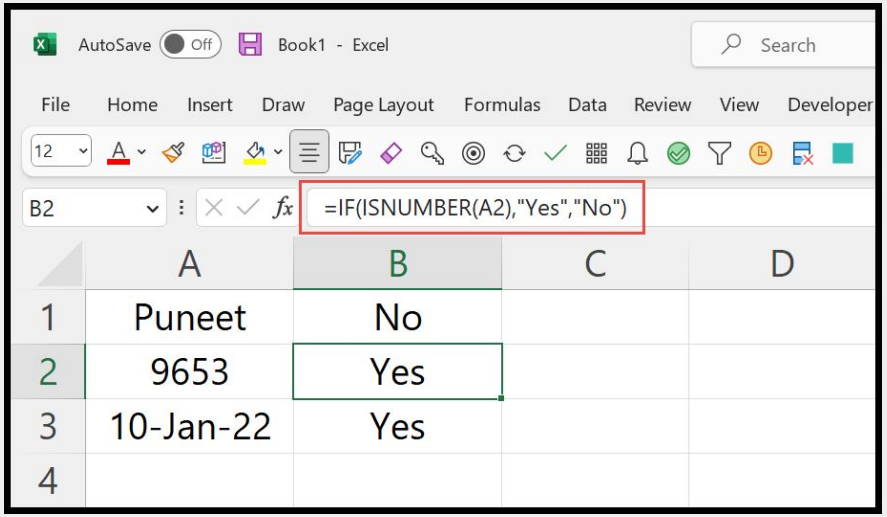

ALL MODERN DIGITAL INFRASTRUCTURE

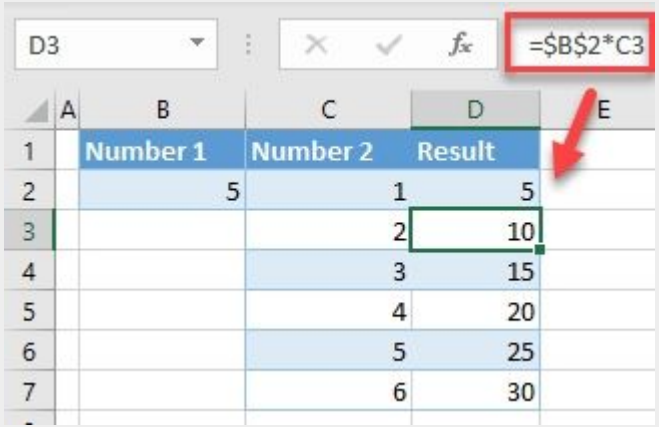
https://habr.com/ru/articles/237641/

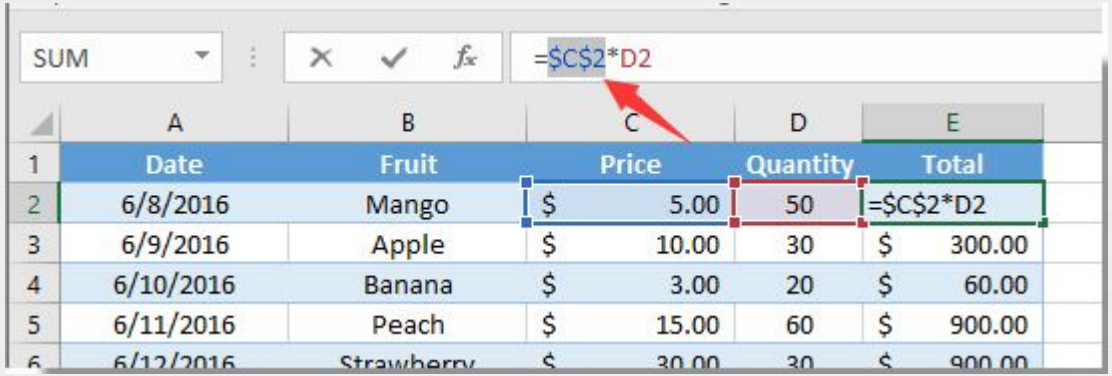
Текстура (лист в Excel) Ячейка содержит 4 32-битных числа: R G B A



- Fragment Shader
- формула, вставляется сразу в МНОГО клеток
- не может использовать текущий лист, только другие листы и текущие координаты



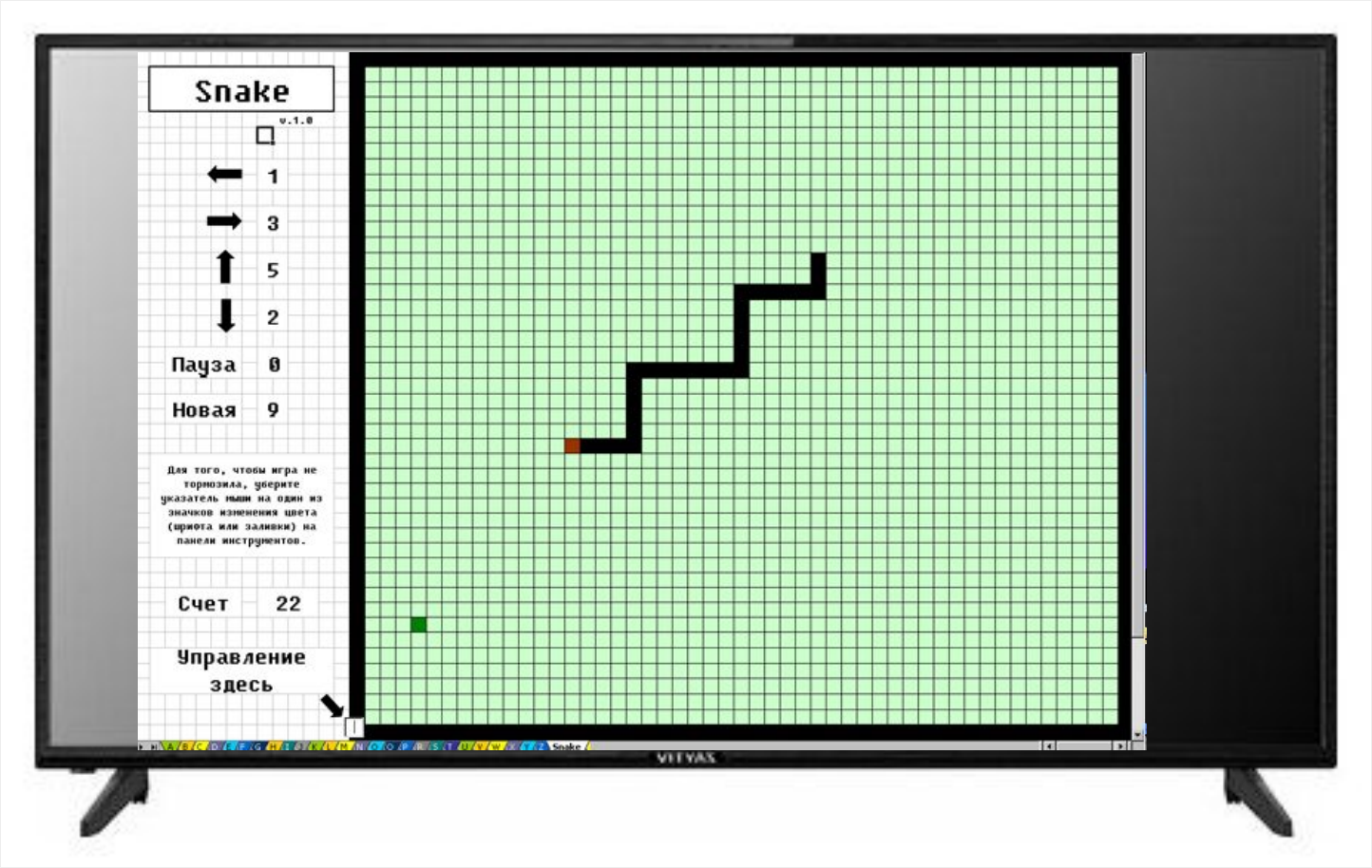




Vertex Shader (видеомагнитофон)

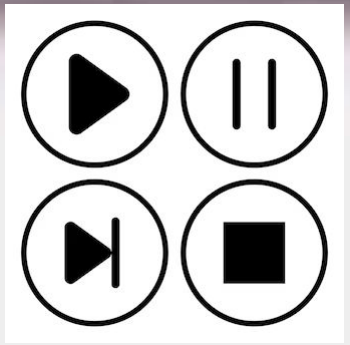
Vertex Buffer (Attribute Buffer)

Framebuffer



Vertex Shader (видеомагнитофон)

Vertex Buffer (Attribute Buffer)



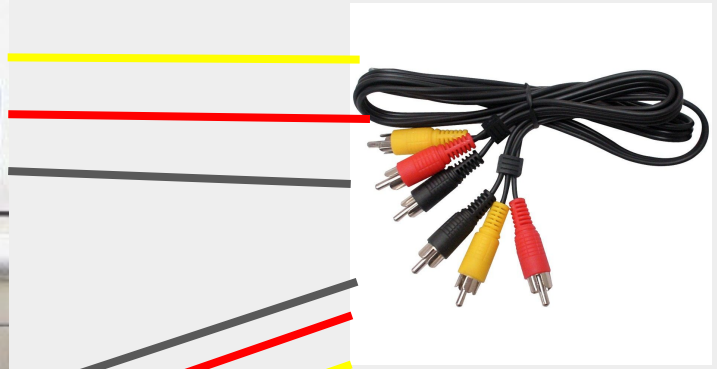
DrawCall

Крутая ссылка

Varyings (связь между vertex и fragment) → WebGL State и Uniforms

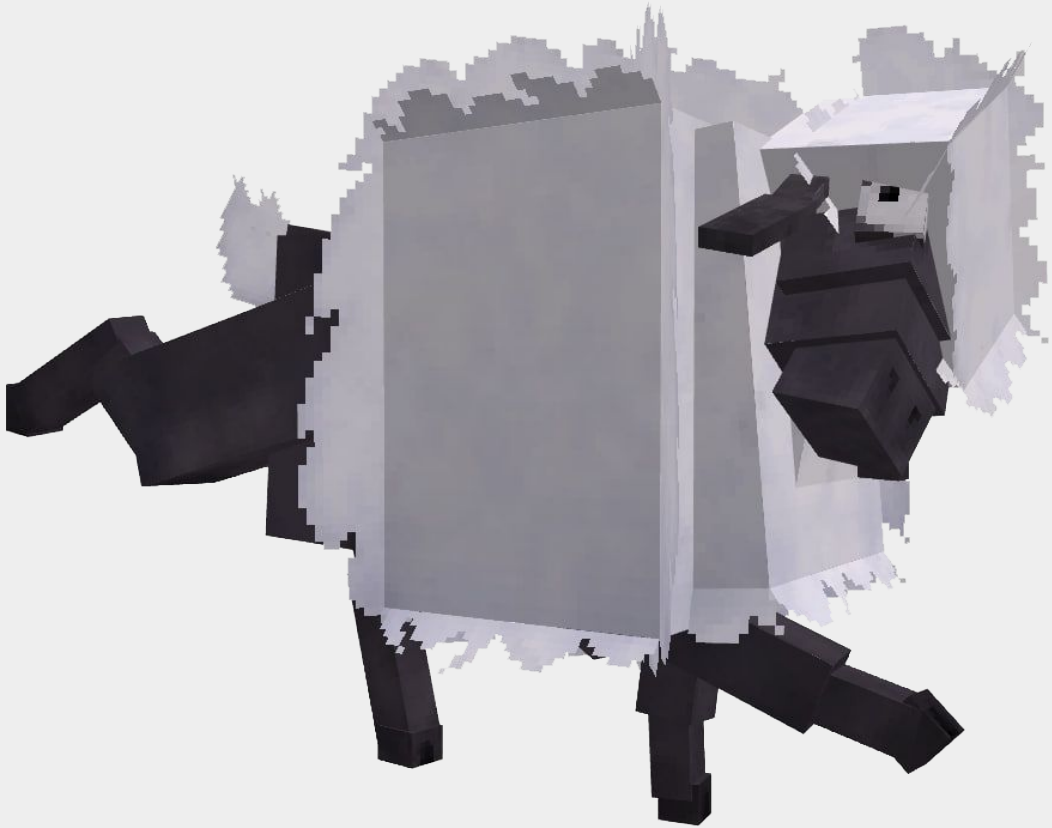
Только когда видик ещё не запущен!

Transform Feedback: из буфера в буфер, копируем Varyings



Обработка данных, Текстура VS Буфер

Проблема	Буфер 	Текстура 
Доступ из шейдера	Поля с именами, vertexAttribPointer	texelFetch(x, y) даёт 4 флота R G B A
соседи?	программа защиты свидетелей	Всегда можно прочитать (random-access)
Обработка а-ля Array.map()	Vertex Shader, Transform Feedback	Fragment Shader, рисуем квадрат в другую текстуру
Ограничение на запись	Пару десятков vec4 выдержит	Только один vec4, или drawBuffers пишем в одну клетку на разных текстурах (листиках в Excel)



Статика и динамика

Draw-calls = Quads x чанки + всякая живность




Тут текстура для чанка

Тут чуток шейдера

Статика

Примитивы = X + динамика

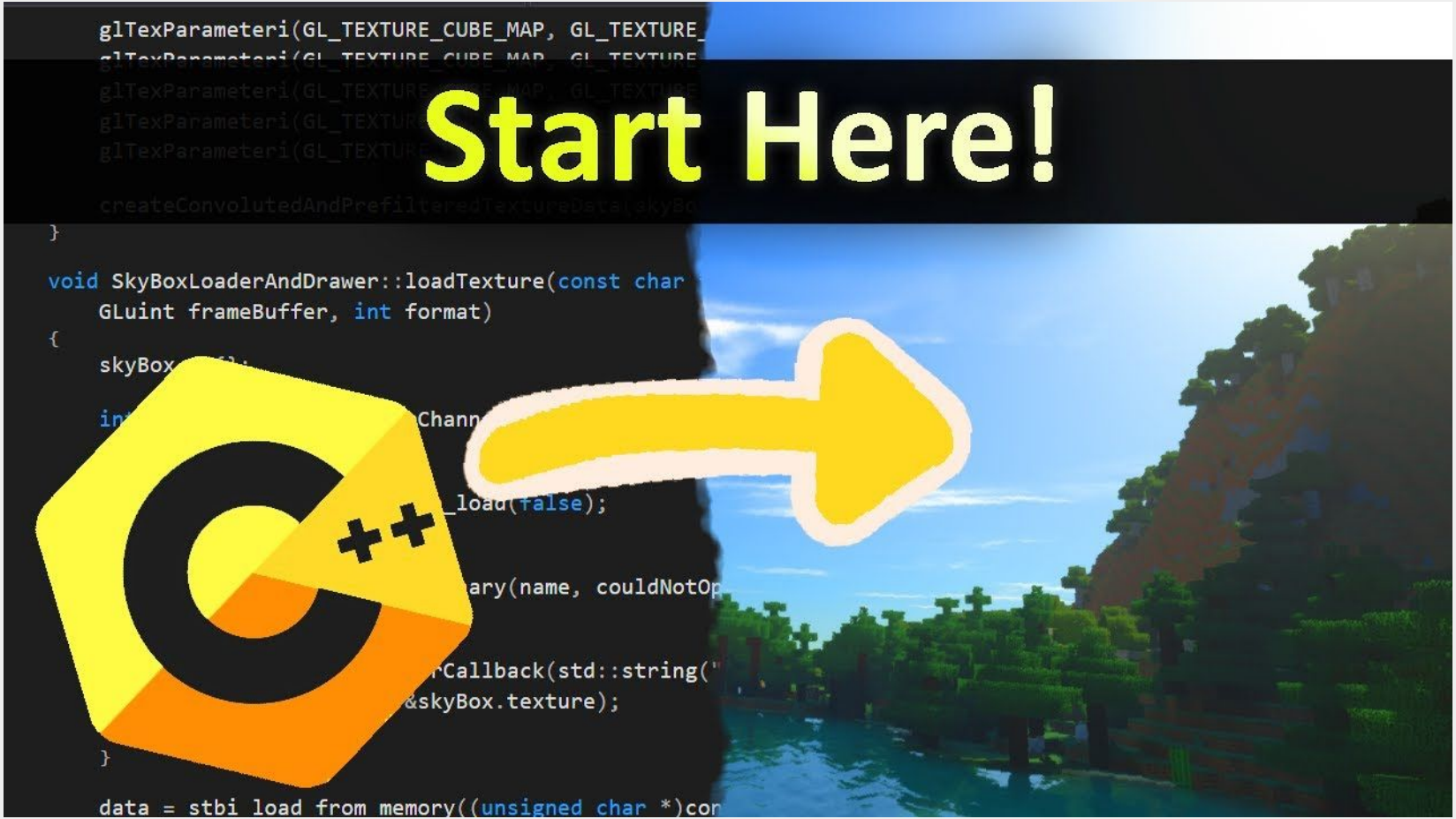
Инстанс

Хранение

- Буфер: воспользуемся видеопрокатом!
  gl.STATIC vs gl.DYNAMIC

- Текстура: просто Float32 RGBA

Low Level Game Dev youtube



Ограничения

1. Нельзя писать в определенное место памяти!

```
out_buffer = in_buffer.map((attrs) =>
    { shader code; return value; })
```

2. Во время обработки нельзя прочитать результат расчёта других объектов.

Оптимизация моделей: хранение костей в текстуре

Кости!

Квады для батчера (1) = X + квады без костей

Текстура костей

Квады + № кости (2)

Шейдер костей

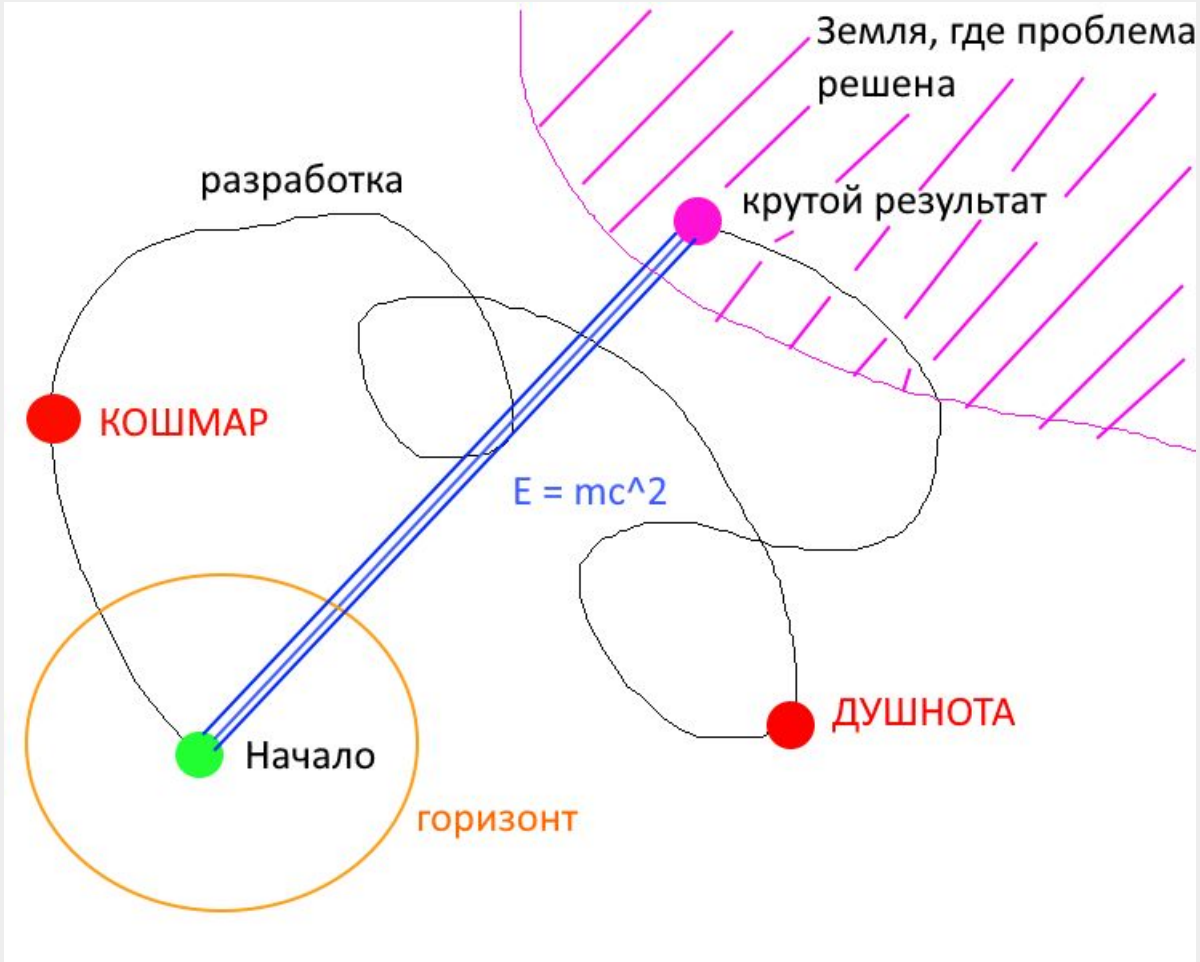
Инстансинг на коленке: шаринг анимаций

Алгоритм	Традиционная проблема WebGL-движков
Квады вместо треугольников	Объем квада теперь от 16 floats (32-bit), буфера быстрее грузятся и рассчитываются, инфы влезает много
Батчинг по материалам	bindTexture, смена GL-стейта
Менеджер памяти	bindBuffer
Вынос статики чанков в WebWorker	вычисление вершинок геометрии на процессоре
Вынос костей в текстуру	skinning mesh перенос вычисления вершин на GPU

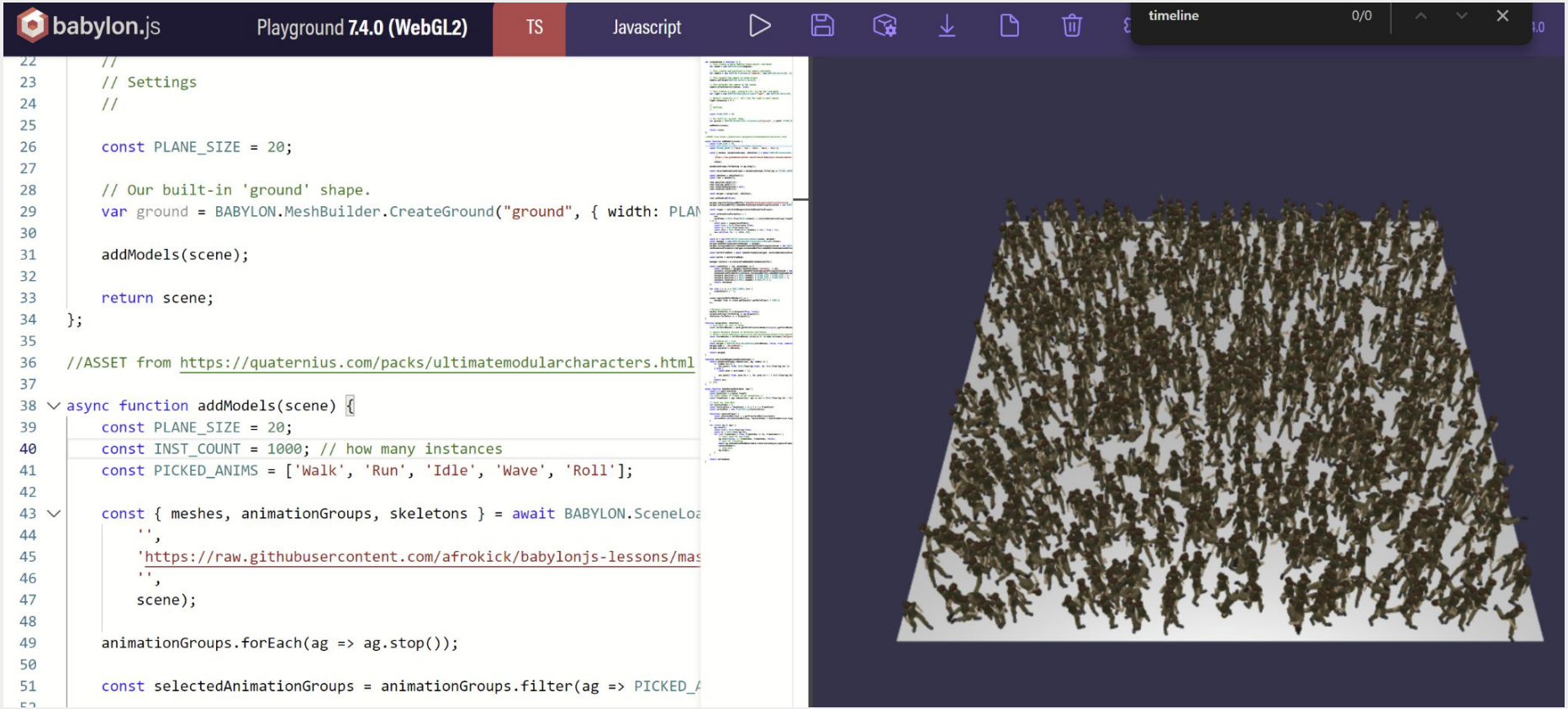

WEBGL
Я
ШЕЙДЕРА


```
updateTransform(): void
{
    if (this.sortableChildren && this.sortDirty)
    {
        this.sortChildren();
    }

    this._boundsID++;

    this.transform.updateTransform(this.parent.transform);

    // TODO: check render flags, how to process stuff here
    this.worldAlpha = this.alpha * this.parent.worldAlpha;

    for (let i = 0, j = this.children.length; i < j; ++i)
    {
        const child = this.children[i];

        if (child.visible)
        {
            child.updateTransform();
        }
    }
}
```

Таков путь


Земля, где проблема решена
разработка
крутой результат
КОШМАР
E = mc^2
ДУШНОТА
Начало
горизонт

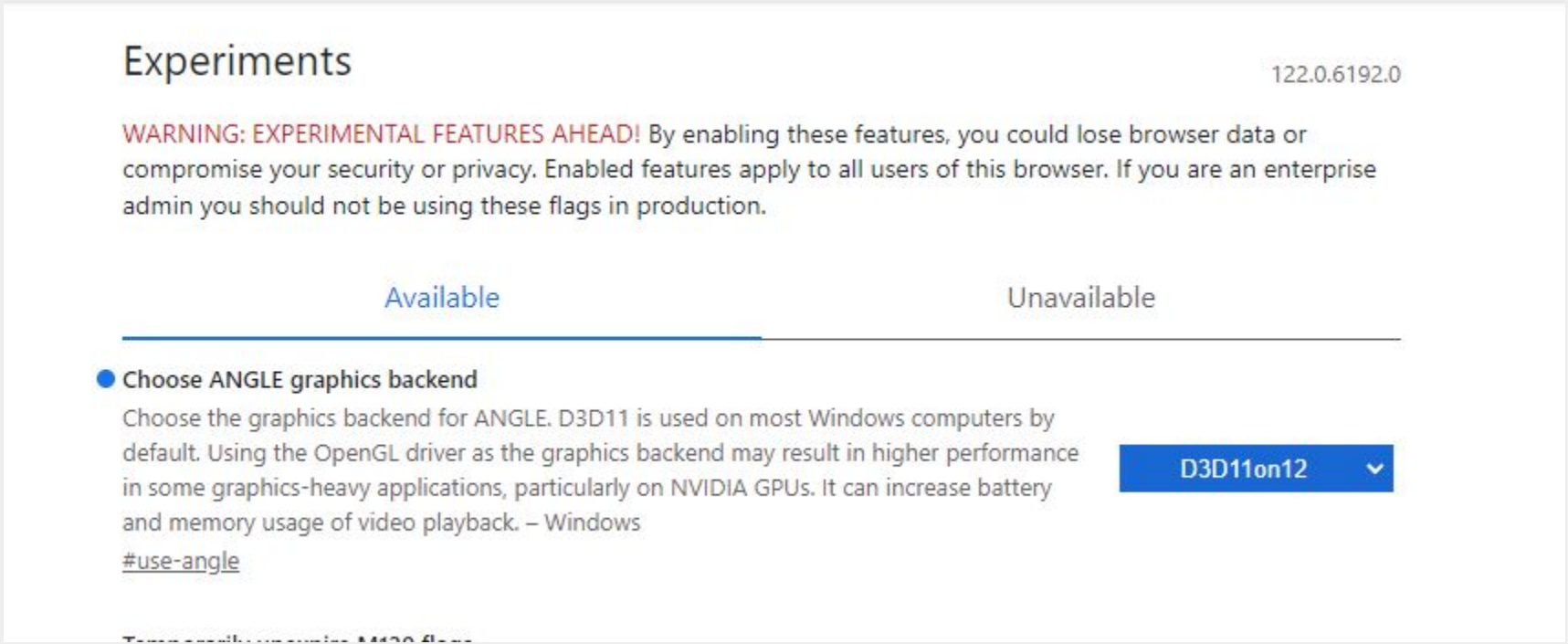
https://playground.babylonjs.com/#3NIXCL%2315



```
ivec2 coord = ivec2(gl_FragCoord.xy);
int num = coord.x & 3;
coord.x = coord.x & ~3;

vec4 row_0 = texelFetch(matrix_sample, coord, 0);
vec4 row_1 = texelFetch(matrix_sample, ivec2(coord.x + 1, coord.y), 0);
vec4 row_2 = texelFetch(matrix_sample, ivec2(coord.x + 2, coord.y), 0);
float parent_id = texelFetch(matrix_sample, ivec2(coord.x + 3, coord.y), 0).x - 1.0;

out_row = vec4(0., 0., 0., 0.);

if (parent_id < 0.0) {
    // no parent
    if (num == 0) {
        out_row = row_0;
    } else if (num == 1) {
        out_row = row_1;
    } else if (num == 2) {
        out_row = row_2;
    } else if (num == 3) {
        out_row = vec4(0., 0., 0., 0.);
    }
    return;
}
```

```
if (parent_id >= REL_SHIFT) {
    parent_id = (input_size.x * float(coord.y)) + float(coord.x) - (parent_id - REL_SHIFT) * 4.0;
} else {
    parent_id = parent_id * 4.0;
}

//TODO: maybe use <<, >> instead of *
float py = floor(parent_id * input_size.z + 0.0001);
float px = floor(parent_id - py * input_size.x + 0.0001);

vec4 parent_row = texelFetch(matrix_sample, ivec2(int(px) + num, int(py)), 0);

if (num == 3) {
    out_row = parent_row;
    if (out_row.x >= REL_SHIFT) {
        // Divide by 4 Pixels per bone, add 1.0
        out_row.x = parent_id * 0.25 + 1.0 - (out_row.x - REL_SHIFT - 1.0);
    }
    return;
}
out_row.x = parent_row.x * row_0.x + parent_row.y * row_1.x + parent_row.z * row_2.x;
out_row.y = parent_row.x * row_0.y + parent_row.y * row_1.y + parent_row.z * row_2.y;
out_row.z = parent_row.x * row_0.z + parent_row.y * row_1.z + parent_row.z * row_2.z;
out_row.w = parent_row.x * row_0.w + parent_row.y * row_1.w + parent_row.z * row_2.w + parent_row.w;
```

Отладочка на видеокарте

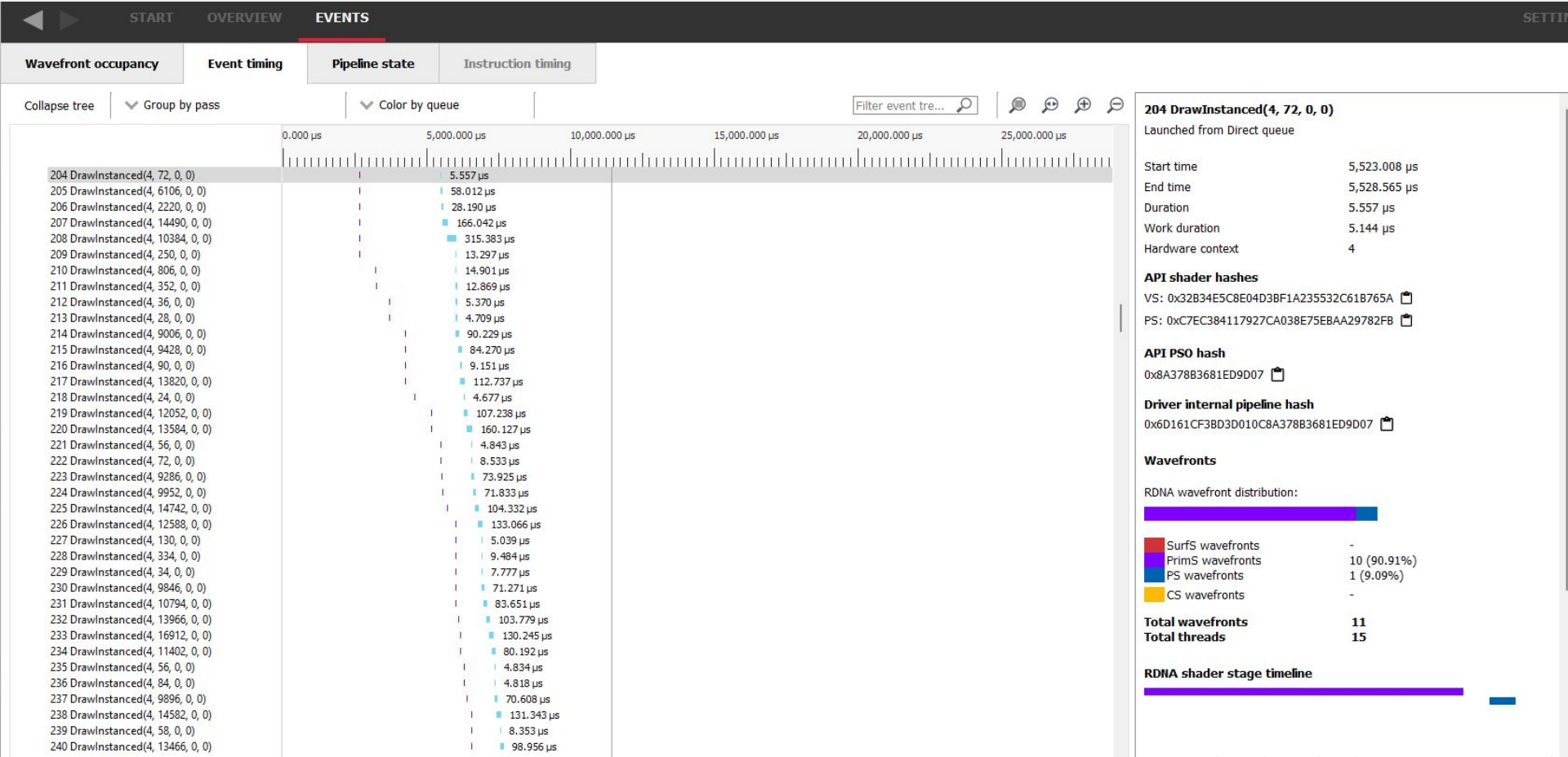


Radeon GPU Profiler

NVidia NSight

Intel Performance Graphics Analyzers

Radeon GPU Profiler



Wavefront occupancy | Event timing | Pipeline state | Instruction timing

Collapse tree | Group by pass

307 DrawInstanced(4, 508, 0, 0)
308 DrawInstanced(4, 364, 0, 0)
309 DrawInstanced(4, 328, 0, 0)
310 DrawInstanced(4, 384, 0, 0)
311 DrawInstanced(4, 64, 0, 0)
312 DrawInstanced(4, 64, 0, 0)
313 DrawInstanced(4, 152, 0, 0)
314 DrawInstanced(4, 314, 0, 0)
315 DrawInstanced(4, 276, 0, 0)
316 DrawInstanced(4, 224, 0, 0)
317 DrawInstanced(4, 230, 0, 0)
318 DrawInstanced(4, 308, 0, 0)
319 DrawInstanced(4, 168, 0, 0)
320 DrawInstanced(4, 718, 0, 0)
321 DrawInstanced(4, 464, 0, 0)
322 DrawInstanced(4, 348, 0, 0)
323 DrawInstanced(4, 16, 0, 0)
324 DrawInstanced(4, 232, 0, 0)
325 DrawInstanced(4, 1780, 0, 0)
326 DrawInstanced(4, 188, 0, 0)
327 DrawInstanced(4, 442, 0, 0)
328 DrawInstanced(4, 480, 0, 0)
329 DrawInstanced(4, 364, 0, 0)
330 DrawInstanced(4, 72, 0, 0)
331 DrawInstanced(4, 216, 0, 0)
332 DrawInstanced(4, 252, 0, 0)
333 DrawInstanced(4, 256, 0, 0)
334 DrawInstanced(4, 408, 0, 0)
335 CmdBarrierBlitSync()
336 ResourceBarrier()

Color pass 10

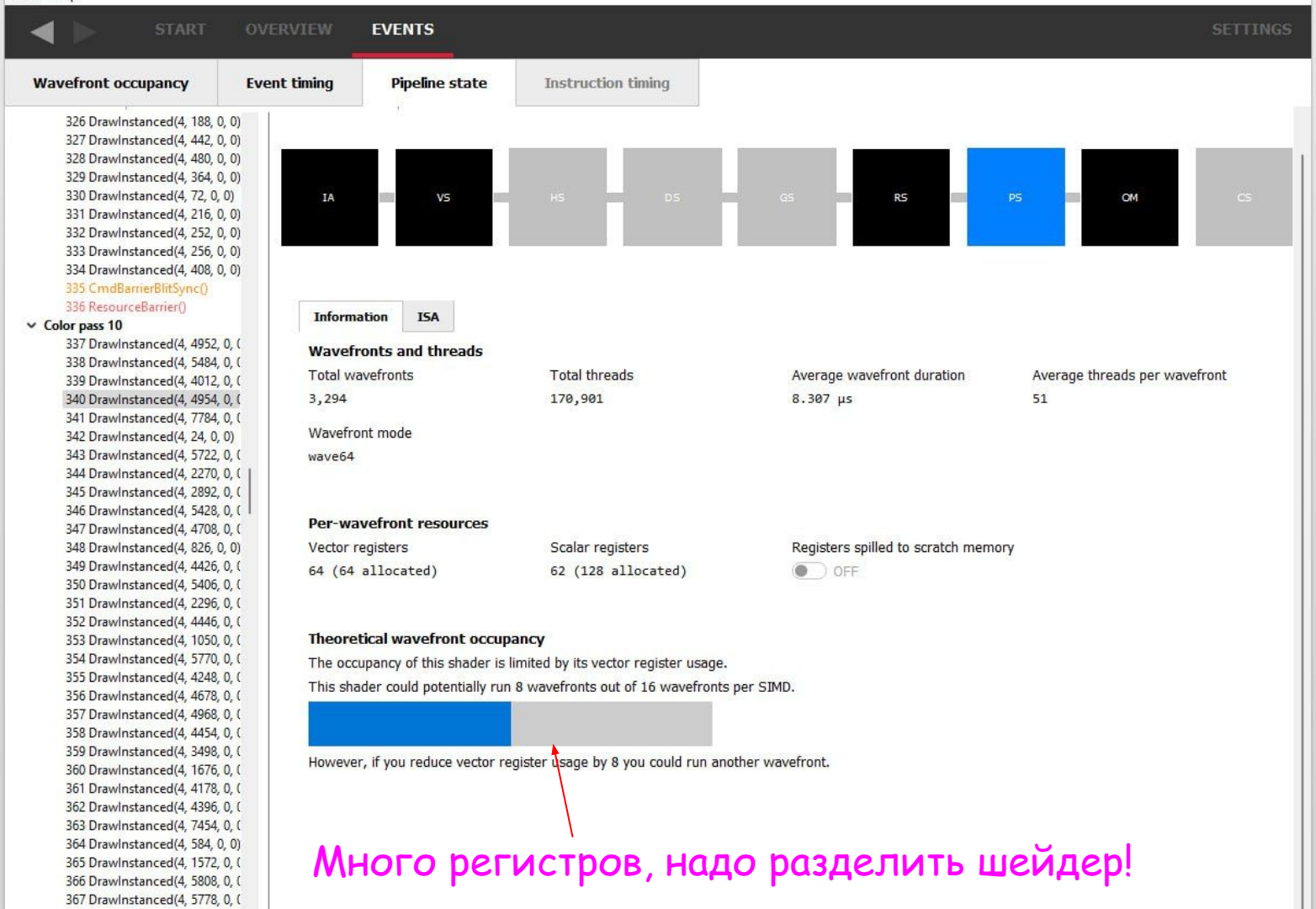
337 DrawInstanced(4, 4952, 0, 0)
338 DrawInstanced(4, 5484, 0, 0)
339 DrawInstanced(4, 4012, 0, 0)
340 DrawInstanced(4, 4954, 0, 0)
341 DrawInstanced(4, 7784, 0, 0)
342 DrawInstanced(4, 24, 0, 0)
343 DrawInstanced(4, 5722, 0, 0)
344 DrawInstanced(4, 2270, 0, 0)
345 DrawInstanced(4, 2892, 0, 0)
346 DrawInstanced(4, 5428, 0, 0)
347 DrawInstanced(4, 4708, 0, 0)
348 DrawInstanced(4, 826, 0, 0)
349 DrawInstanced(4, 4426, 0, 0)
350 DrawInstanced(4, 5406, 0, 0)

Information | ISA

Shader ISA

Viewing Options

	Opcode	Operands
3774	BBF1_66	
3775	s_mov_b64	vcc, exec
3776	s_wqm_b64	exec, vcc
3777	v_mov_b32_dpp	v2, v23 quad_perm:[0, 0, 0, 0] row_mask:0xf bank_mask:0xf bound_ctrl:1
3778	v_mov_b32_dpp	v3, v16 quad_perm:[0, 0, 0, 0] row_mask:0xf bank_mask:0xf bound_ctrl:1
3779	v_mov_b32_dpp	v8, v23 quad_perm:[0, 0, 0, 0] row_mask:0xf bank_mask:0xf bound_ctrl:1
3780	v_mov_b32_dpp	v7, v16 quad_perm:[0, 0, 0, 0] row_mask:0xf bank_mask:0xf bound_ctrl:1
3781	v_sub_f32_dpp	v2, v23, v2 quad_perm:[2, 2, 2, 2] row_mask:0xf bank_mask:0xf bound_ctrl:1
3782	v_sub_f32_dpp	v3, v16, v3 quad_perm:[2, 2, 2, 2] row_mask:0xf bank_mask:0xf bound_ctrl:1
3783	v_sub_f32_dpp	v8, v23, v8 quad_perm:[1, 1, 1, 1] row_mask:0xf bank_mask:0xf bound_ctrl:1
3784	v_sub_f32_dpp	v9, v16, v7 quad_perm:[1, 1, 1, 1] row_mask:0xf bank_mask:0xf bound_ctrl:1
3785	s_mov_b64	exec, vcc
3786	s_load_dwordx8	s[8:15], s[6:7], null
3787	s_load_dwordx4	s[24:27], s[4:5], null
3788	s_waitcnt	lgkmcnt(0)
3789	s_andn2_b32	s14, s14, 0x100000
3790	image_sample	v[10:12], [v23, v16], s[8:15], s[24:27] dmask:0x7 dim:SQ_RSRC_IMG_2D
3791	v_add_f32_e64	v2, |v2|, |v8|
3792	v_add_f32_e64	v3, |v3|, |v9|
3793	v_mul_f32_e32	v7, 0x42c80000, v2
3794	v_mul_f32_e32	v3, 0x42c80000, v3
3795	v_mul_f32_e32	v2, v7, v7
3796	v_fmac_f32_e32	v2, v3, v3
3797	v_sqrt_f32_e32	v2, v2
3798	v_fmaak_f32	v8, -0.5, v2, 0x3dcccce
3799	v_fmaak_f32	v3, 0.5, v2, 0x3dcccce
3800	v_sub_f32_e32	v2, v3, v8
3801	v_sub_f32_e32	v3, 0x3e99999a, v8


START
OVERVIEW
EVENTS
SETTINGS
Wavefront occupancy
Event timing
Pipeline state
Instruction timing
326 DrawInstanced(4, 188, 0, 0)
327 DrawInstanced(4, 442, 0, 0)
328 DrawInstanced(4, 480, 0, 0)
329 DrawInstanced(4, 364, 0, 0)
330 DrawInstanced(4, 72, 0, 0)
331 DrawInstanced(4, 216, 0, 0)
332 DrawInstanced(4, 252, 0, 0)
333 DrawInstanced(4, 256, 0, 0)
334 DrawInstanced(4, 408, 0, 0)
335 CmdBarrierBlitSync()
336 ResourceBarrier()
Color pass 10
337 DrawInstanced(4, 4952, 0, (
338 DrawInstanced(4, 5484, 0, (
339 DrawInstanced(4, 4012, 0, (
340 DrawInstanced(4, 4954, 0, (
341 DrawInstanced(4, 7784, 0, (
342 DrawInstanced(4, 24, 0, 0)
343 DrawInstanced(4, 5722, 0, (
344 DrawInstanced(4, 2270, 0, (
345 DrawInstanced(4, 2892, 0, (
346 DrawInstanced(4, 5428, 0, (
347 DrawInstanced(4, 4708, 0, (
348 DrawInstanced(4, 826, 0, 0)
349 DrawInstanced(4, 4426, 0, (
350 DrawInstanced(4, 5406, 0, (
351 DrawInstanced(4, 2296, 0, (
352 DrawInstanced(4, 4446, 0, (
353 DrawInstanced(4, 1050, 0, (
354 DrawInstanced(4, 5770, 0, (
355 DrawInstanced(4, 4248, 0, (
356 DrawInstanced(4, 4678, 0, (
357 DrawInstanced(4, 4968, 0, (
358 DrawInstanced(4, 4454, 0, (
359 DrawInstanced(4, 3498, 0, (
360 DrawInstanced(4, 1676, 0, (
361 DrawInstanced(4, 4178, 0, (
362 DrawInstanced(4, 4396, 0, (
363 DrawInstanced(4, 7454, 0, (
364 DrawInstanced(4, 584, 0, 0)
365 DrawInstanced(4, 1572, 0, (
366 DrawInstanced(4, 5808, 0, (
367 DrawInstanced(4, 5778, 0, (
IA
VS
HS
DS
GS
RS
PS
OM
CS
Information
ISA
Wavefronts and threads
Total wavefronts
3,294
Total threads
170,901
Average wavefront duration
8.307 µs
Average threads per wavefront
51
Wavefront mode
wave64
Per-wavefront resources
Vector registers
64 (64 allocated)
Scalar registers
62 (128 allocated)
Registers spilled to scratch memory
OFF
Theoretical wavefront occupancy
The occupancy of this shader is limited by its vector register usage.
This shader could potentially run 8 wavefronts out of 16 wavefronts per SIMD.
However, if you reduce vector register usage by 8 you could run another wavefront.
Много регистров, надо разделить шейдер!

Анимация

Текущий position, rotation, scale считается по TimeLine анимации:

а именно, по каждому "каналу" текущее значение считается по двум ключевым кадрам (KeyFrame)

Расчёт костей

Есть сложный объект, который для рендеринга каждый раз полностью проходится по дереву

Для каждой кости, рекурсивно:

- Композиция: рассчитать матрицу по текущей position, rotate, scale
- Умножить матрицу на родительскую
- Модификаторы: Callbacks, отрисовка доп элементов (брони, блоков в руке), вычисление текущей текстуры
- Вызвать прорисовку кубиков привязанных к этой кости

```
drawBuffered(batch: MeshPosInstance,
             parent_matrix: imat4,
             bone_matrix: float[] = null,
             mesh_parts: MeshPartCollection = null) {
    this.applyAnimations(
        this.mesh.model_groups.bone_local_buffer)
    this.world_matrix = mat43mul(this._mx,
        parent_matrix, this.local_matrix);
    if(this.draw_group_callback) {
        this.draw_group_callback(this, batch.batcher,
            batch.pos, parent_matrix)
    }
    for (let i = 0; i < this.children.length; i++) {
        const child = this.children[i];
        if (child.is_hidden) {
            continue
        }
        child.drawBuffered(batch, this.world_matrix,
            bone_matrix, mesh_parts)
    }
    if(this.need_build_geom) {
        // запихнуть кубических детей в geometries
    }
    batch.addParts(this.geometries,
        this.world_matrix, this.local_bone_id)
}
```

Инстансинг на коленке: шаринг анимаций

Инстанс!

Квады (2) = X + обычные (2)

Квады + № инстанса (3)

Текстура инстанса

Шейдер инстанса

FOR FOR FOR

TRUE INSTANCING

Transform feedback

Шейдер, в основном просто читает текстуры

Квады для батчера

=

Меши
Стат квады
Дин квады
Кости

X

№ меша
№ квада
№ кости

Когда кода осталось мало

Выигрышь перфа = Вызов расчёта на GPU вместо CPU / Кадры (FPS) X - перерасчет

- Все серьёзные операции перенесены на GPU
- Рекурсивный проход пускается только когда менялась структура
- Каждый кадр меняются какие-то индексы в уже подготовленном массиве для элемента (mesh_id, quad_id, bone_id)