Человеческое WebGL в и хардкор с рисованием линий

by Иван from PixiJS и CRAZY PANDA

Для HOLY JS

План доклада (оптимистичный)

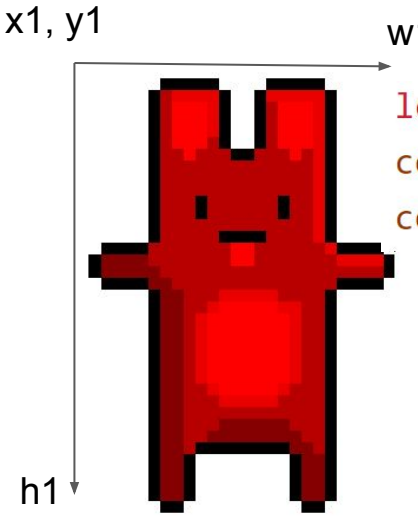
1. Ввод для тех кто графикой не занимался или забыл. 10000 кроликов.
2. Описание проблемы, как сейчас рисуют линии в Canvas / WebGL
3. Геометрия: что (пере-)изобрели за 10 лет
4. Сглаживание: **надеюсь вы до него доживёте**
5. Примеры использования в production и возможные оптимизации

Хардкорная часть доклада на английском:
https://www.khronos.org/assets/uploads/developers/presentations/Crazy_Panda_How_to_draw_lines_in_WebGL.pdf

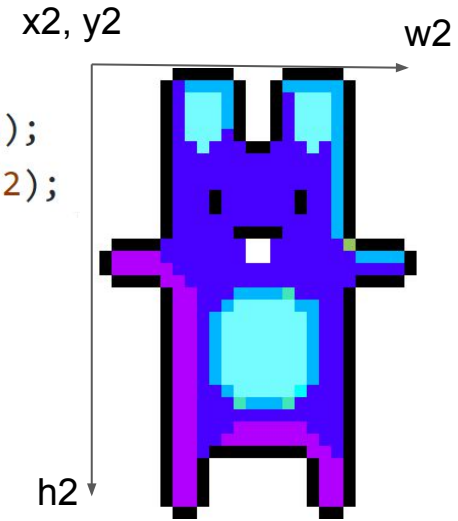
Основы Canvas 2D: как сделать 2 кролика

- Две строчки, уже понятно как работает.
- Работает вообще везде
- Хорошо держит 1000 кроликов

Canvas 2D




```
let context = canvas.getContext('2d');
context.drawImage(bunny_red, x1, y1, w1, h1);
context.drawImage(bunny_blue, x2, y2, w2, h2);
```



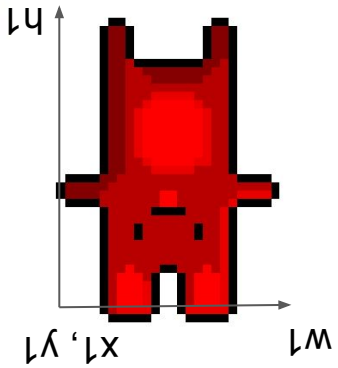
Основы WebGL: как сделать 2 кролика

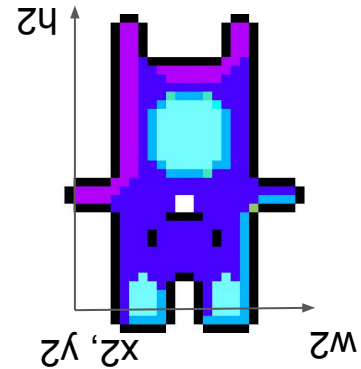
```
let texCoordLocation = gl.getAttribLocation(program, "a_texCoord");
let positionLocation = gl.getAttribLocation(program, "a_position");

let texCoordBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, texCoordBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([0.0,  0.0, 1.0,  0.0,
        0.0,  1.0, 0.0,  1.0, 1.0,  0.0, 1.0,  1.0]), gl.STATIC_DRAW);
gl.enableVertexAttribArray(texCoordLocation);
gl.vertexAttribPointer(texCoordLocation, 2, gl.FLOAT, false, 0, 0);

let positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([
        x1, y1, x1 + w1, y1, x1, y1 + h1, x1,
        y1 + h1, x1 + w1, y1, x1 + w1, y1 + h1]), gl.STATIC_DRAW);
gl.enableVertexAttribArray(positionLocation);
gl.vertexAttribPointer(positionLocation, 2, gl.FLOAT, false, 0, 0);
```

WebGL





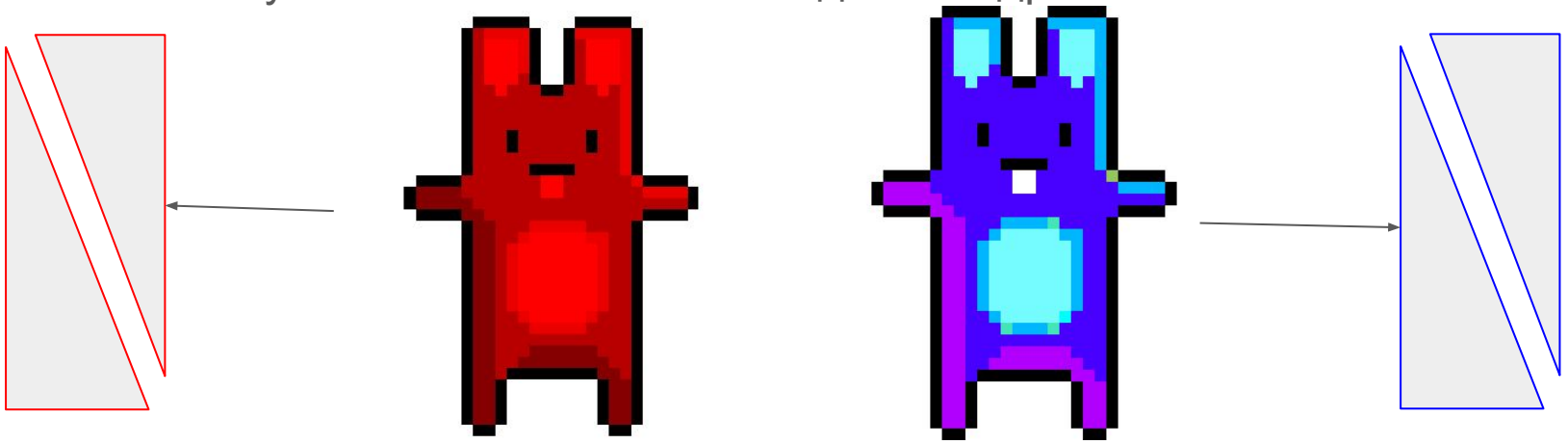
Стоп! Давайте пока без кода

1. С первого раза у вас только чёрный экран получится
2. Голое WebGL API взрывает мозг, а в WebGPU оно ещё и другое
3. Главное - как преобразуются данные!

Структура одного вызова отрисовки (draw call)

1. Задать геометрию : координаты треугольников, много данных
2. Задать uniforms : константы (цвета, текстуры), чуток данных
3. Задать шейдер: алгоритм отрисовки
4. DrawCall!

Таких вызовов могут быть сотни в течение одного кадра.



Как формируется геометрия, на примере двух кроликов

Аттрибуты (**Ввод**)

Индекс

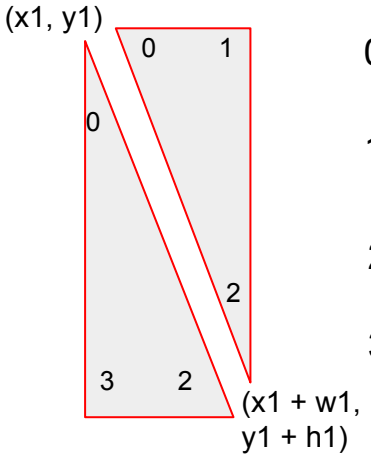
Инстанс : x,y,w,h,цвет

0, 2, 3	0, 1, 2

Обычные u, v

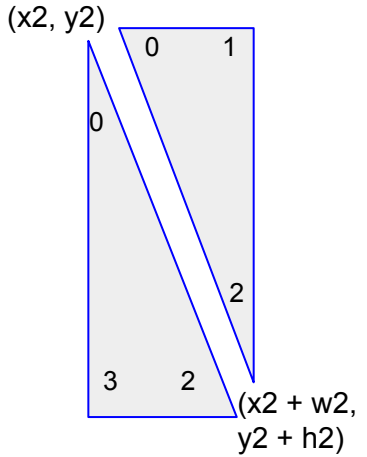
0	u=0, v=0
1	u=0, v=1
2	u=1, v=1
3	u=1,v=0

x1, y1, w1, h1, ■	x2, y2, w2, h2, ■



Вывод

x1, y1, ■	x2, y2, ■	0
x1 + w1, y1, ■	x2 + w2, y2, ■	1
x1 + w1, y1 + h1, ■	x2 + w2, y2 + h2, ■	2
x1, y1 + h1, ■	x2, y2 + h2, ■	3



Uniforms

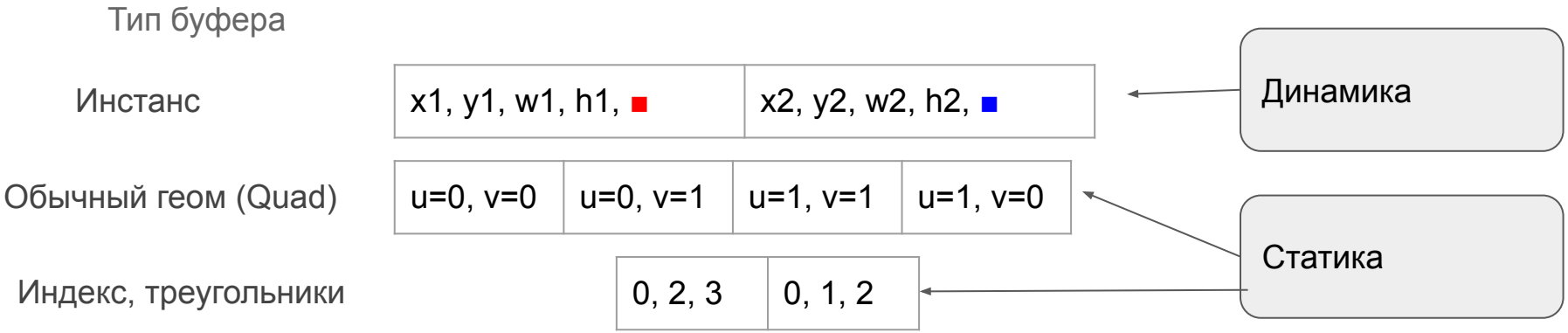
camera_scale

camera_translate

Vertex Shader

```
gl_Position.x = (x + w * u) * camera_scale.x + camera_translate.x;
gl_Position.y = (y + h * v) * camera_scale.y + camera_translate.y;
out_color = color;
out_uv = vec2(u, v);
```

Что на входе геометрии



- Статика: один раз
- Динамика: каждый кадр

Как получить 10.000 кроликов за 1 drawcall

Инстанс

x1, y1, w1, h1, ■	x2, y2, w2, h2, ■
x3, y3, w3, h3, ■	x4, y4, w4, h4, ■
x5, y5, w5, h5, ■	x6, y6, w6, h6, ■
x7, y7, w7, h7, ■	x8, y8, w8, h8, ■
x9, y9, w9, h9, ■	...
...	...

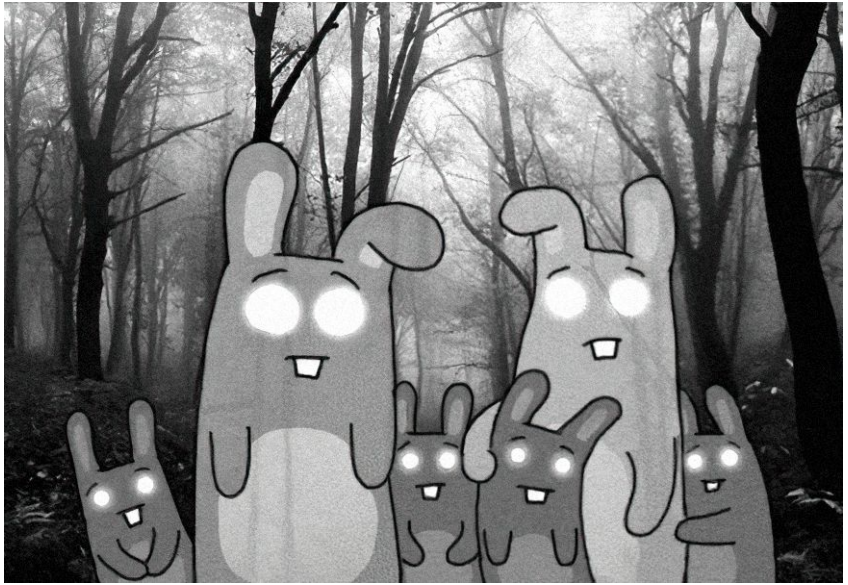
Если хоть один поменял координату - на следующий кадр придётся перезаливать всё

Напоминание: API всё ещё ужасно

```
const posBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, posBuffer);
const posArray = new ArrayBuffer(5 * 2 * 4);
const posFloat = new Float32Array(posArray);
const posUint = new Uint32Array(posArray);
posFloat.set([x1, y1, w1, y1], 0); posUint[4] = RED;
posFloat.set([x2, y2, w2, h2], 5); posUint[9] = BLUE;
gl.bufferData(gl.ARRAY_BUFFER, posArray), gl.DYNAMIC_DRAW);
gl.vertexAttribPointer(x_loc, 1, gl.FLOAT, false, 0, 0);
gl.vertexAttribPointer(y_loc, 1, gl.FLOAT, false, 0, 0);
gl.vertexAttribPointer(w_loc, 1, gl.FLOAT, false, 0, 0);
gl.vertexAttribPointer(h_loc, 1, gl.FLOAT, false, 0, 0);
gl.vertexAttribPointer(color_loc, 4, gl.UNSIGNED_BYTE,
    true, 0, 0);
gl.vertexAttribDivisor(x_loc, 1);
gl.vertexAttribDivisor(y_loc, 1);
gl.vertexAttribDivisor(w_loc, 1);
gl.vertexAttribDivisor(h_loc, 1);
gl.vertexAttribDivisor(color_loc, 1);
```

кролики тут

включается инстансинг

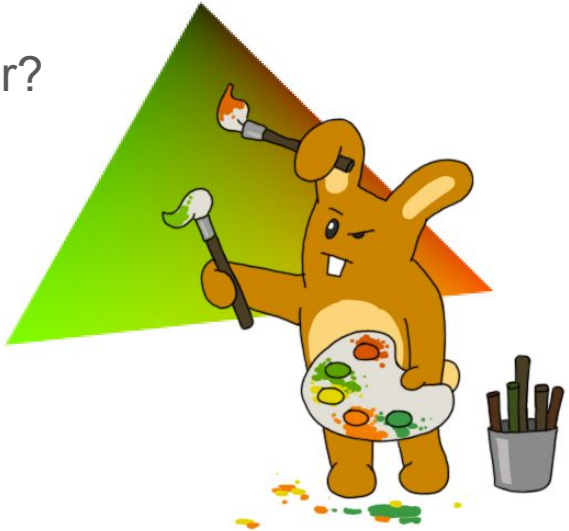


```
const quadBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, quadBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(
    [0, 0, 1, 0, 0, 0, 0, 1]), gl.STATIC_DRAW);
gl.vertexAttribPointer(quad_loc, 2, gl.FLOAT, false, 0, 0);

const indexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(
    [0, 2, 3, 0, 1, 2]), gl.STATIC_DRAW);
```

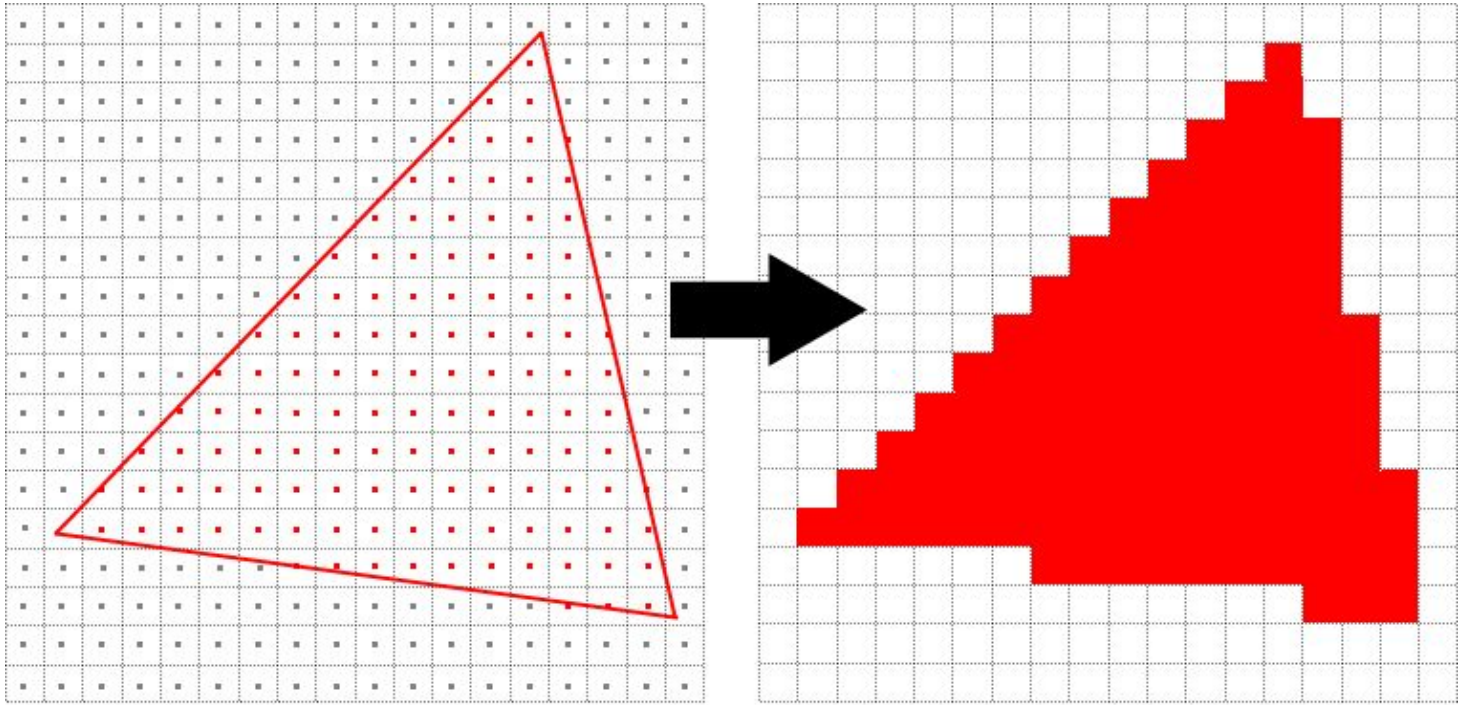
Заливка пикселей

1. Какие пиксели заливаются?
2. Какие данные приходят на fragment shader?
3. Как взять данные с текстуры кролика?



Нам писать ничего не надо. WebGL так работает. Не трогайте!

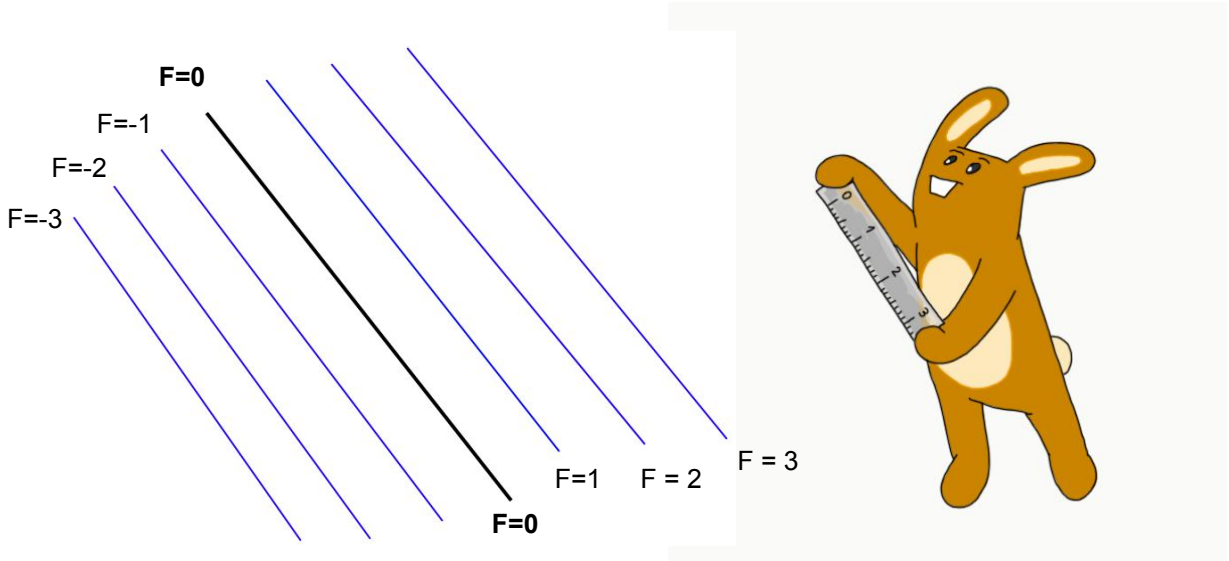
Какие пиксели заливаем?



- Sample point
- Sample point covered by the triangle

Важная информация: линейная функция на плоскости

- Положим F(x, y) = a * x + b * y + c
- При разных a, b, c уровни функции будут прямыми на равном расстоянии друг от друга
- Расстояние от точки до фиксированной прямой является линейной функцией
- Линейная функция однозначно задаётся по значениям в трёх точках

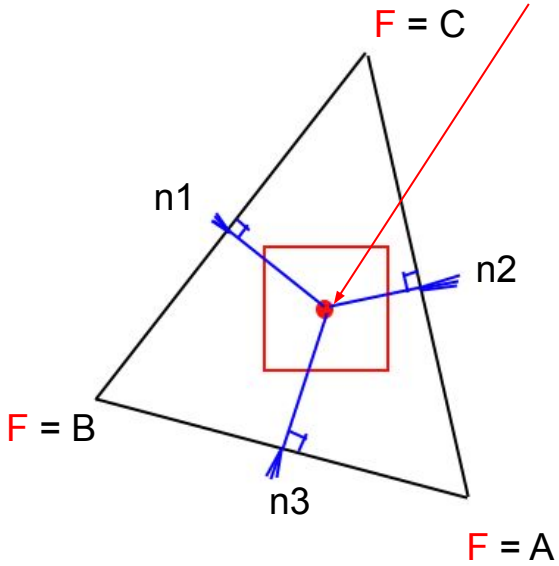


Нам писать ничего не надо. WebGL так работает. Не трогайте!

Интерполяция: о мой кролик, что это?

Дано: все координаты и значения F вершин треугольника

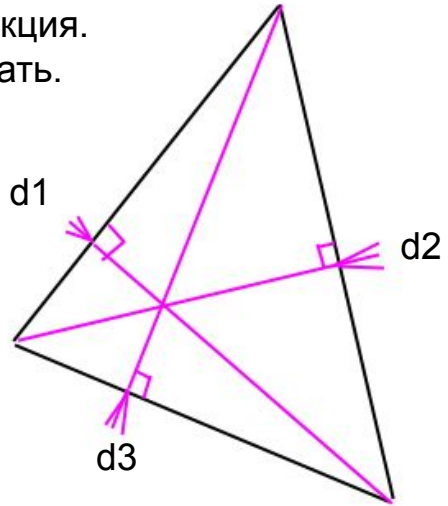
Вопрос: чему равно F для центра пикселя?



Решение: высота - линейная функция.
попробуем что-нибудь с ней сделать.

r1 = n1 / d1
r2 = n2 / d2
r3 = n3 / d3

r1 + r2 + r3 = 1



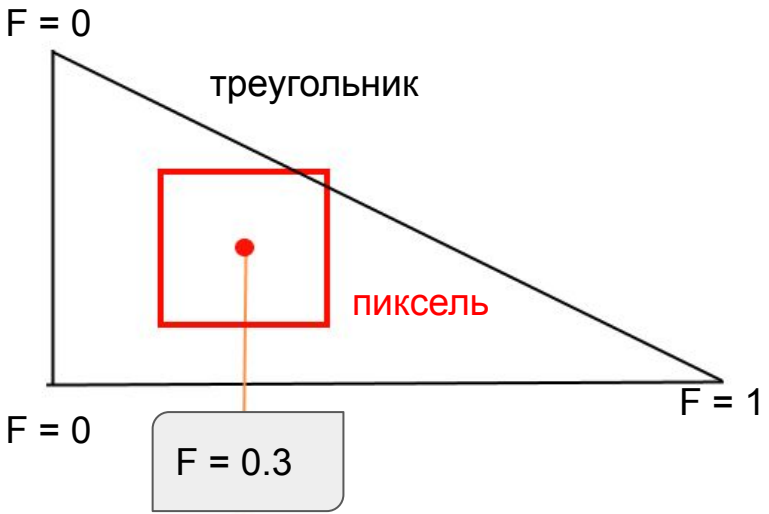
F = r1 * A + r2 * B + r3 * C

СЛОЖНА !!!

Для умников: это не барицентрическое что-то там

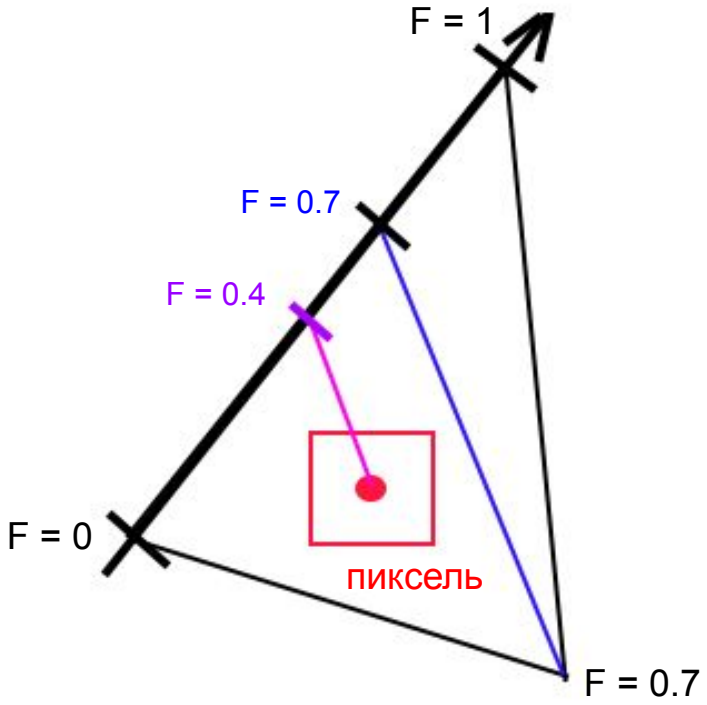
Интерполяция: простой случай

- Прямоугольный треугольник
- Две вершины имеют F = 0, третья F = 1
- Для центра пиксели F будет соответственно отсечке на оси - катете



Интерполяция: сложный случай

- Допустим две вершины имеют F = 0, F = 1
- Проведем от третьей линию до соответствующего F=0.7 на оси между первой и второй вершиной
- Проведем параллельную линию от центра пикселя, найдем F для него



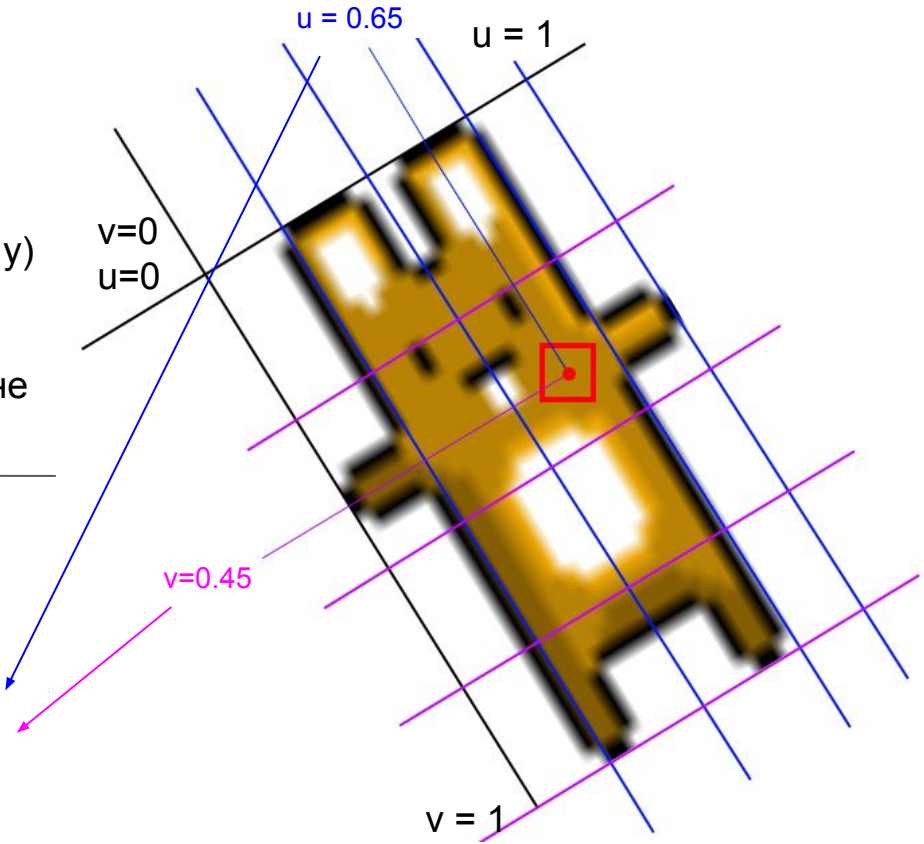
Интерполяция: финал

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;
```



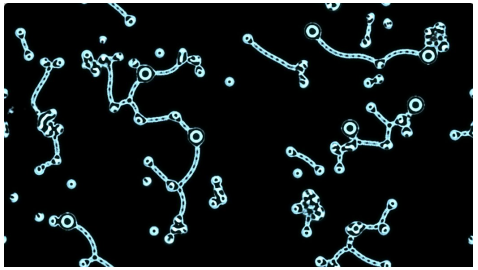
Фрагментный шейдер - всему голова

- Выполняется для каждого пикселя
- Получает на вход информацию от интерполятора
- Имеет доступ к Uniforms, в частности к сэмплерам (указатели на текстуры)

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

Q: Вроде бы он простой, зачем он нужен, почему вместо когда не сделать какие-то режимы заливки?

A: Это мощное, но дорогое средство, на нём можно вообще написать там всю программу: по координате пикселя вычислять цвет в математической сцене описываемой raymarching SDF.



Smooth Life (L)
https://www.shadertoy.com/view/XtdSDn

Happy Jumping
https://www.shadertoy.com/view/3lsSzf

В далёком прошлом, на какой-то игровой приставке...

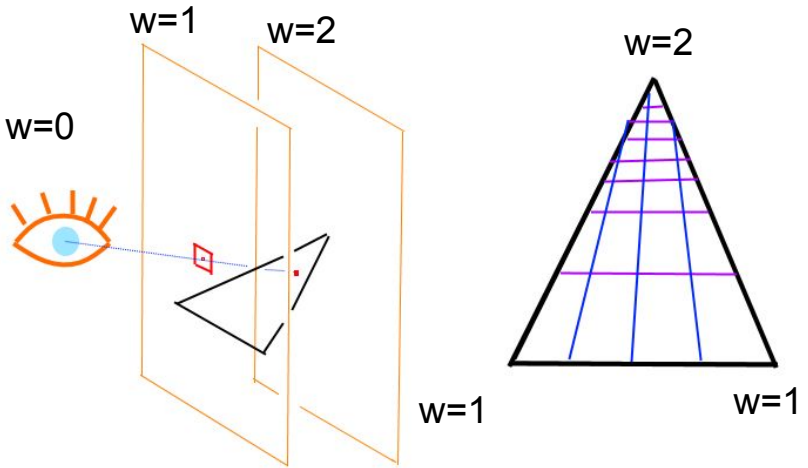
Интерполяция: А что с 3D ?

Доски погнуты

А должно быть вот так

Интерполяция: война бесконечности

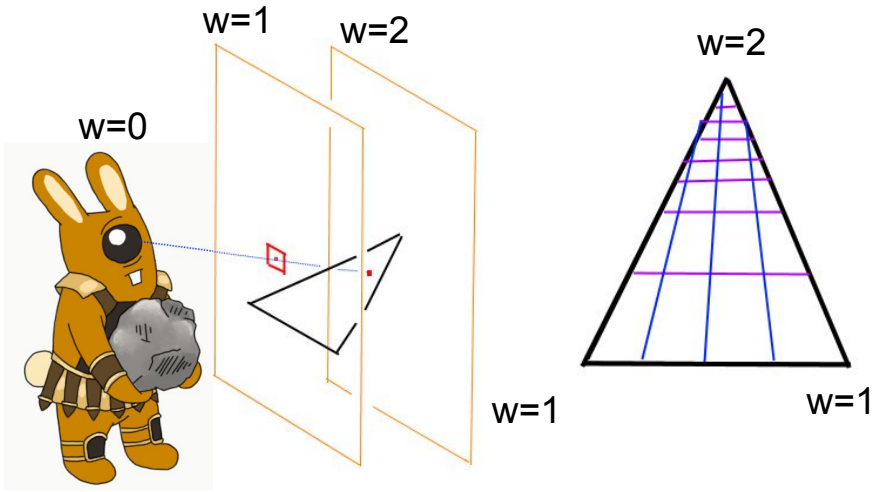
- В 3D добавляется проекционная координата
- Чем дальше вершина, тем больше скукоживается сетка
- Этого нет в Canvas 2d



Можно построить школьной геометрией, но сейчас мне лень

Интерполяция: война бесконечности

На самом деле "просто":
1. Представить что w линейная. Посчитать для центра пикселя, пусть получилось w0
2. Взять F * w , посчитать её для центра пикселя как будто она линейная, пусть получилось F0
3. Результат это F0 / w0 , из-за того что там где-то подобные треугольники



Vertex Shader

```
// d - расстояние до глаза
gl_Position.x *= d;
gl_Position.y *= d;
gl_Position.w = d;
```

Часть 2. Описание хардкорной проблемы

open-source решения нет. коммерческого наверное тоже

Основы Canvas: линии

- Базовое рисование как в Turbo Pascal
- И как во Flash, Java, C#
- Можно даже нарисовать пони
- Можно менять ширину линии и будет гладенько

```
context.lineStyle = 'blue';
context.beginPath();
context.moveTo(330,249);
context.lineTo(340,232);
context.lineTo(342,237);
context.lineTo(344,235);
context.stroke();
```

Canvas 2D

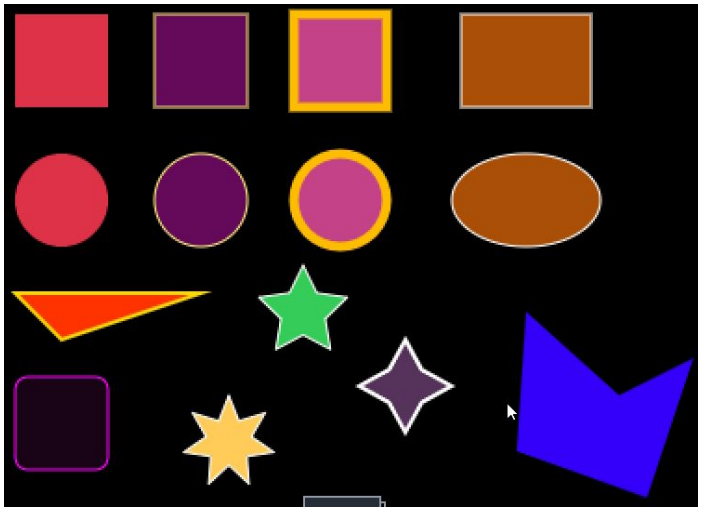
Join & Cap

Join:

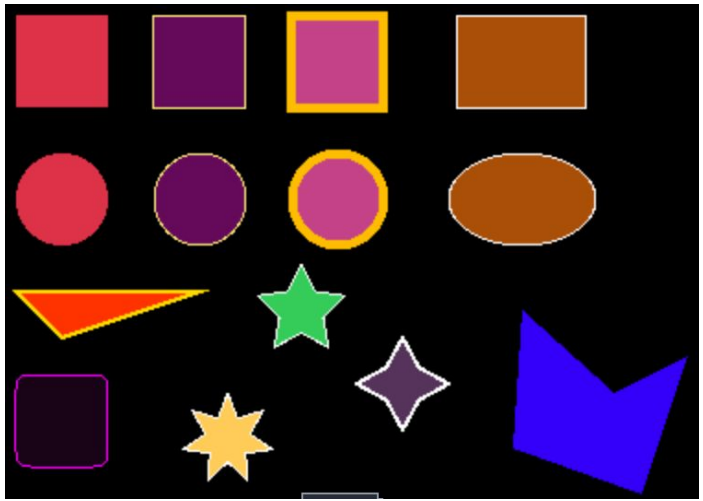
Cap:

getContext('2d') **vs** getContext('webgl')

PixiJS canvas 2d



PixiJS WebGL , без AA

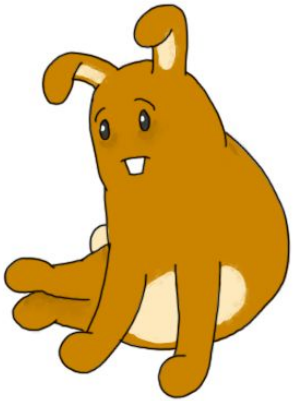


Как нарисовать линию в WebGL

"Нативная" линия в WebGL API

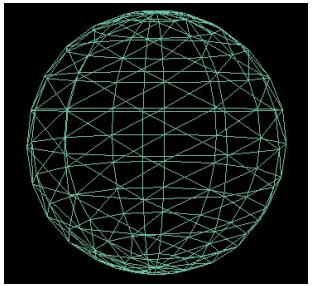
- drawArrays , LINE_STRIP
- lineWidth вроде должно всё решать

> As of January 2017 most implementations of WebGL only support a minimum of 1 and a maximum of 1 as the technology they are based on has these same limits.



Вывод: подходит только для отладки

(wire mesh)

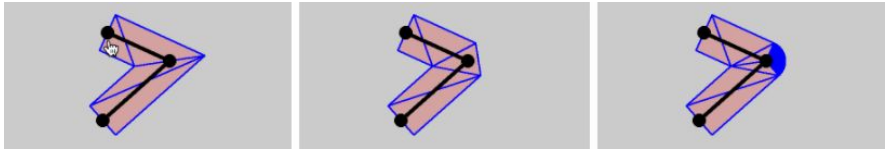


Как нарисовать линию в WebGL

Сделать меш (кучу треугольников)

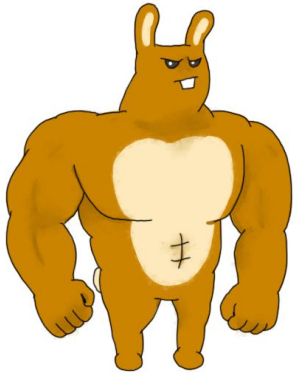
Вычислить все точки на JS (PIXI.Graphics) смотря на путь и

- lineStyle { width, color, texture }
- lineStyle { lineCap, lineJoin, miterLimit }
- lineStyle { alignment } ??? - объясним позже



Для вершин есть position, color, uv на фрагмент уходит color, uv

Поддерживает сочленения потому что у нас крутое коммьюнити!

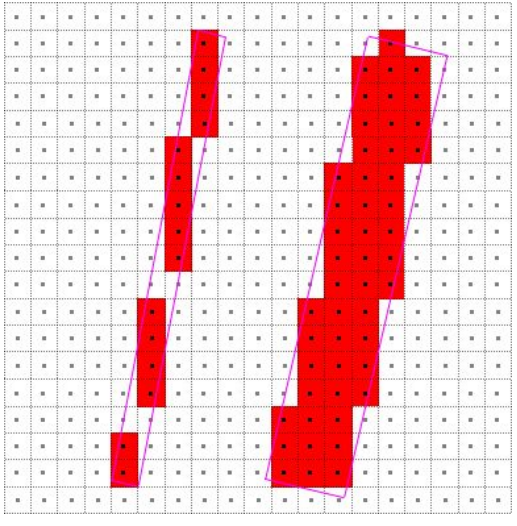


WebGL MultiSampling Anti-Aliasing (MSAA)

Без MSAA

Для каждого пикселя у которого центр попал в какой-то треугольник, вызываем fragment shader. Хорошо работает только на вертикальных и горизонтальных линиях.

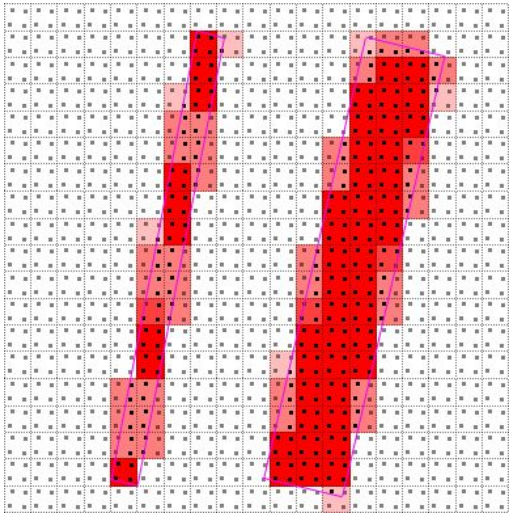
При тонкой линии может начать пропускать пиксели!



MSAA x16

Для каждого пикселя храним 16 сэмплов, если хотя бы один покрыт - вызываем fragment shader и смешиваем результат во все покрытые сэмплы. Потом объединяем сэмплы в пиксель.

- x6 памяти Вообще-то я не помню
- тормозит **всё**, не только линии
- рисование текстуру - только в WebGL2
- renderbufferStorageMultisample вызывается на каждый буфер



в примере MSAA x4,

x16 показать трудно

Как нарисовать линию в Canvas 2d

1. Просто beginPath, moveTo, lineTo, closePath
2. Заполнить SVG-шный Path2D
3. Свойства: lineWidth, strokeStyle (цвет и узор)
4. Для экспертов: lineJoin, lineCap, miterLimit
5. Начертить путь
6. ПРОФИТ!

Не настраивается:

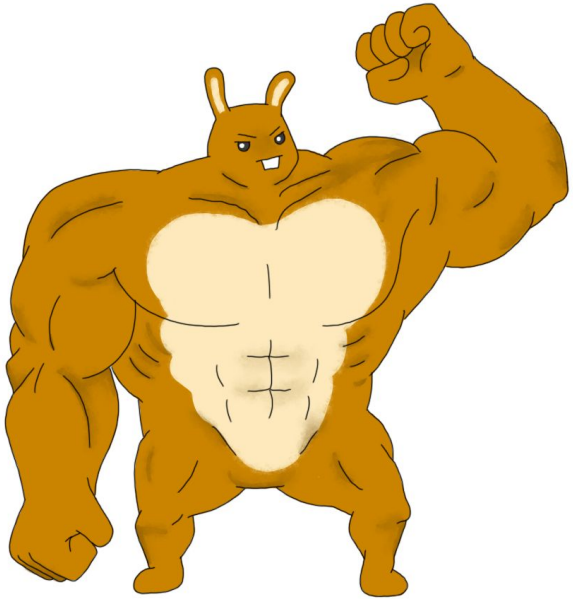
- Линия всегда сглажена
- Ширина всегда меняется при zoom / scale

JavaScript

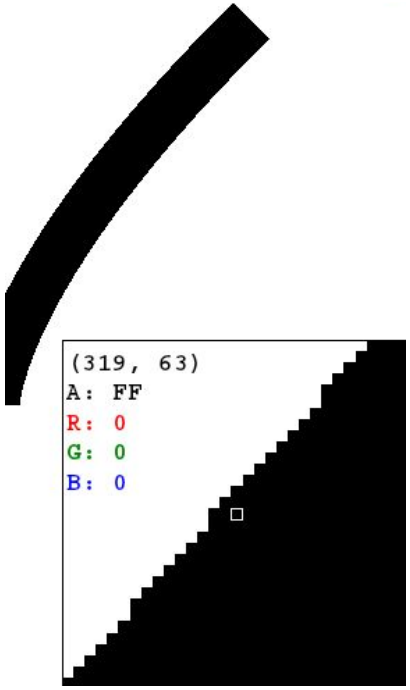
```
const canvas = document.getElementById('
const ctx = canvas.getContext('2d');

ctx.moveTo(90, 130);
ctx.lineTo(95, 25);
ctx.lineTo(150, 80);
ctx.lineTo(205, 25);
ctx.lineTo(210, 130);
ctx.lineWidth = 15;
ctx.stroke();
```

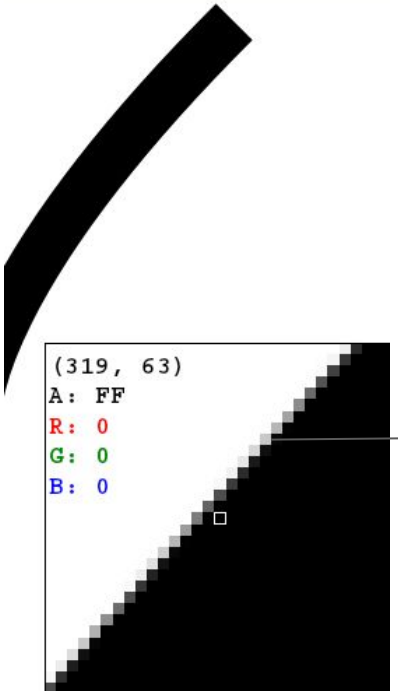
Result



Источник: https://skia.org/docs/dev/design/aaa/

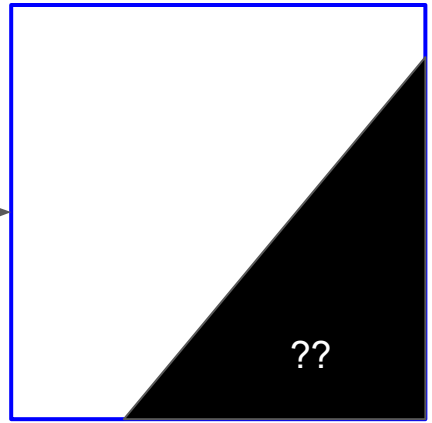


No AA



AA

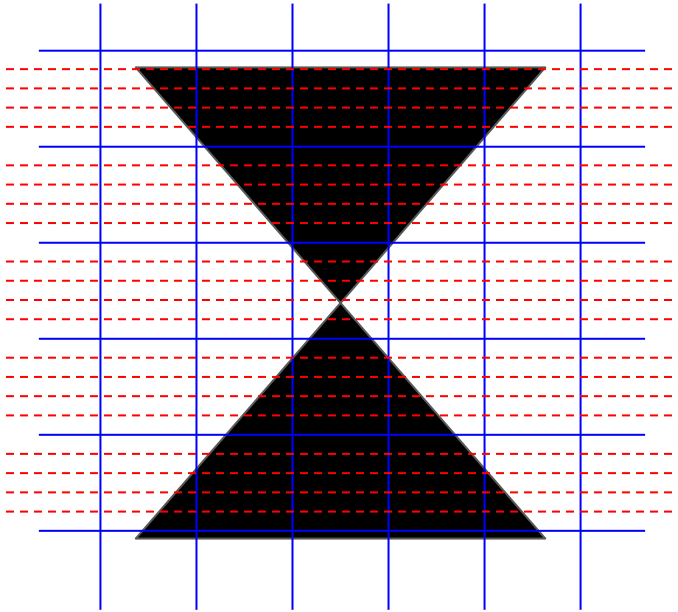
The number to compute:
coverage per pixel
(Area of the Intersection)



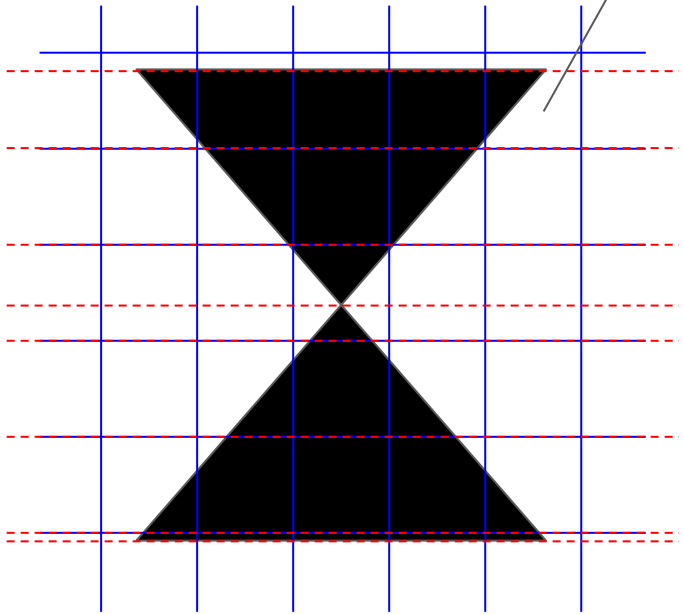
A single pixel
(unit area = 1.0 = 0xFF = 255)

Источник: https://skia.org/docs/dev/design/aaa/

CPU Scanline: old vs. new



16x supersampling:
4 scan-lines per pixel



Analytic AA:
1 scan-line per pixel, per edge endpoint, per intersection

There are only trapezoids (triangles, rects) between scanlines

(similar to Cairo and Direct2D, except we never tesselate explicitly)

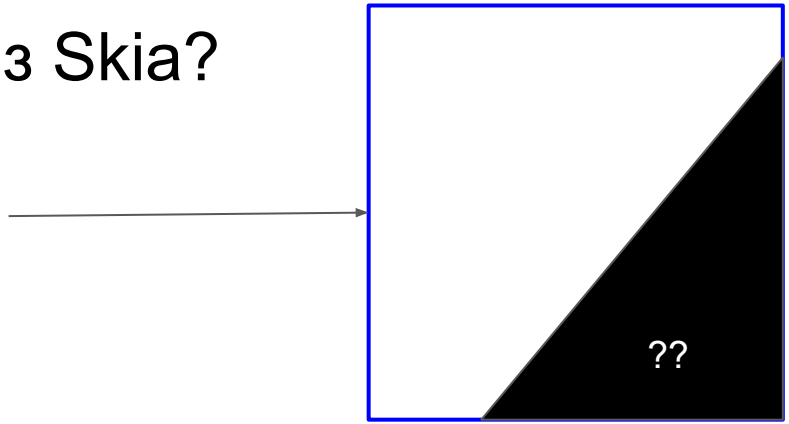
Вывод: почему нельзя сделать то же самое в WebGL?

- Главный алгоритм - Scanline - работает нативно на CPU. Портировать это на JS выйдет, но будет очень медленно
- Даже если взять wasm, придётся подавать результат в GPU с помощью texImage2D - а загрузка данных на видюху у нас более дорогая чем у Skia!
- Если бы было возможно, Skia уже бы перенесли всё на GPU
- Некоторым частям нужны геометрические шейдеры, а они есть только в (WebGPU)

Что взять от слайдов ребят из Skia?

Забудьте всё!

Надо вычислить покрытие!



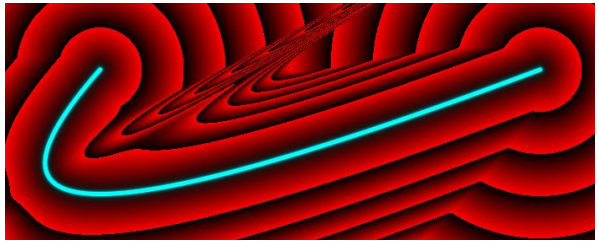
Кривые Безье? SDF? Другие умные слова?

Красивости

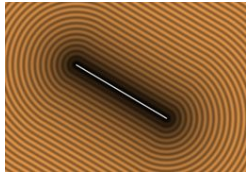
Shadertoy bezier curve спасибо Taylor Holliday.

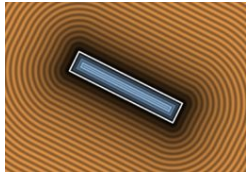
Заметим интересный фрагмент фрагмента:

```
if(d < thickness) {
  a = 1.0;
} else {
  // Anti-alias the edge.
  a = 1.0 - smoothstep(d, thickness, thickness+1.0);
}
```



коллекция 2D shapes SDF спасибо Inigo Quilez





B production

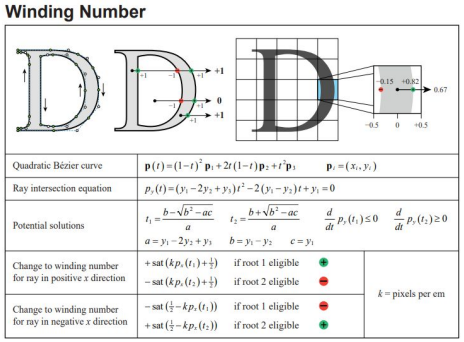
библиотека Slug для шрифтов

Хорошо:

- Не только линии, любые фигуры!
- Круто!
- Лучшее качество!

Плохо:

- Платная лицензия
- Всё на fragment shader - тормоза



Вывод

Если всё на фрагменте - это тормоза.

Сколько можно читать из текстур с данными на тормозных устройствах?

Насколько трудная математика?

Ну, может мы можем использовать какие-нибудь кусочки.

Часть 3. Готовим геометрию

Магии нет.

Много рендереров имеют реализации геометрии с lineCap / lineJoin, к примеру AwayJS и Next2D . ThreeJS имеет свои fat lines

Следующие статьи содержат улучшения, например как вычислять координаты вершин в шейдере:

- 2013 апрель, Robust polyline rendering with WebGL спасибо Dan Bagnell
- 2015 март, Drawing Lines is Hard спасибо Matt Deslauriers
- 2017 июнь, Рисование толстых линий в WebGL спасибо openglobus
- 2019 ноябрь, Drawing lines in WebGL спасибо Matt Stobbs
- 2019 ноябрь, Instanced line rendering спасибо Rye Terrell
- 2021 октябрь, Instanced Line Rendering Part II: Alpha blending от него же

Сконвертировать пути и точки в сочленения (joint)

1. Убрать дубликаты точек и ненужные рёбра
2. Добавить дополнительные конечные точки
3. Объединить всё в joint data: x, y, type
4. Собрать все joint data в массив

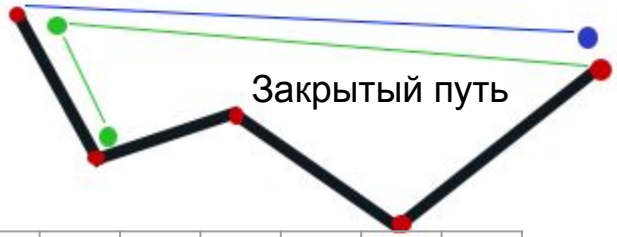
x1	y1	0	x0	y0	cap	x1	y1	join	x2	y2	join	x3	y3	cap	x2	y2	0

данные нужны соседним отрезкам

тип

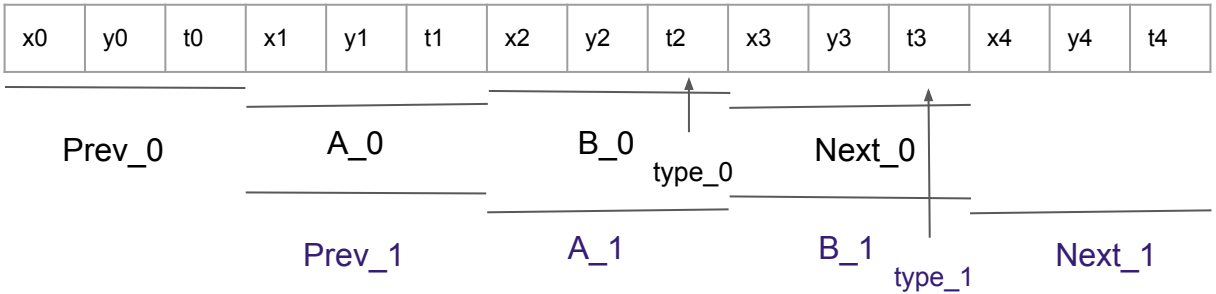
caps: square, butt, round

join: miter, bevel, round

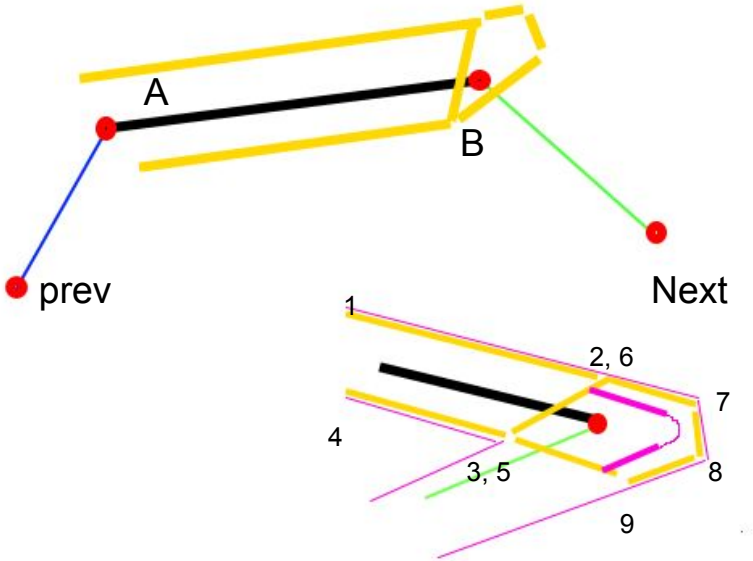


x3	y3	0	x0	y0	0	x1	y1	join	x2	y2	join	x3	y3	join	x0	y0	join	x1	y1	0

Упаковать сочленения в инстансы



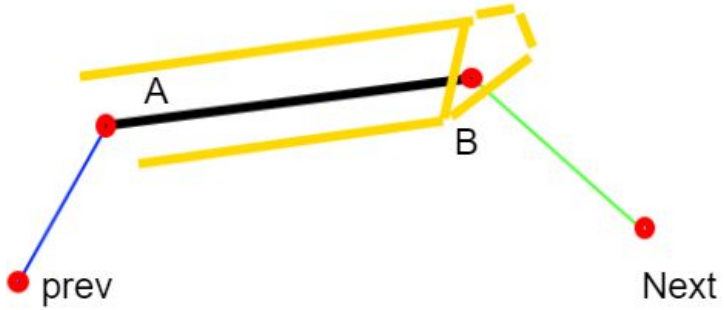
Вершины пронумерованы 1-9



Данные в инстансе:

- Каждый инстанс рисует 1 отрезок и 1 сочленение
- Инстансы перекрываются по данным! 4 точки, съезжают по одной вправо
- 4 вершины на отрезок, 5 на сочленение (BEVEL) = 9 вершин на инстанс!
- Добавить стили в буфер (ширину, цвет, текстуру)
- Добавить доп. стиль (miterLimit, bevelLimit, alignment)

Vertex shader вычисляет координаты и что-то ещё

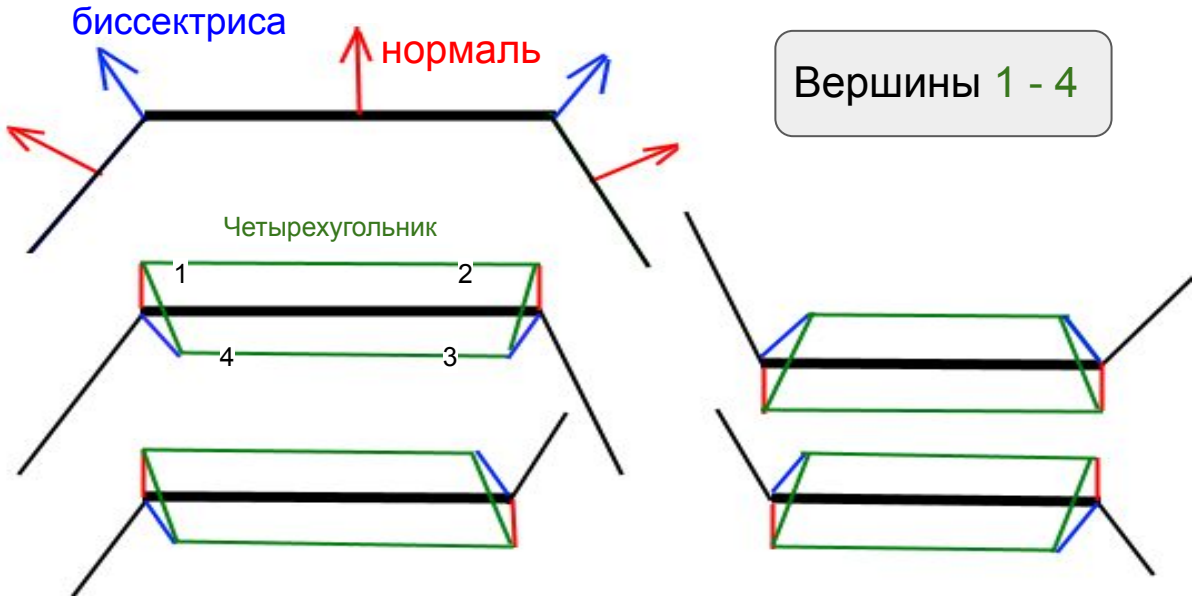


1. Перевести (A, B, prev, next, width) в **пиксельные координаты**
2. Вычислить нормали и биссектрисы
3. По номеру вершины вычислить её положение
4. Передать цвета и стиль интерполятору
5. Передать расстояние до прямой интерполятору

Пригодится :)

Vertex shader : как расположить вершины

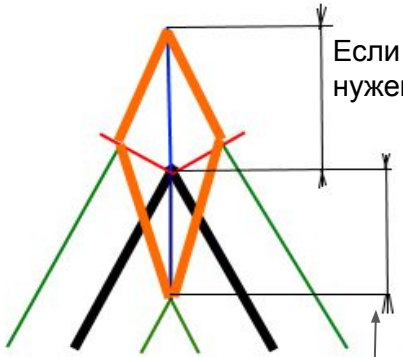
- Полных формул в этих слайдах нет, но есть в указанных статьях
- Есть несколько векторов о которых надо знать, все остальное это их + -, скалярное произведение (dot) и умножение на скаляр (*)
- Я пытаюсь это объяснить без взрыва мозга, потому что это не главная часть слайдов!



Vertex shader : как расположить вершины - 2

Вершины 5-9

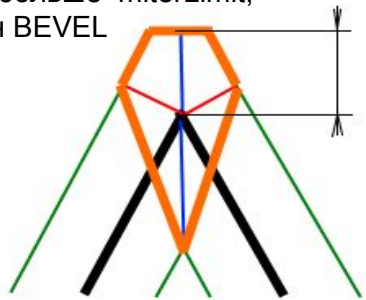
MITER



Если больше miterLimit, нужен BEVEL

BEVEL

bevel limit или round radius

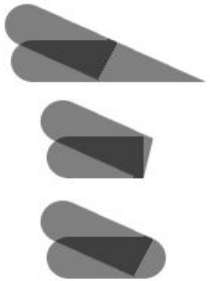


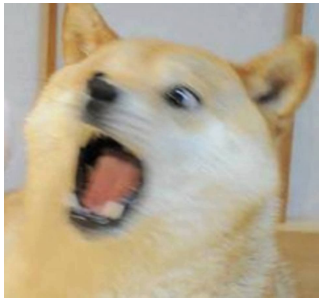
ROUND

как bevel, обрежем во фрагменте

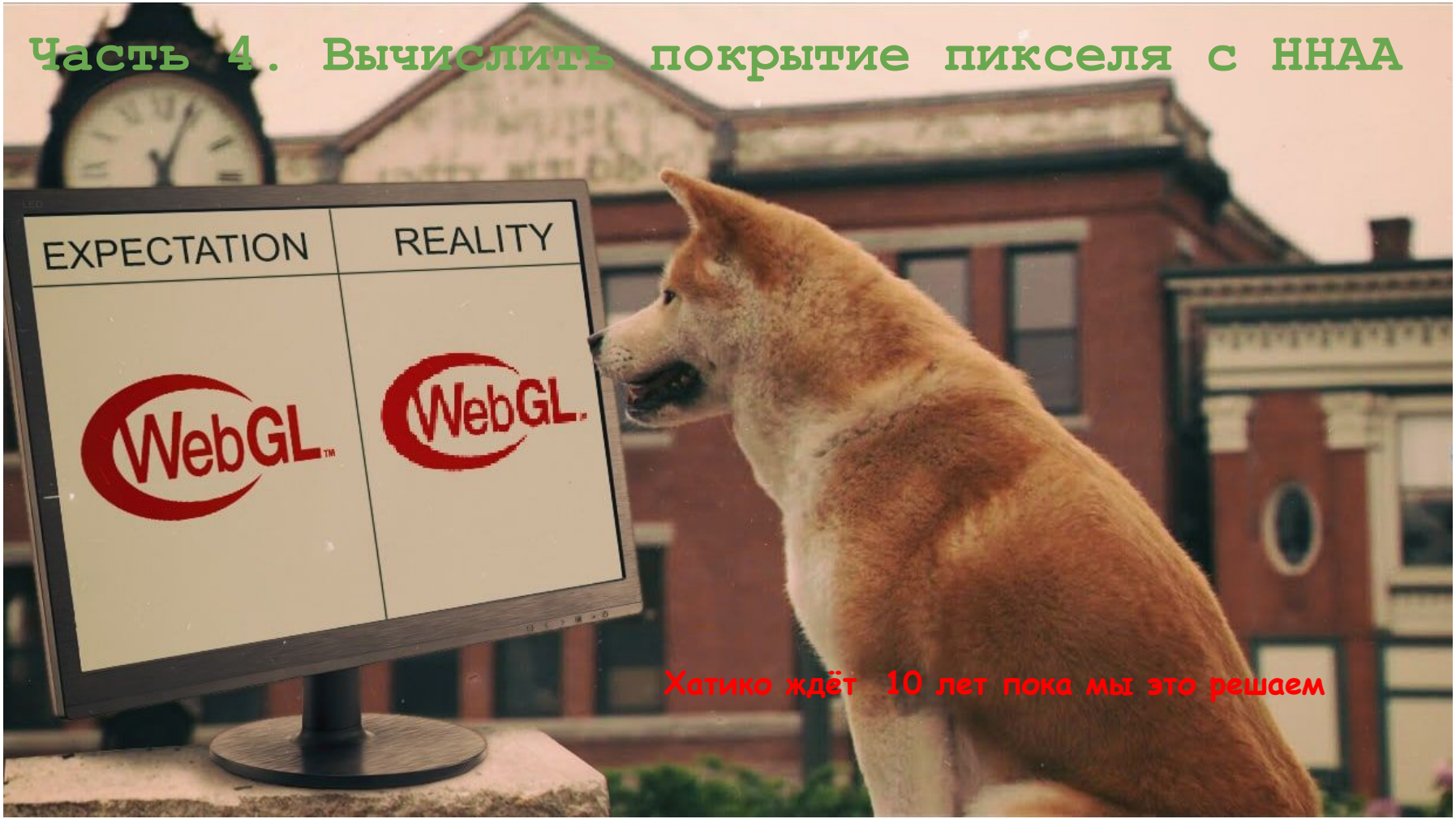
Если длиннее чем отрезок - пусть самопересекается:

FUBAR



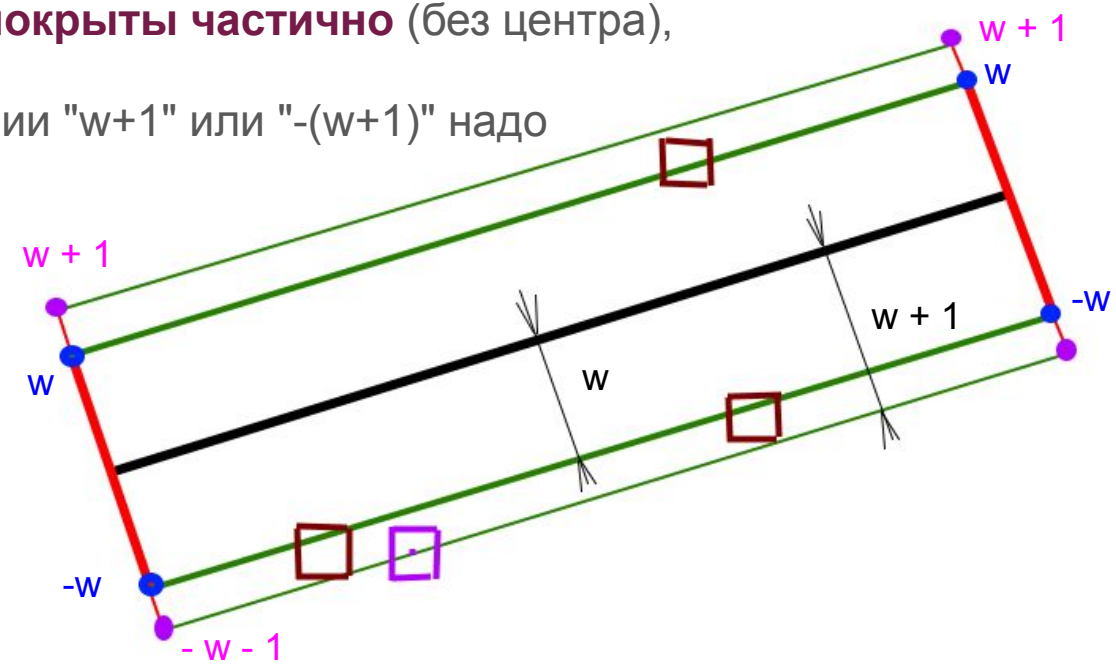


Часть 4. Вычислить покрытие пикселя с HHAA



Отрезок: inflation!

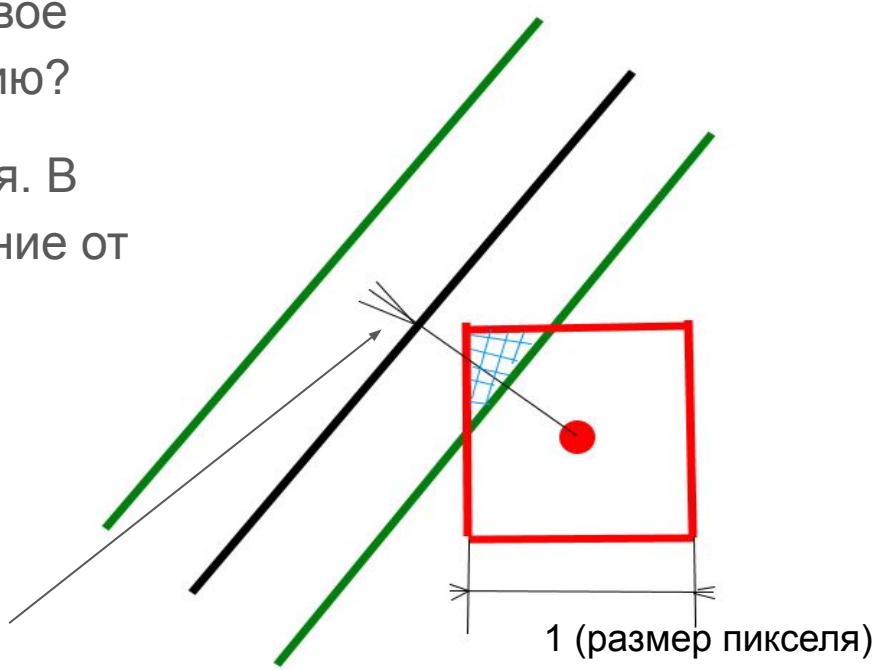
- w = lineWidth/2
- Для старой позиции вершин мы использовали сдвиг "w". Давайте использовать "w+1" для новой позиции
- Все пиксели которые были **покрыты частично** (без центра), теперь покрыты с центром
- Знаковое расстояние до линии "w+1" или "-(w+1)" надо передать в интерполятор



Отрезок: знаковое расстояние

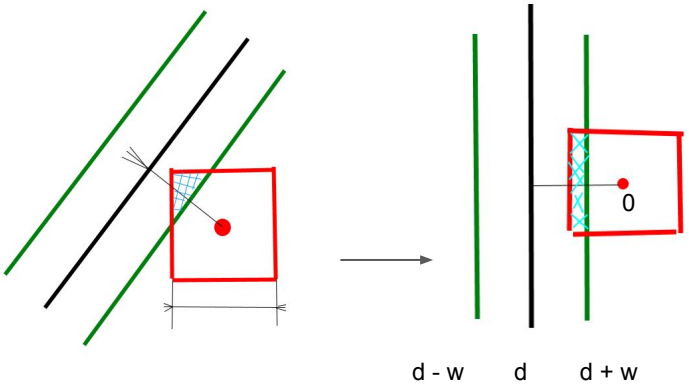
Что будет, если мы передадим знаковое расстояние до прямой в интерполяцию?

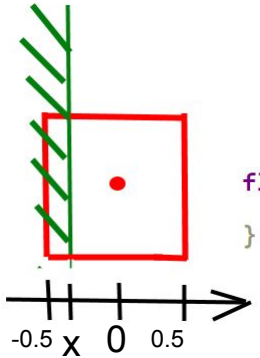
Ответ простой: это линейная функция. В fragment shader мы получим расстояние от центра пикселя до прямой



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

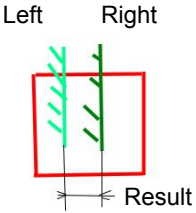





```
float pixelLine(float x) {
    return clamp(x + 0.5, 0.0, 1.0);
}
```

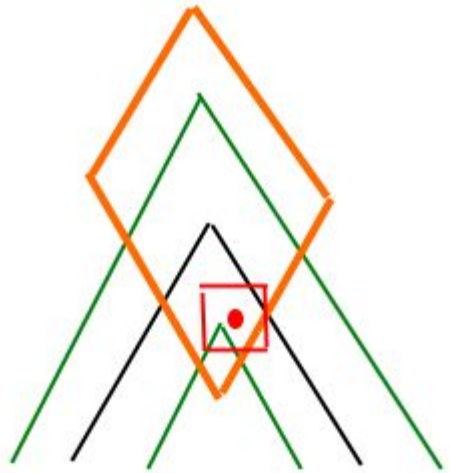
```
in vec4 vColor; //color style
in float vType; //type of segment/joint
in float d; //varying signed distance
in float w; //varying half-width

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;
}
```



Уголок (Miter): ой всё

- Miter это угол между двумя отрезками
- Линии ориентированы
- Геометрия немножко больше, из-за "w+1" вместо "w" в vertex
- Посчитаем в vertex расстояние до обеих прямых
- Поворот? Считаем что нету.

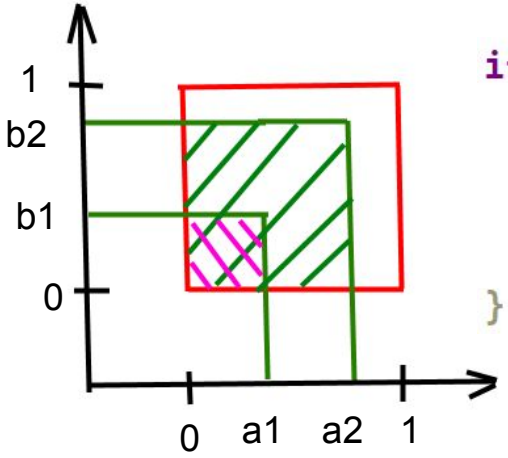


Уголок (Miter): подержите моё пиво

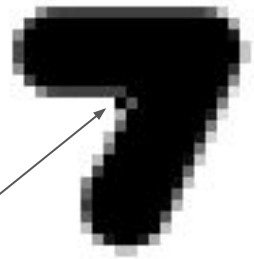
Используя pixelLine() можно посчитать a1,b1,a2,b2 - расстояния от угла до краев линии.

Большой квадрат имеет площадь a2*b2, мелкий a1*b1

Ответ - надо вычесть одно из другого: a2*b2 - a1*b1



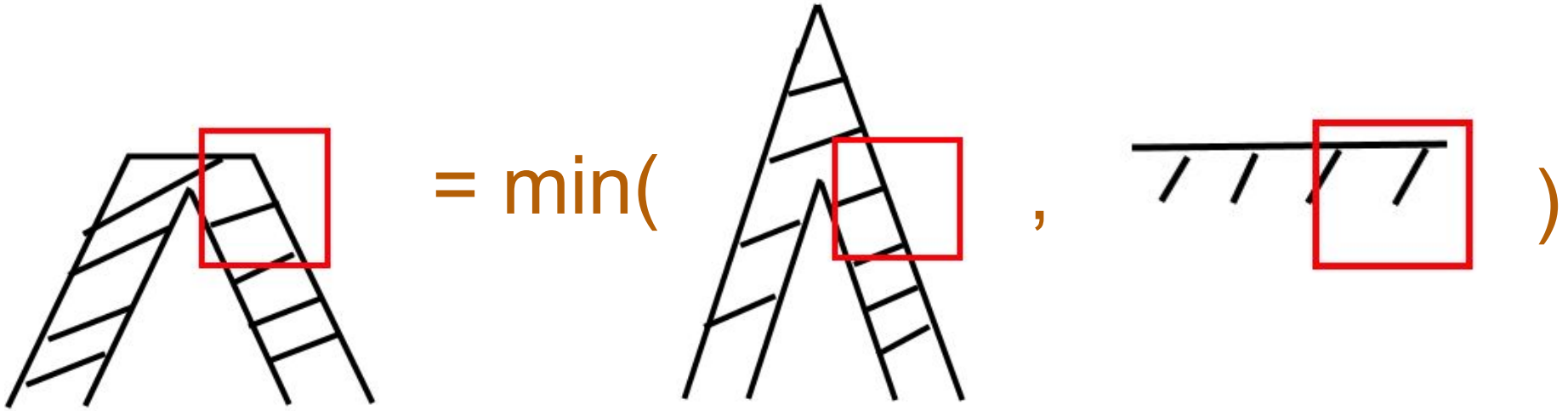

```
if (vType == 1.0) {
    float a1 = pixelLine(d1 - w);
    float a2 = pixelLine(d1 + w);
    float b1 = pixelLine(d2 - w);
    float b2 = pixelLine(d2 + w);
    result = a2 * b2 - a1 * b1;
}
```



Если не вычесть a1*b1 то можно получить волосы подмышкой

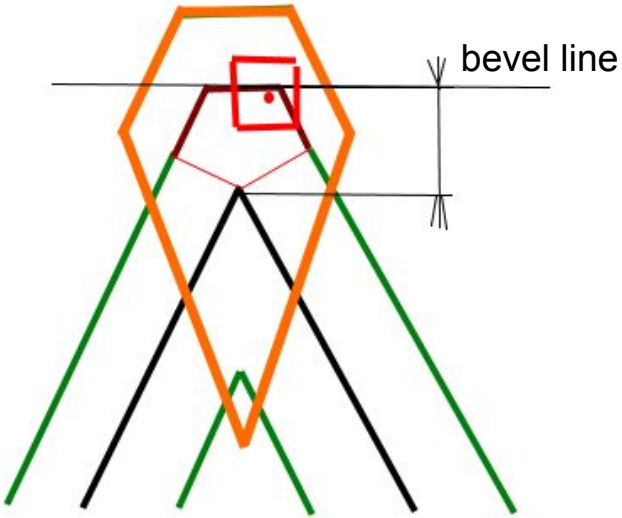
Фаска (Bevel)

- Как MITER но с дополнительной отсечкой
- надо пересечь две площади
- пересечение где-то между min(S1, S2) и S1 * S2



Фаска (Bevel) : остановите это безумие!

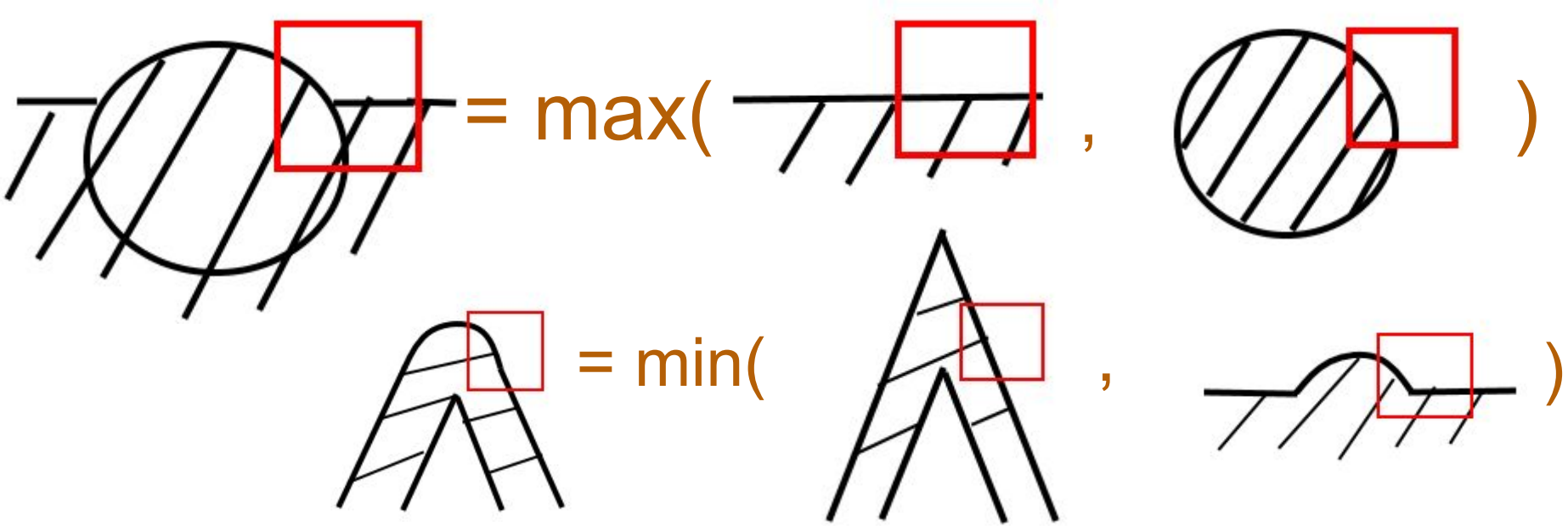
- Как MITER но с дополнительной pixelLine
- Взять результат от MITER и пересечь с полуплоскостью BEVEL помощью функции min()
- Точный результат с помощью тригонометрии, да ну нафиг
- Допустим d3 это расстояние от центра пикселя до bevel-линии




```
if (vType == 2.0) {
    float a1 = pixelLine(d1 - w);
    float a2 = pixelLine(d1 + w);
    float b1 = pixelLine(d2 - w);
    float b2 = pixelLine(d2 + w);
    float result_miter = a2 * b2 - a1 * b1;
    float result_bevel = pixelLine(d3);
    result = min(result_miter, result_bevel);
}
```

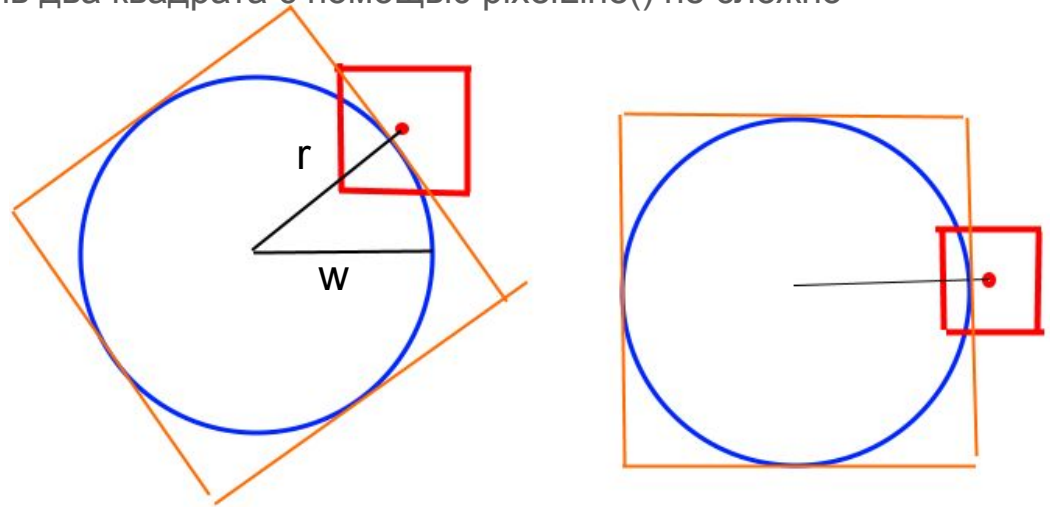
Закругление (Round)

- Соединим bevel полуплоскость с окружностью
- Пересечение где-то между max(S1, S2) и 1 - (1 - S1) * (1 - S2)

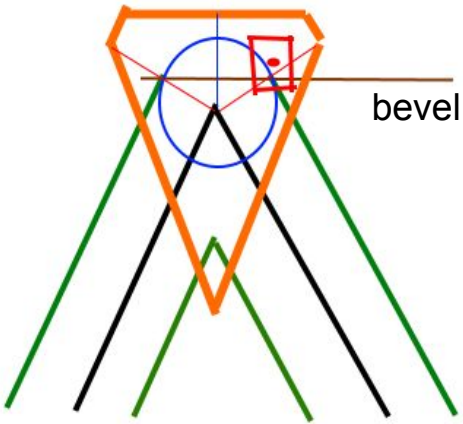


Закругление (Round) : армейские технологии

- Как пересечь окружность?
- Пусть "circle" это координаты центра пикселя относительно центра круга (можно проинтерполировать)
- Посчитаем r, расстояние до центра
- Представим что круг - это квадрат (Pi = 4, в военное время)
- Пересечь два квадрата с помощью pixelLine() не сложно



Закругление (Round) : ну всё




```
if (vType == 3.0) {
    float a1 = pixelLine(d1 - w);
    float a2 = pixelLine(d1 + w);
    float b1 = pixelLine(d2 - w);
    float b2 = pixelLine(d2 + w);
    float result_miter = a2 * b2 - a1 * b1;
    float result_bevel = pixelLine(d3);

    float r = length(circle.xy);
    float circle_hor = pixelLine(w + r) - pixelLine(-w + r);
    float circle_vert = min(w * 2.0, 1.0); //height of square
    float result_circle = circle_hor * circle_vert;

    result = min(result_miter, max(result_bevel, result_circle));
}
```

Идеальный пиксель

Что если мы всё-таки можем учесть вращение пикселя?

При анимации линии, +-4% альфы это большая неточность

Пусть **A** = max(abs(**n**.x), (**n**.y)), **B** = min(abs(**n**.x), abs(**n**.y)) где **n** это нормаль

```
float pixelLine(float x, float A, float B) {
    float y = abs(x), s = sign(x);
    if (y * 2.0 < A - B) {
        return 0.5 + s * y / A;
    }
    y -= (A - B) * 0.5;
    y = max(1.0 - y / B, 0.0);
    return (1.0 + s * (1.0 - y * y)) * 0.5;
    //return clamp(x + 0.5, 0.0, 1.0);
}
```

HHAA

Scanline

Часть 6. Решение есть, что дальше?

PixiJS Graphics Smooth

PixiJS библиотека graphics-smooth это замена "PIXI.Graphics". Все пользователи PixiJS могут вырубить "antialias:false" для WebGL.

pixi-candles это лайтовая версия, чтобы нарисовать одну линию - её можно портировать в другие рендереры.

Обе библиотеки это первые реализации **HHAA**

Shader code: 350 LoC для vertex и 60 для fragment

FoundryVTT , один из основных контрибьюторов PixiJS взяли её в первую же неделю. Сейчас всё в продакшне.

Проблема тонких линий

Всё началось с проблемы с CAD software

Дополнительные параметры стиля

Line Scale Mode

- Adobe Flash Graphics
- Specifies how to scale line width depending on transform
- Line width is the same vertically and horizontally
- Very useful for scaling plots!
- Many requests in PixiJS issues

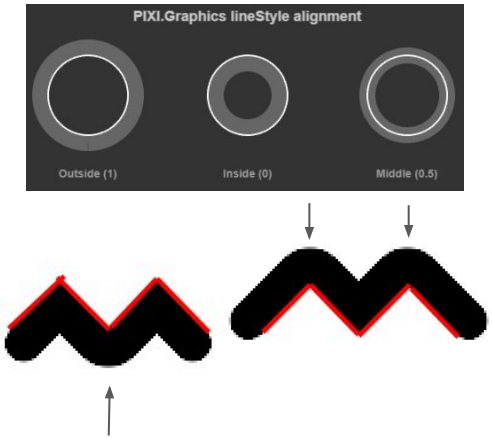
vertical horizontal

$$\begin{bmatrix} a & c & t_x \\ b & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

```
float avgScale = 1.0;
if (scaleMode > 2.5) {
    avgScale = length(translationMatrix * vec3(1.0, 0.0, 0.0));
} else if (scaleMode > 1.5) {
    avgScale = length(translationMatrix * vec3(0.0, 1.0, 0.0));
} else if (scaleMode > 0.5) {
    vec2 avgDiag = (translationMatrix * vec3(1.0, 1.0, 0.0)).xy;
    avgScale = sqrt(dot(avgDiag, avgDiag) * 0.5);
}
lineWidth *= 0.5 * avgScale;
```

Line Alignment

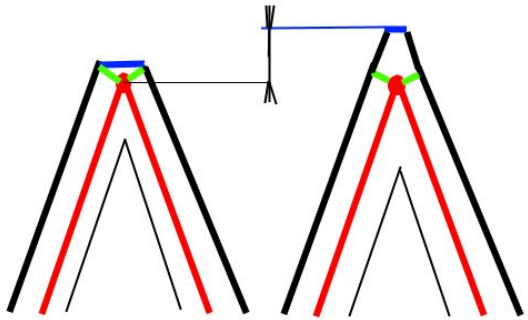
- Pixi Graphics
- Shifts the line
- Added by Mat Groves
- Cannot implement in canvas2d because it affects joins!
- Outline shapes inside or outside, maybe with gradient



Bevel Limit WIP

- Bevel sometimes looks bad on plots
- Bevel looks bad with alignment=0
- If equal to miter limit, smooth transition between miter and bevel
- Extra shift of bevel line

bevelLimit * lineWidth/2



Batching & Performance

Attributes:

- Color

Style:

- Width
- Texture
- Gradient WIP
- Dash WIP

Extra:

- MiterLimit WIP
- BevelLimit WIP
- Alignment

My general approach on batching:

How to separate style into attributes, style and extra style? Its up to you.

Style & Extra can be stored in UBO or Data Textures, just include their ID in attribute to reference it in vertex shader.

Animated props are better in styles, that way you can animate single uniform or UBO without re-uploading the buffer.

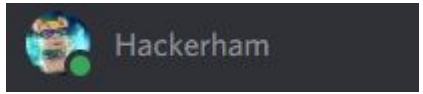
Props that are different for many objects should go in attributes, because UBO is limited.

HHAA successfully works in production, I did not do any benchmarks yet, it's just looks fast.

If you want a benchmark, please help me by posting a demo :)

Спасибо за просмотр!

Просто опишите свою проблему на гитхаб
PixiJS discussions
или в телеграм
https://t.me/webgl_ru
Или просто упомяните WebGL в твиттере
https://twitter.com/ivanpopelyshev



Вакансия JS джуна для crazypanda.ru

Оформление слайдов кроликами:
https://www.youtube.com/channel/UCZfYIIsllUFyfmLodIpzU0g