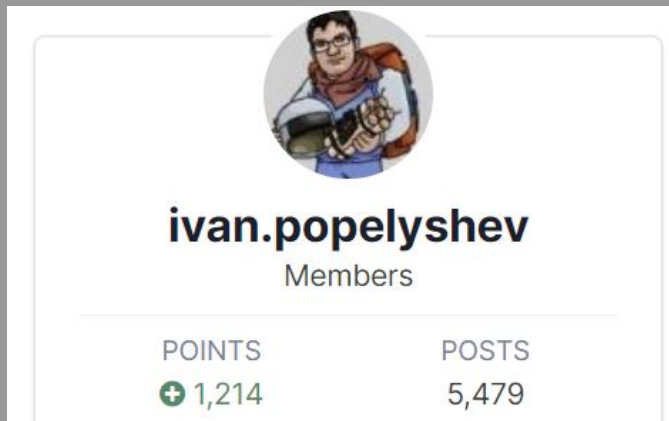




<https://t.me/hackerham>

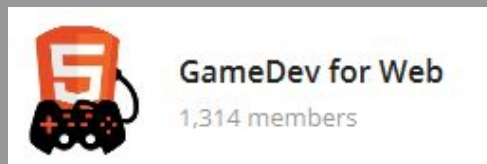


A white profile card for a Telegram channel. At the top left is a circular profile picture of a man with glasses and a backpack. Below it, the name 'ivan.popelyshev' is written in bold black text, followed by 'Members' in a smaller font. At the bottom, there are two columns of statistics: 'POINTS' with a green plus icon and '1,214', and 'POSTS' with '5,479'.

POINTS	POSTS
+ 1,214	5,479



<https://t.me/gamedevforweb>



A white profile card for a Telegram channel. On the left is an icon of a game controller with a red 'G' on it. To the right, the name 'GameDev for Web' is written in bold black text, followed by '1,314 members' in a smaller font.



Цель

Я делаю крутую штуку и хочу про это рассказать

Тема слишком сложная и перенасыщена кодом

Хочу объяснить понятно, и пропускаю весь кошмар

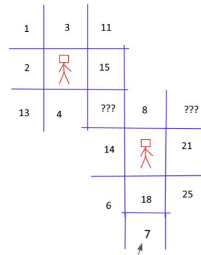


В предыдущих частях

<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



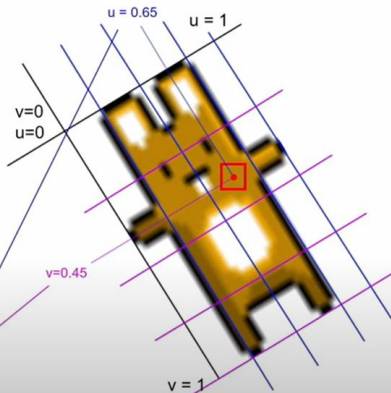
<https://www.youtube.com/watch?v=w8wQcsZEvpw>



<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 нахалыву посчитают значения U, V для центра пикселя!
4. Цвет кролика у нас один на всех вершинах - он не изменится при интерполяции



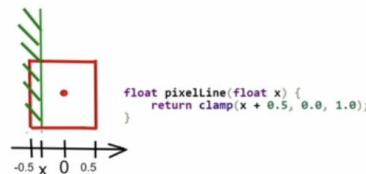
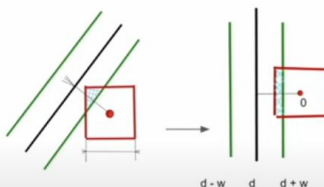
А вот тут надо дописать, как после интерполяции рассчитать цвет пикселя

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

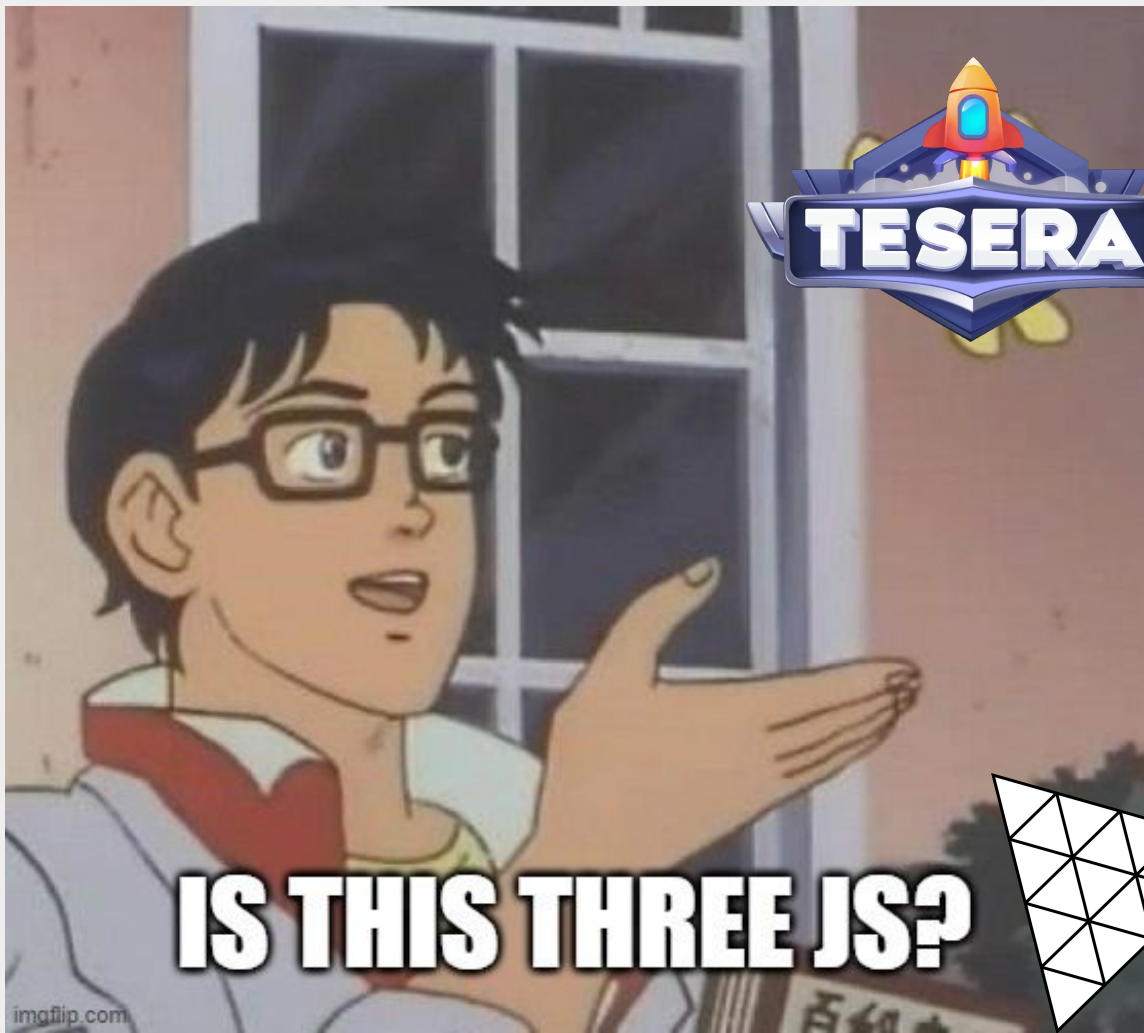
доклад с намёком на прикладную математику



Внимание, вопрос:

Что такое текстура?





Blockbench
A low-poly 3D model editor



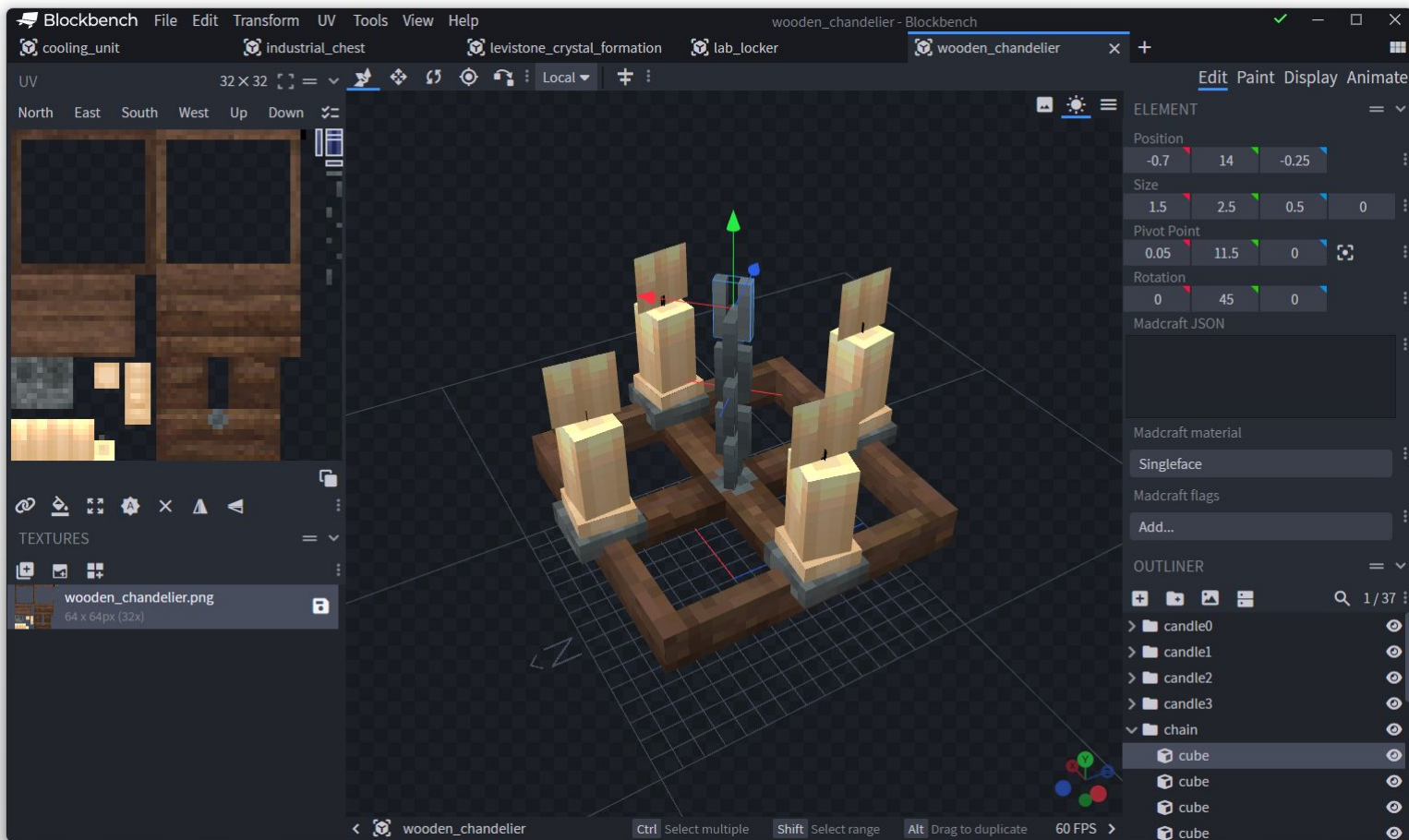
Pixi^{JS}

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

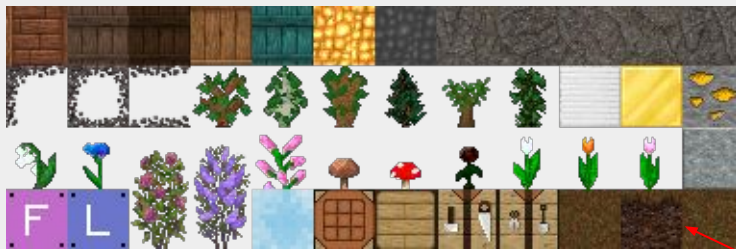


Quad: основной примитив графики

```
{  
    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**

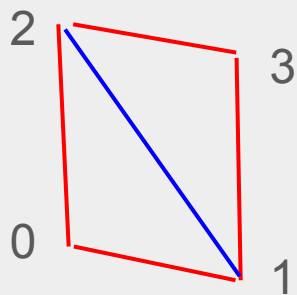
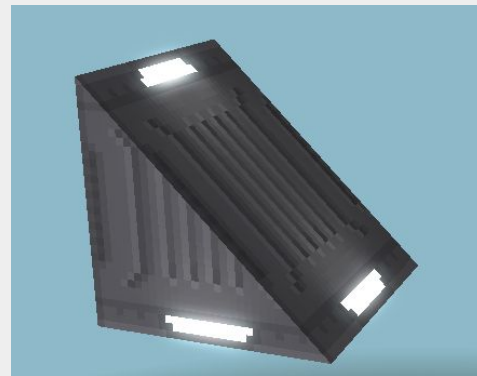


uv = cX, cY, sX, sY - текстурные координаты и размер

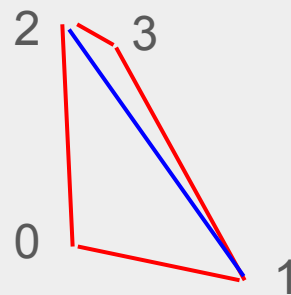


position - центр

Треугольники = Квады x 2 триангла + уголки



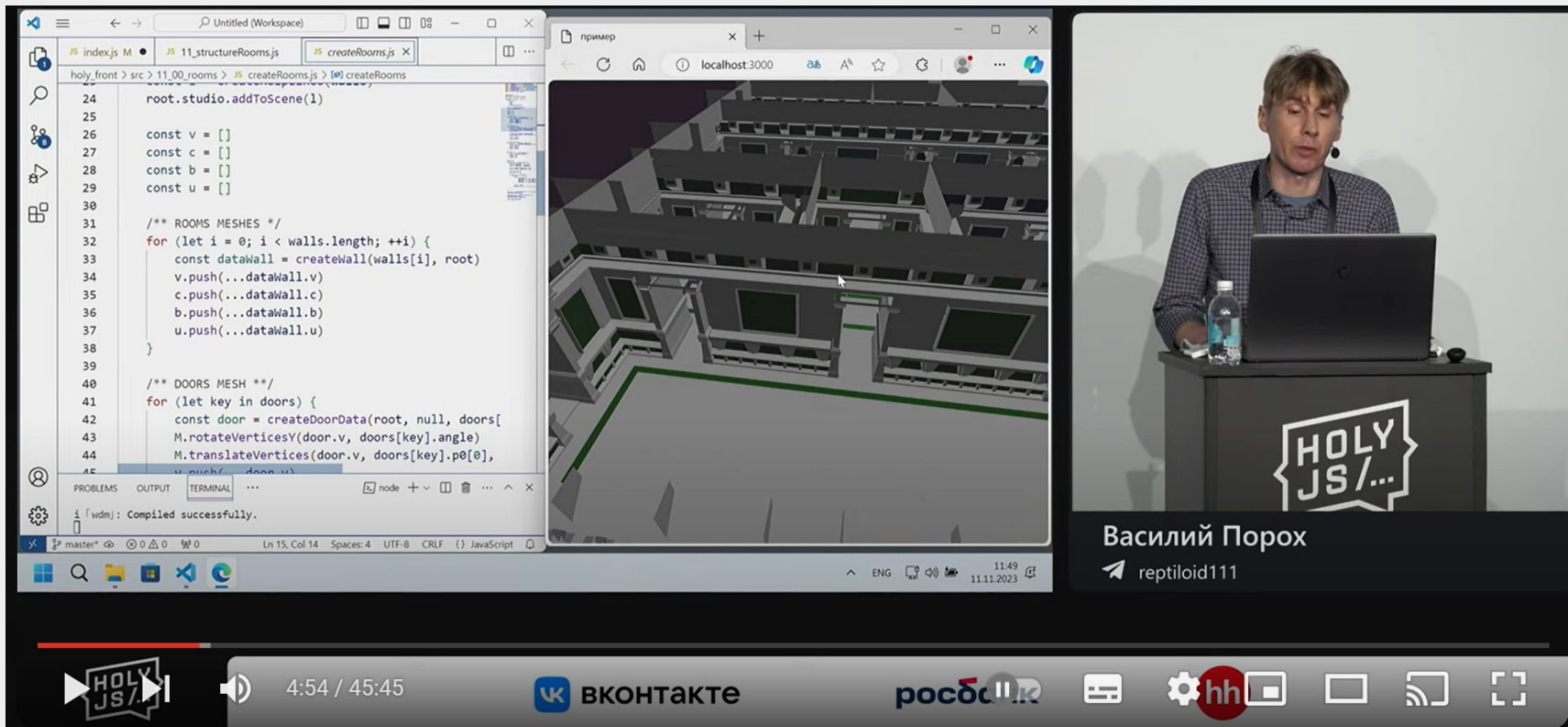
gl_VertexID





Генерация большой карты из примитивов

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



```
24 root.studio.addToScene(1)
25
26 const v = []
27 const c = []
28 const b = []
29 const u = []
30
31 /** ROOMS MESHES */
32 for (let i = 0; i < walls.length; ++i) {
33   const dataWall = createWall(walls[i], root)
34   v.push(...dataWall.v)
35   c.push(...dataWall.c)
36   b.push(...dataWall.b)
37   u.push(...dataWall.u)
38 }
39
40 /** DOORS MESH */
41 for (let key in doors) {
42   const door = createDoorData(root, null, doors[
43     M.rotateVerticesY(door.v, doors[key].angle)
44     M.translateVertices(door.v, doors[key].p[0]),
45     v.push(...door.u)
```

Василий Порох
reptiloid11

4:54 / 45:45

ВКонтакте | росбизнес | hh

Массивы и буфера

```
export class QuadArray {
  static strideFloats = 16

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

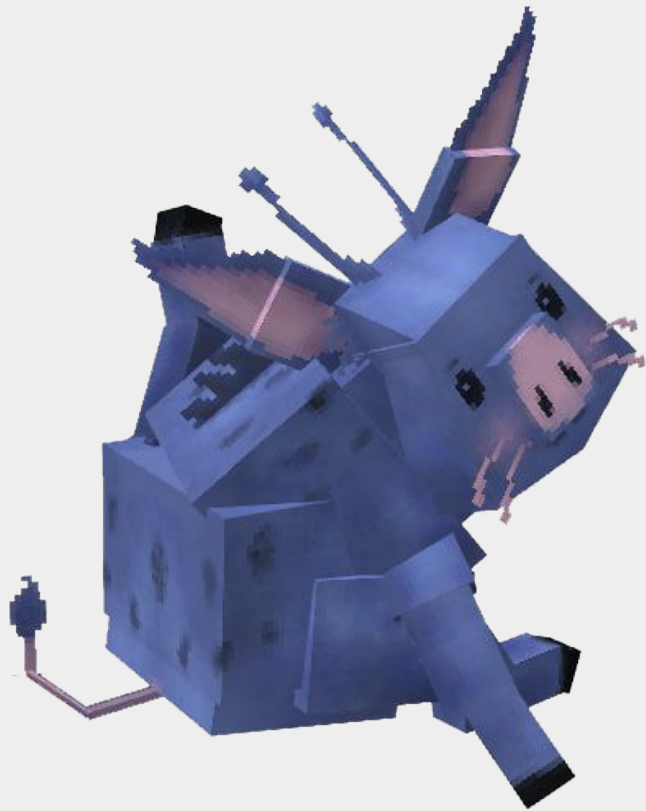
  constructor(public mat: QuadMaterial) {
  }

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

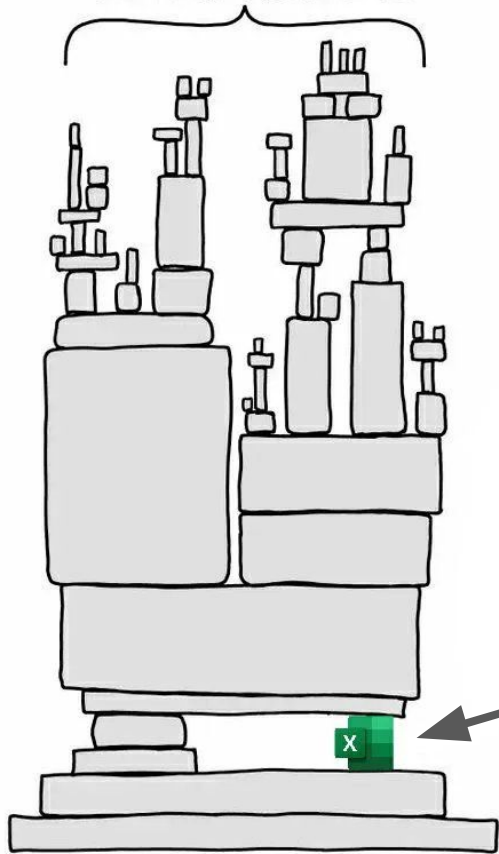
```
export class QuadBuffer {
  data: Float32Array;
  int32_data: Int32Array;
  count: int = 0

  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++
  }
}
```

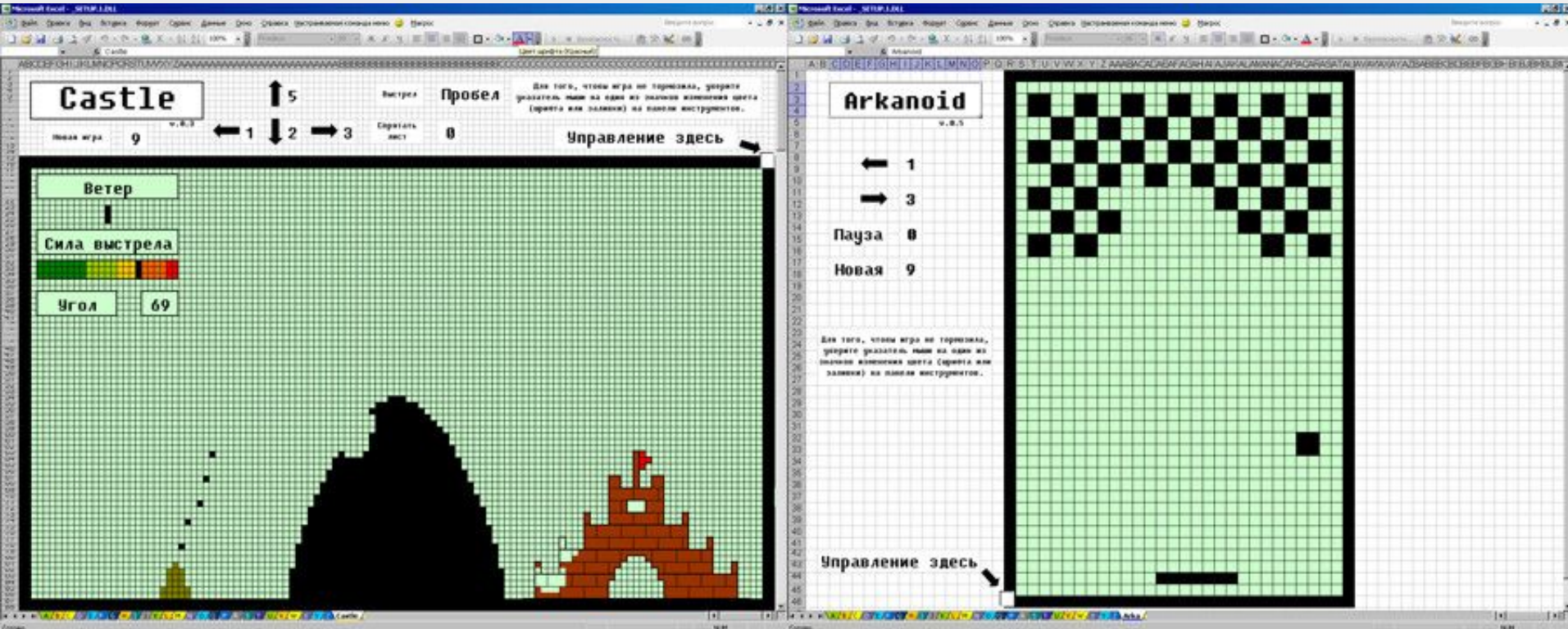
ALL MODERN DIGITAL
INFRASTRUCTURE



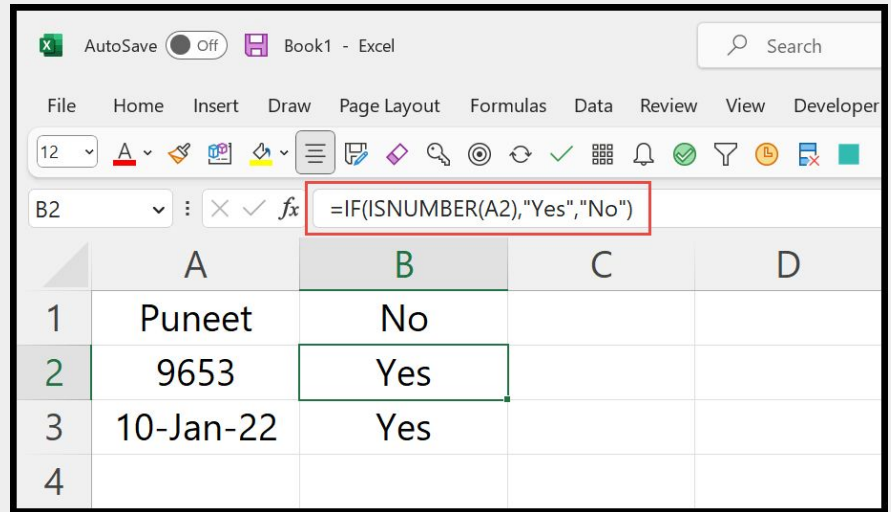
<https://habr.com/ru/articles/237641/>

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



The screenshot shows the Excel interface with the formula bar containing `=B2*C3`. The grid below shows the following data:

	A	B	C	D	E
1		Number 1	Number 2	Result	
2		5	1	5	
3			2	10	
4			3	15	
5			4	20	
6			5	25	
7			6	30	

The screenshot shows the Excel interface with the formula bar containing `=C2*D2`. The grid below shows the following data:

	A	B	C	D	E	
1		Date	Fruit	Price	Quantity	Total
2		6/8/2016	Mango	\$ 5.00	50	=\$C\$2*D2
3		6/9/2016	Apple	\$ 10.00	30	\$ 300.00
4		6/10/2016	Banana	\$ 3.00	20	\$ 60.00
5		6/11/2016	Peach	\$ 15.00	60	\$ 900.00
6		6/12/2016	Strawberry	\$ 30.00	30	\$ 900.00

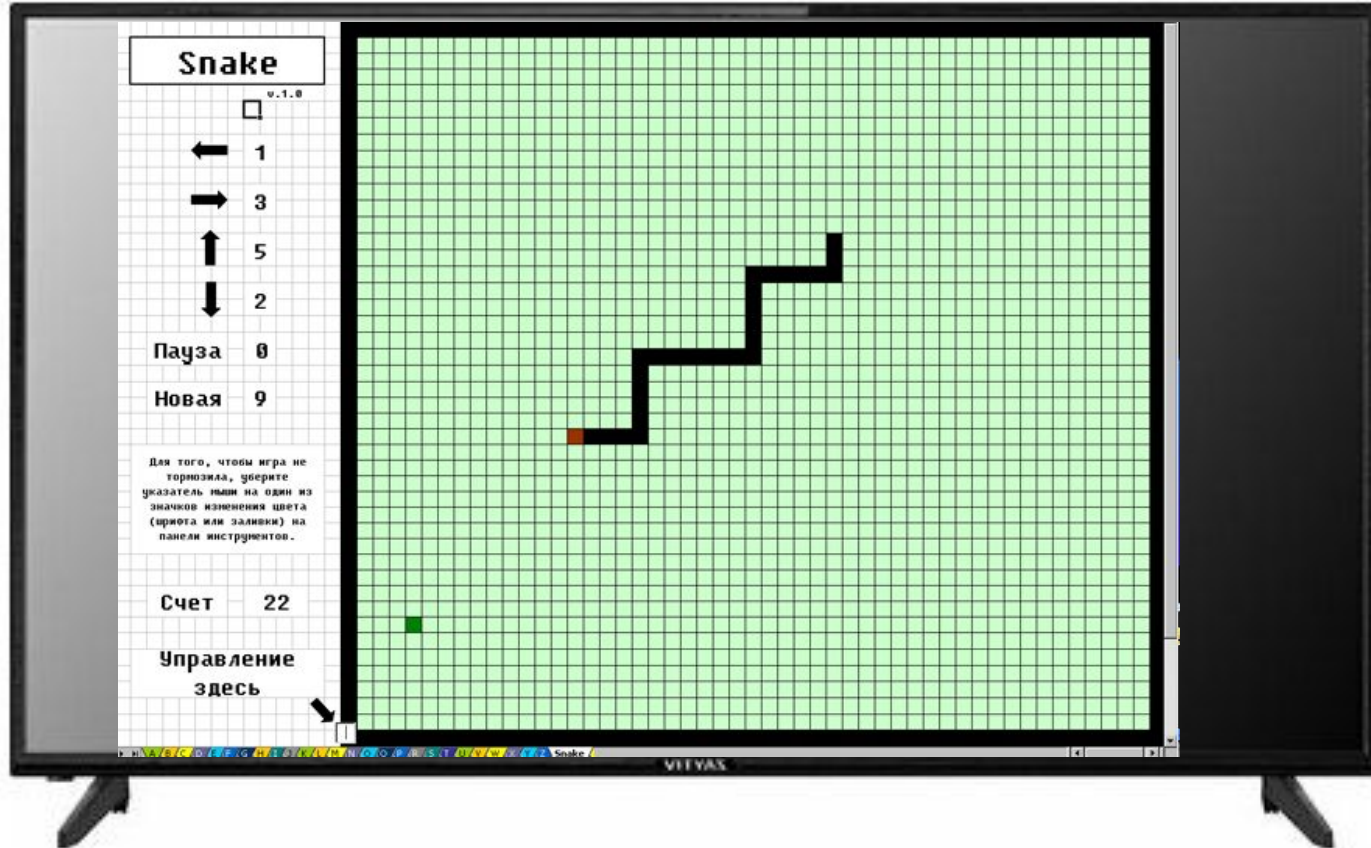
[Vertex Shader](#) (видеомагнитофон)



[Vertex Buffer](#) (Attribute Buffer)



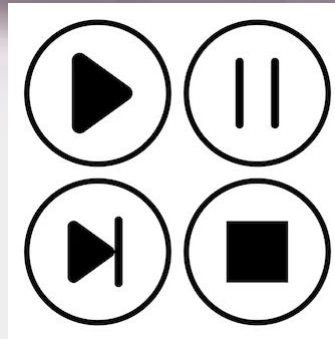
Framebuffer



Vertex Shader (видеомагнитофон)



Vertex Buffer (Attribute Buffer)



DrawCall

Крутая ссылка

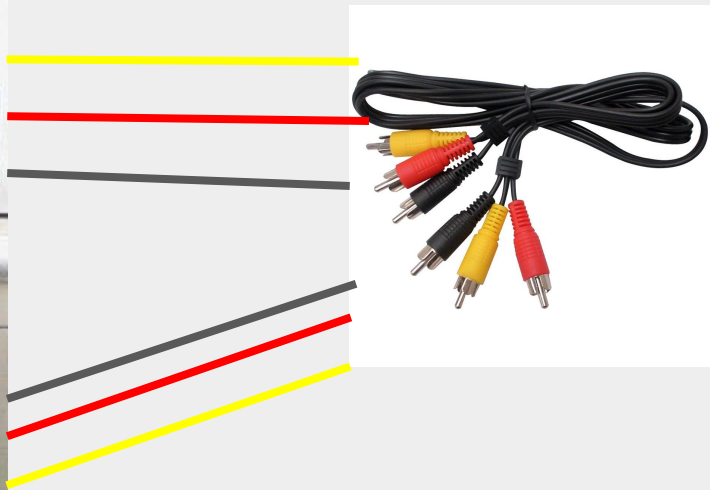
Varyings (связь между vertex и fragment)

WebGL State и Uniforms





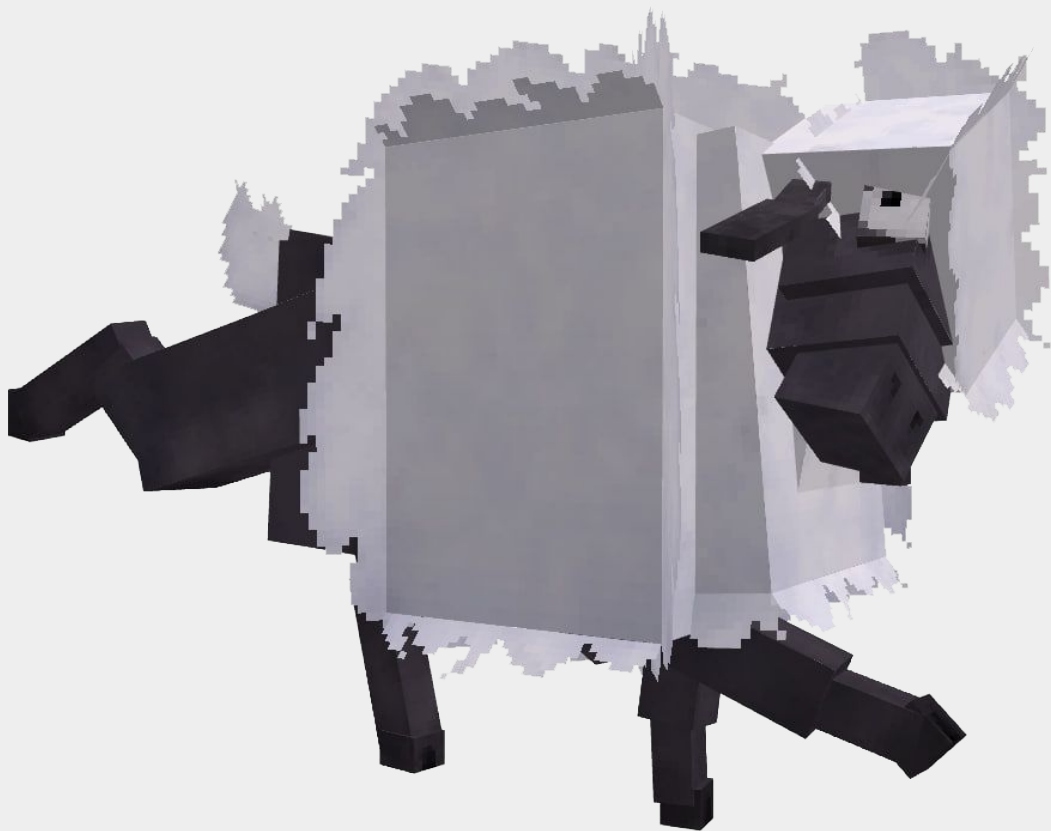
Только когда видик ещё не запущен!

Transform Feedback: из буфера в буфер, копируем Varyings



Обработка данных, Текстура VS Буфер

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



Start Here!

```
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
createConvolvedAndPrefilteredTextureData(skyBox);
}

void SkyBoxLoaderAndDrawer::loadTexture(const char* filename,
GLuint framebuffer, int format)
{
    skyBox.texture = stbi_load(filename, &width, &height, &channels, format);
    if (!skyBox.texture) {
        std::cerr << "Failed to load texture: " << filename << "\n";
        return;
    }
    GLuint texID = glCreateTexture2D(framebuffer, width, height, format, skyBox.texture);
    glTexStorage2D(texID, 1, format, width, height);
    glTexSubImage2D(texID, 0, 0, 0, width, height, format, skyBox.texture);
    glDeleteTextures(1, &texID);
}

data = stbi_load_from_memory((unsigned char *)con
```





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 квадра 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++;

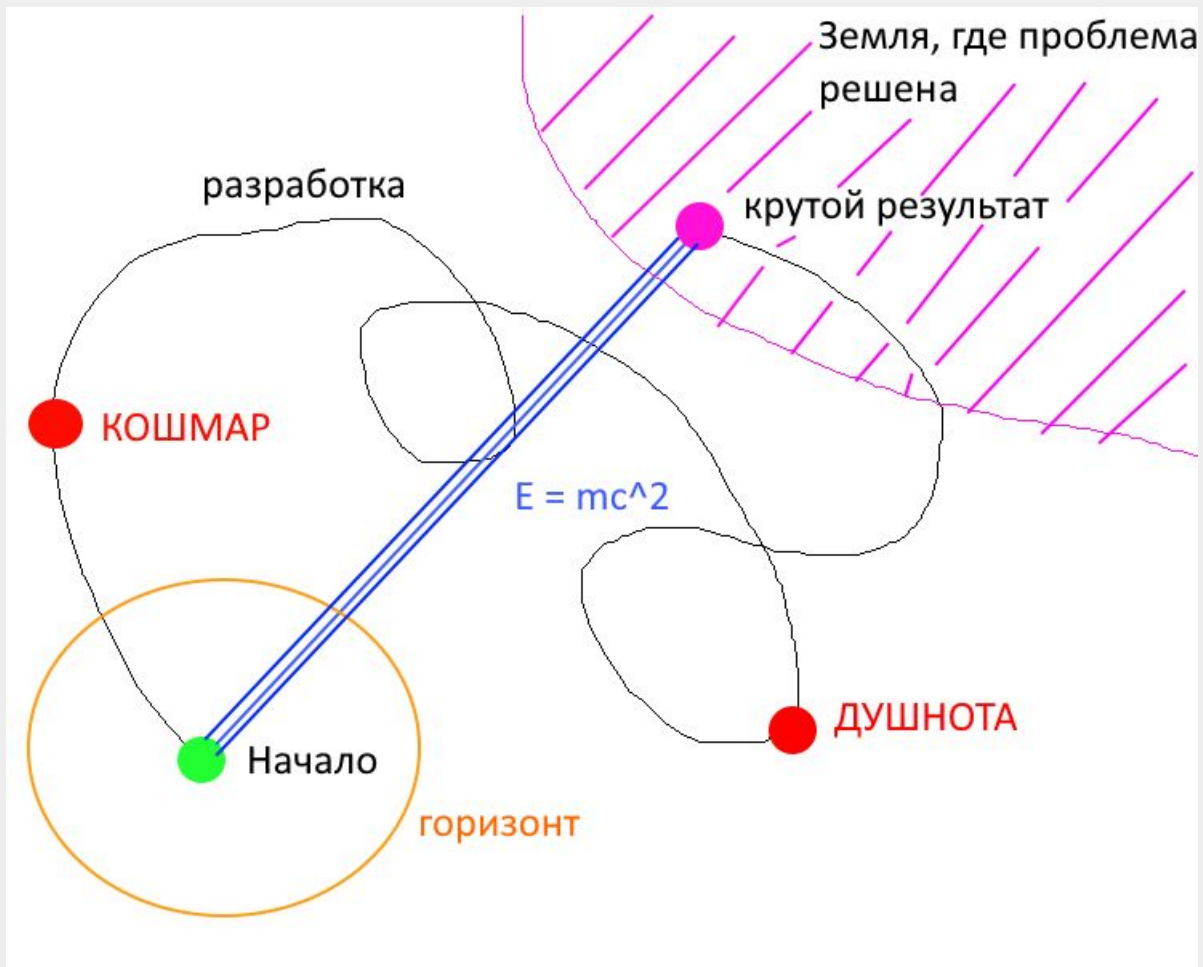
  this.transform.updateTransform(this.parent.transform);

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

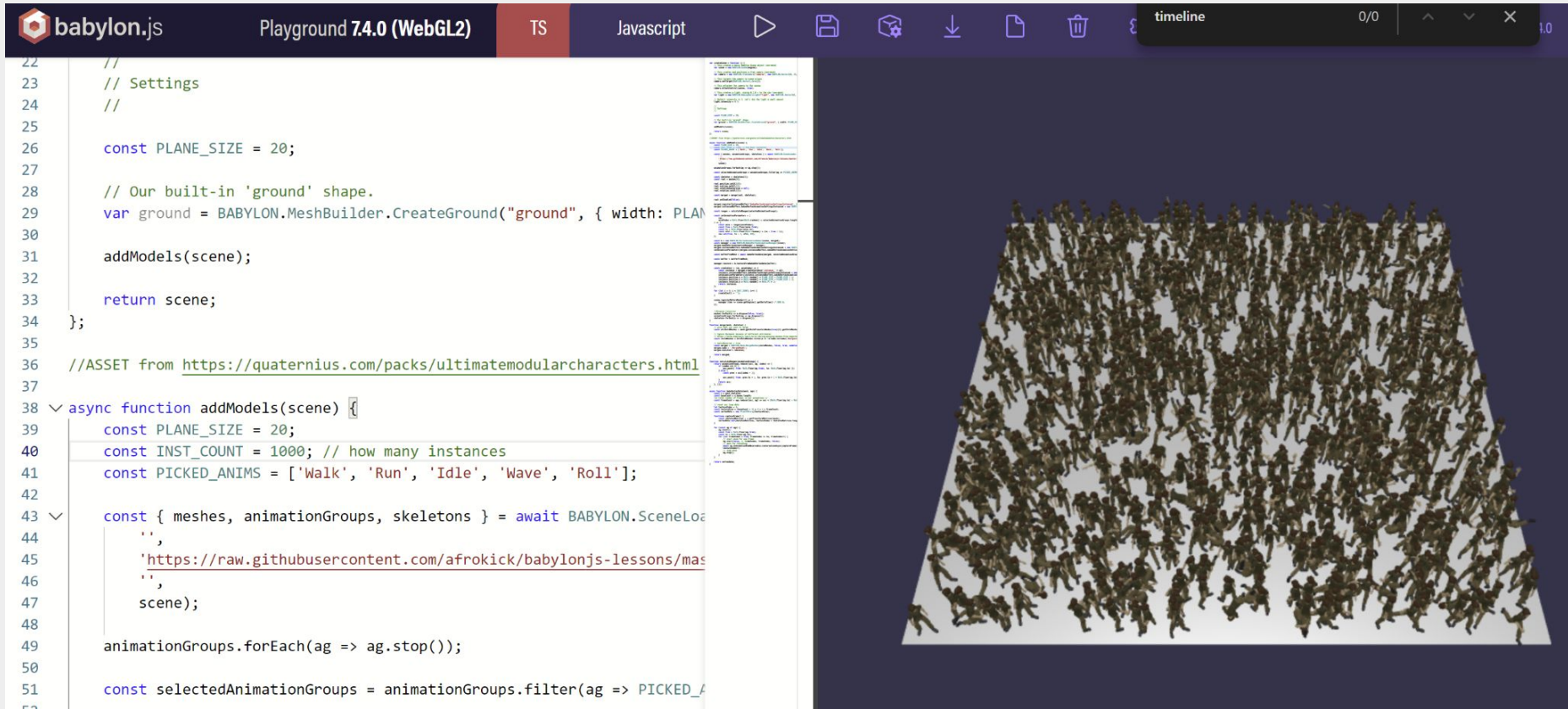
  for (let i = 0, j = this.children.length; i < j; ++i)
  {
    const child = this.children[i];

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

Таков путь



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



The image shows a screenshot of the Babylon.js Playground 74.0 (WebGL2) interface. The top bar includes the Babylon.js logo, the playground name, and tabs for 'TS' and 'Javascript'. A toolbar with icons for play, save, home, download, file, trash, and search is visible. A 'timeline' panel shows '0/0' and navigation arrows. The main area is split into a code editor on the left and a 3D scene on the right.

```
22 //
23 // Settings
24 //
25
26 const PLANE_SIZE = 20;
27
28 // Our built-in 'ground' shape.
29 var ground = BABYLON.MeshBuilder.CreateGround("ground", { width: PLAN
30
31 addModels(scene);
32
33 return scene;
34
35
36 //ASSET from https://quaternius.com/packs/ultimatemodularcharacters.html
37
38 ▼ async function addModels(scene) {
39     const PLANE_SIZE = 20;
40     const INST_COUNT = 1000; // how many instances
41     const PICKED_ANIMS = ['Walk', 'Run', 'Idle', 'Wave', 'Roll'];
42
43     ▼ const { meshes, animationGroups, skeletons } = await BABYLON.SceneLoa
44         '',
45         'https://raw.githubusercontent.com/afrokick/babylonjs-lessons/mas
46         '',
47         scene);
48
49     animationGroups.forEach(ag => ag.stop());
50
51     const selectedAnimationGroups = animationGroups.filter(ag => PICKED_A
52
```

The 3D scene on the right displays a large number of small, dark, humanoid characters (models) scattered across a white, trapezoidal ground plane. The characters are in various poses, suggesting they are performing different animations. The background is a dark blue gradient.

```
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

Experiments

122.0.6192.0

WARNING: EXPERIMENTAL FEATURES AHEAD! By enabling these features, you could lose browser data or compromise your security or privacy. Enabled features apply to all users of this browser. If you are an enterprise admin you should not be using these flags in production.

Available

Unavailable

● Choose ANGLE graphics backend

Choose the graphics backend for ANGLE. D3D11 is used on most Windows computers by default. Using the OpenGL driver as the graphics backend may result in higher performance in some graphics-heavy applications, particularly on NVIDIA GPUs. It can increase battery and memory usage of video playback. – Windows

[#use-angle](#)

D3D11on12



[Radeon GPU Profiler](#)

[NVidia NSight](#)

Intel Performance
Graphics Analyzers

Radeon GPU Profiler

Wavefront occupancy

Event timing

Pipeline state

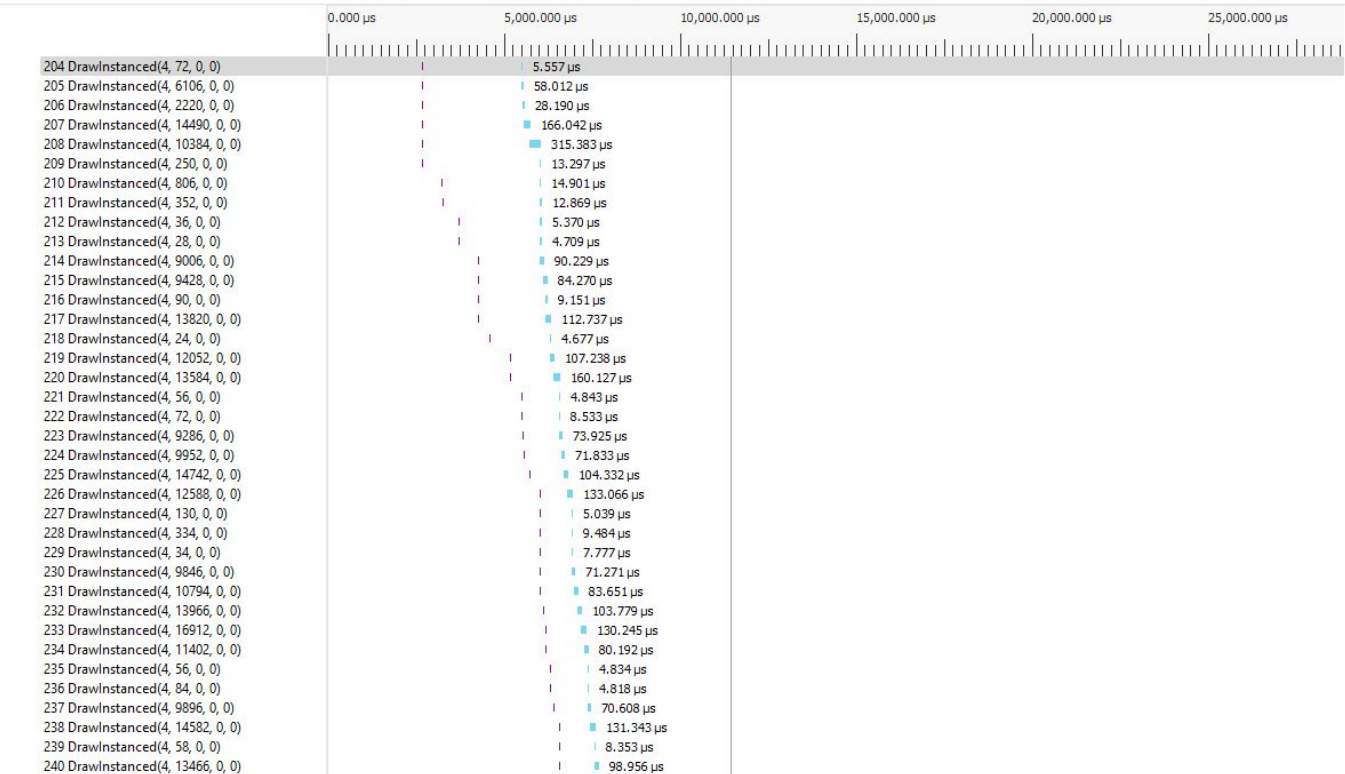
Instruction timing

Collapse tree

Group by pass

Color by queue

Filter event tre...



204 DrawInstanced(4, 72, 0, 0)

Launched from Direct queue

Start time	5,523.008 µs
End time	5,528.565 µs
Duration	5.557 µs
Work duration	5.144 µs
Hardware context	4

API shader hashes

VS: 0x32B34E5C8E04D3BF1A235532C61B765A

PS: 0xC7EC384117927CA038E75E8AA29782FB

API PSO hash

0x8A378B3681ED9D07

Driver internal pipeline hash

0x6D161CF3BD3D010C8A378B3681ED9D07

Wavefronts

RDNA wavefront distribution:



SurfS wavefronts	-
PrimS wavefronts	10 (90.91%)
PS wavefronts	1 (9.09%)
CS wavefronts	-

Total wavefronts 11
Total threads 15

RDNA shader stage timeline



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_fmact_f32_e32	v2, v3, v3
3797	v_sqrt_f32_e32	v2, v2
3798	v_fmaak_f32	v8, -0.5, v2, 0x3dccccce
3799	v_fmaak_f32	v3, 0.5, v2, 0x3dccccce
3800	v_sub_f32_e32	v2, v3, v8
3801	v_sub_f32_e32	v3, 0x3e99999a, v8



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, 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)
351 DrawInstanced(4, 2296, 0, 0)
352 DrawInstanced(4, 4446, 0, 0)
353 DrawInstanced(4, 1050, 0, 0)
354 DrawInstanced(4, 5770, 0, 0)
355 DrawInstanced(4, 4248, 0, 0)
356 DrawInstanced(4, 4678, 0, 0)
357 DrawInstanced(4, 4968, 0, 0)
358 DrawInstanced(4, 4454, 0, 0)
359 DrawInstanced(4, 3498, 0, 0)
360 DrawInstanced(4, 1676, 0, 0)
361 DrawInstanced(4, 4178, 0, 0)
362 DrawInstanced(4, 4396, 0, 0)
363 DrawInstanced(4, 7454, 0, 0)
364 DrawInstanced(4, 584, 0, 0)
365 DrawInstanced(4, 1572, 0, 0)
366 DrawInstanced(4, 5808, 0, 0)
367 DrawInstanced(4, 5778, 0, 0)

```



Information

ISA

Wavefronts and threads

Total wavefronts	Total threads	Average wavefront duration	Average threads per wavefront
3,294	170,901	8.307 μs	51

Wavefront mode

wave64

Per-wavefront resources

Vector registers	Scalar registers	Registers spilled to scratch memory
64 (64 allocated)	62 (128 allocated)	<input type="checkbox"/> 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 проходит 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) {  
    // записать кубических детей в geometries  
  }  
  batch.addParts(this.geometries,  
    this.world_matrix, this.local_bone_id)  
}
```

Инстансинг на коленке: шаринг анимаций

Инстанс! X 

Квады (2) = X + обычные (2)



Текстура
инстанса

Квады + № инстанса (3)



Шейдер
инстанса



FOR FOR FOR



TRUE INSTANCING

Transform
feedback

=

Квады для
батчера



Меши
Стат квады
Дин квады
Кости



Шейдер, в
ОСНОВНОМ
просто читает
текстуры

X

№ меша
№ квадра
№ кости





Когда кода осталось мало

Вызов расчёта на GPU вместо CPU

Выигрыш
перфа

=

X

- перерасчет

Кадры (FPS)

- 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`)