Apple Metal

optimisation

George Ostrobrod

2. Рисуем остаток совы

Why do we need optimisation?

Example task

very naive implementation

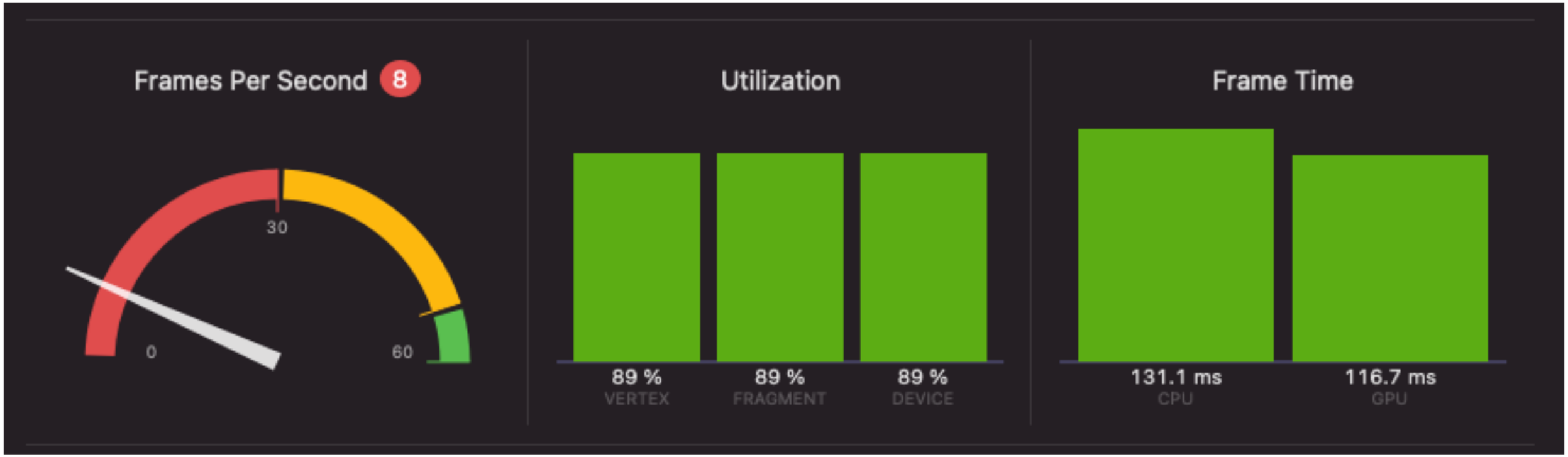
- 256 particles
- each sprite - an image with transparent parts
- each sprite’s transform is updated each frame
- target framebuffer size: 1242 x 1242px

Example task

very naive implementation

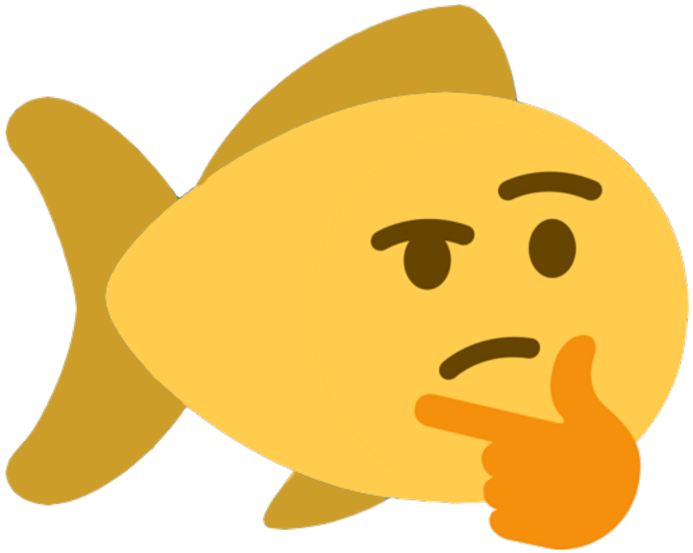
Example task

very naive implementation




THIS IS FINE.

Common sense vs Tools

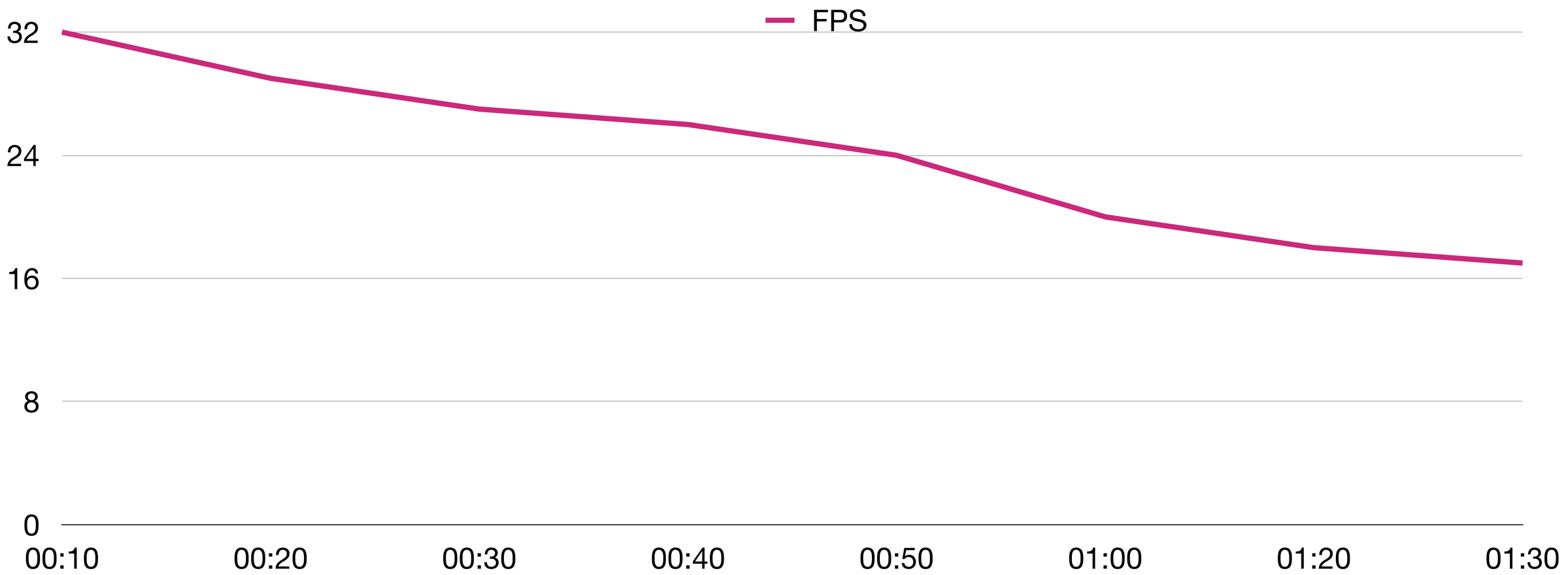


do not even try to optimise `sin` and `cos` in

$$\sin^2(t) + \cos^2(t) = 1$$

Throttling

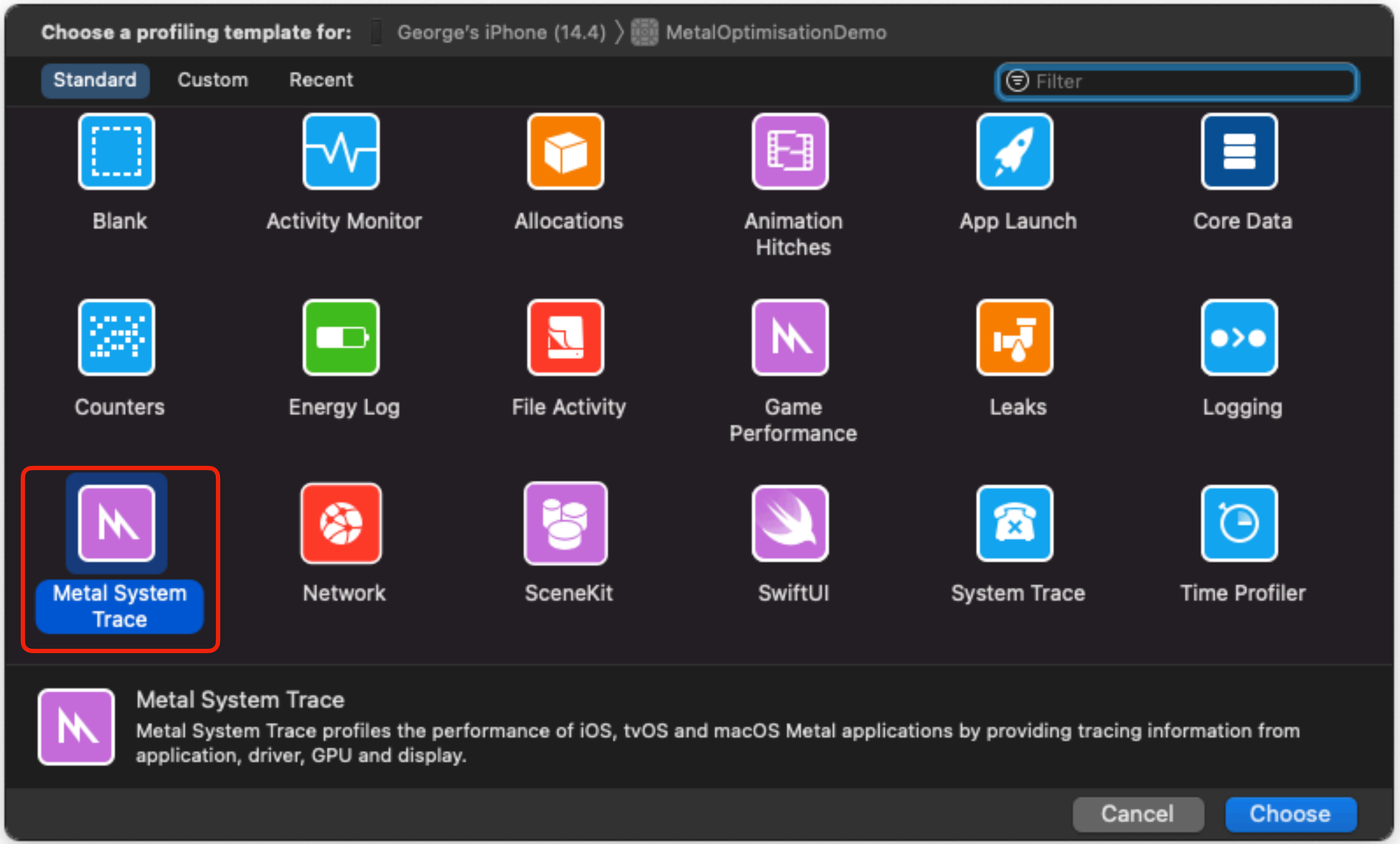
iOS is smart, too smart



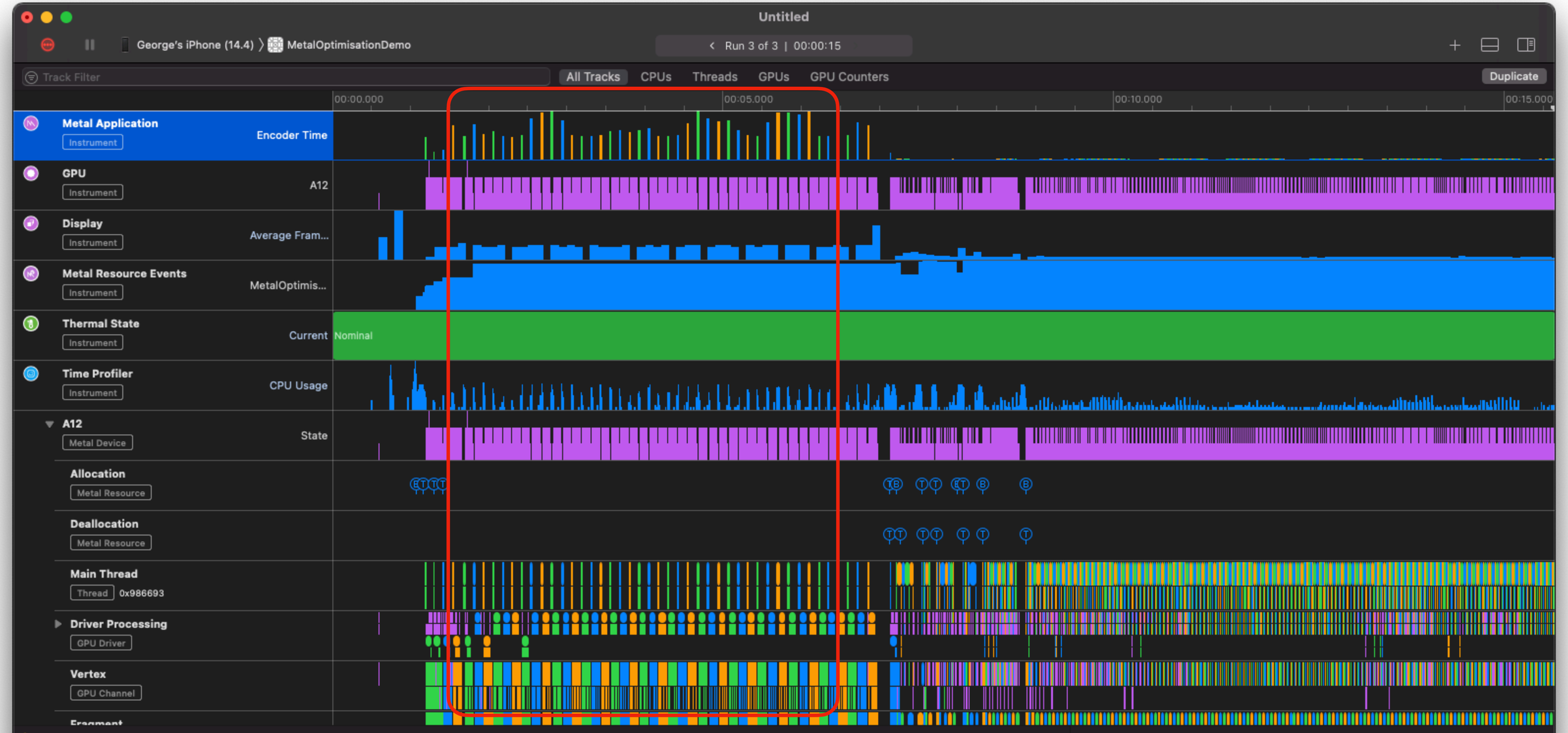
Profiling

Instruments

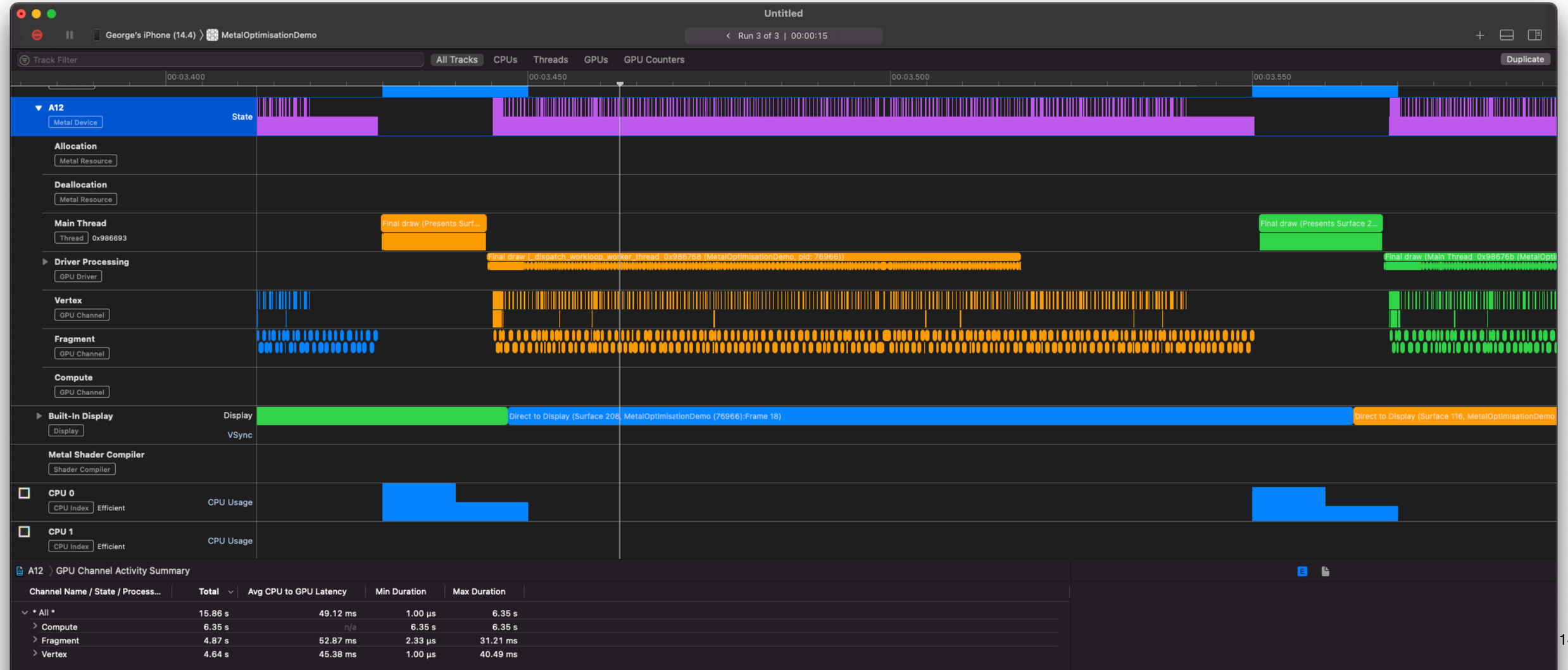
Metal System Trace



Metal System Trace



Metal System Trace



Rendering

Useless synchronisation

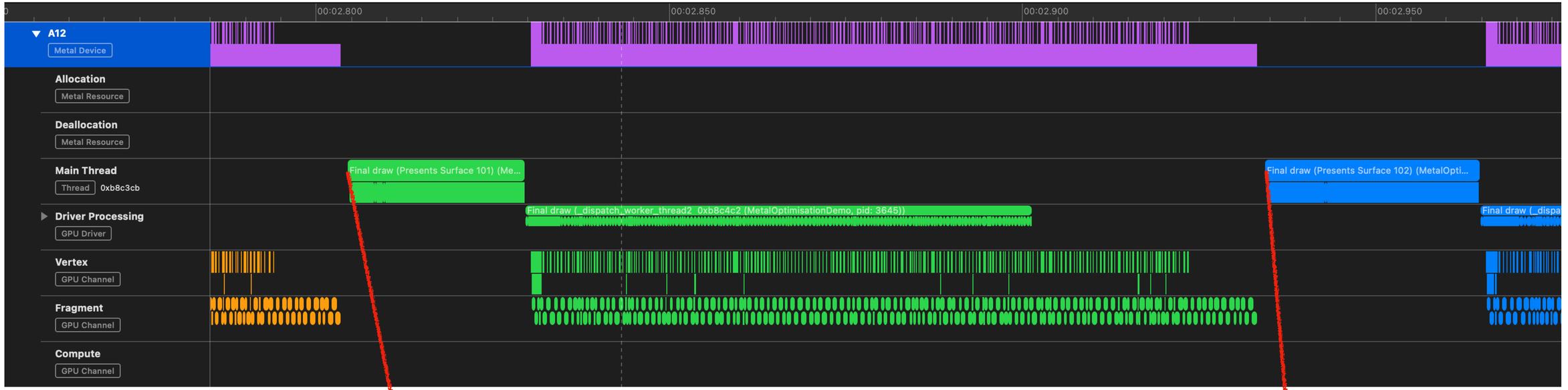
render and wait until buffer started running

```
// ...
    commandBuffer.waitUntilScheduled()
// ...
```

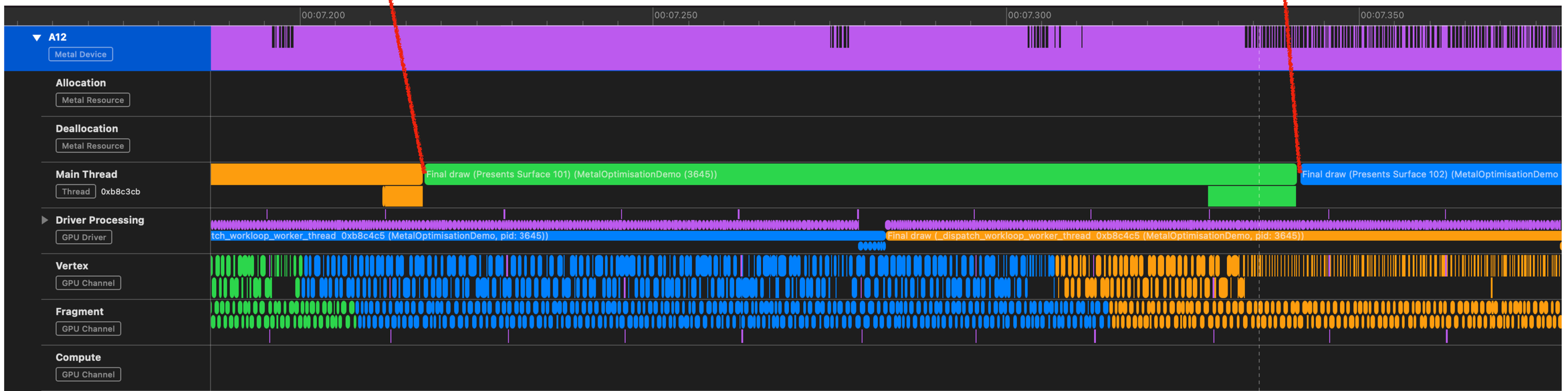
render and wait until whole command buffer is completed

```
// ...
    commandBuffer.waitUntilCompleted()
// ...
```

Useless synchronisation



7 FPS



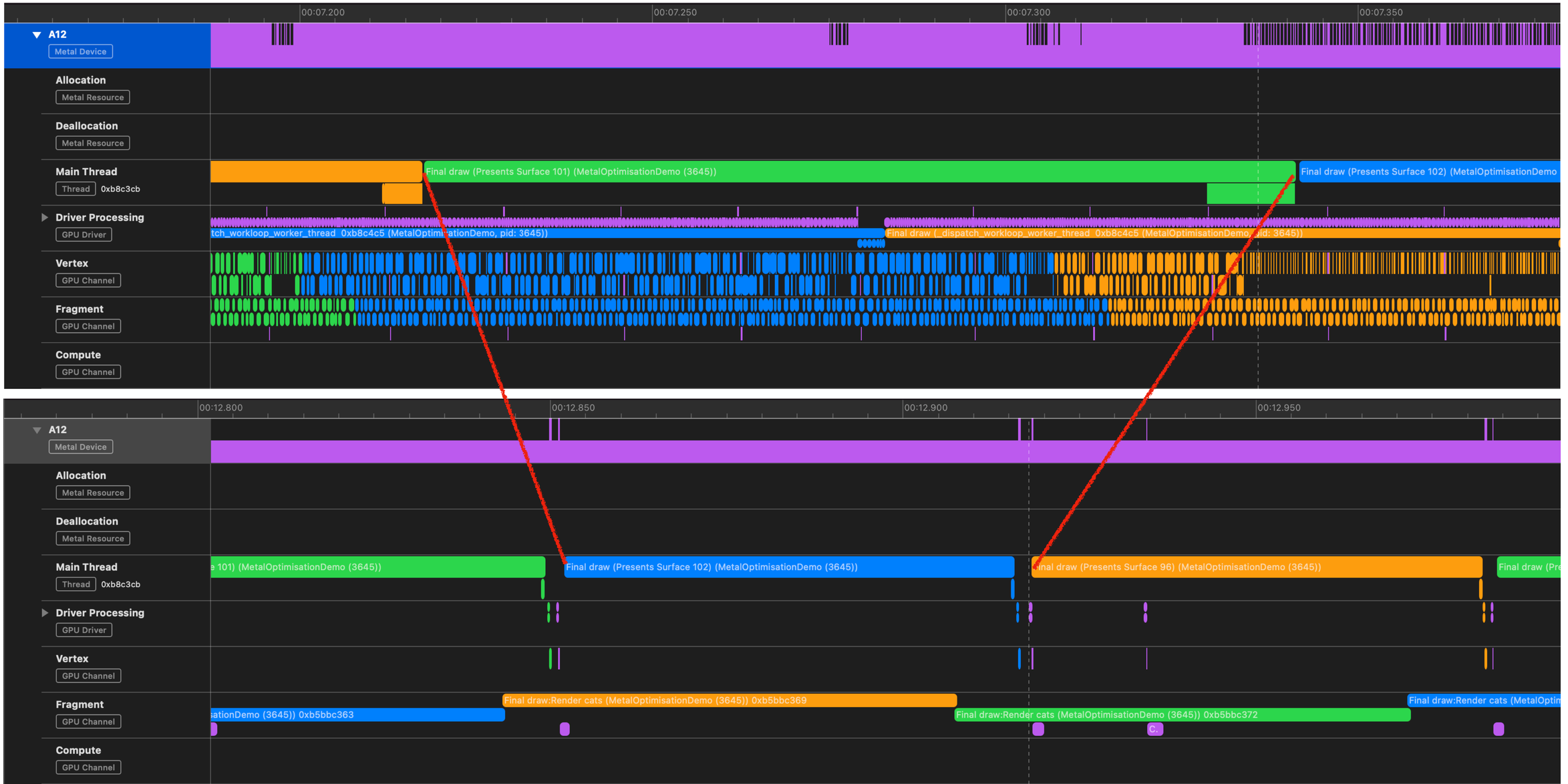
9 FPS

Number of encoders

```
MTLRenderPassDescriptor* descriptor =
   self.mtkView.currentRenderPassDescriptor;
descriptor.colorAttachments[0].clearColor =
   (MTLClearColor){0, 0.2, 0.2, 1};
descriptor.colorAttachments[0].loadAction =
   MTLLoadActionClear;

for (int i = 0; i < MaxInstances; ++i) {
    id <MTLRenderCommandEncoder> encoder =
       [commandBuffer renderCommandEncoderWithDescriptor:
      descriptor];

    // . . .

    [encoder drawPrimitives:MTLPrimitiveTypeTriangleStrip
              vertexStart:0
              vertexCount:4];
    [encoder endEncoding];

    descriptor.colorAttachments[0].loadAction =
      MTLLoadActionLoad;
}
```

```
MTLRenderPassDescriptor* descriptor =
   self.mtkView.currentRenderPassDescriptor;
descriptor.colorAttachments[0].clearColor =
   (MTLClearColor){0, 0.2, 0.2, 1};
descriptor.colorAttachments[0].loadAction =
   MTLLoadActionClear;

id <MTLRenderCommandEncoder> encoder =
   [commandBuffer renderCommandEncoderWithDescriptor:
   descriptor];

// . . .

for (int i = 0; i < MaxInstances; ++i) {
    // . . .

    [encoder
drawPrimitives:MTLPrimitiveTypeTriangleStrip
             vertexStart:0
             vertexCount:4];
}

[encoder endEncoding];
```

Number of encoders

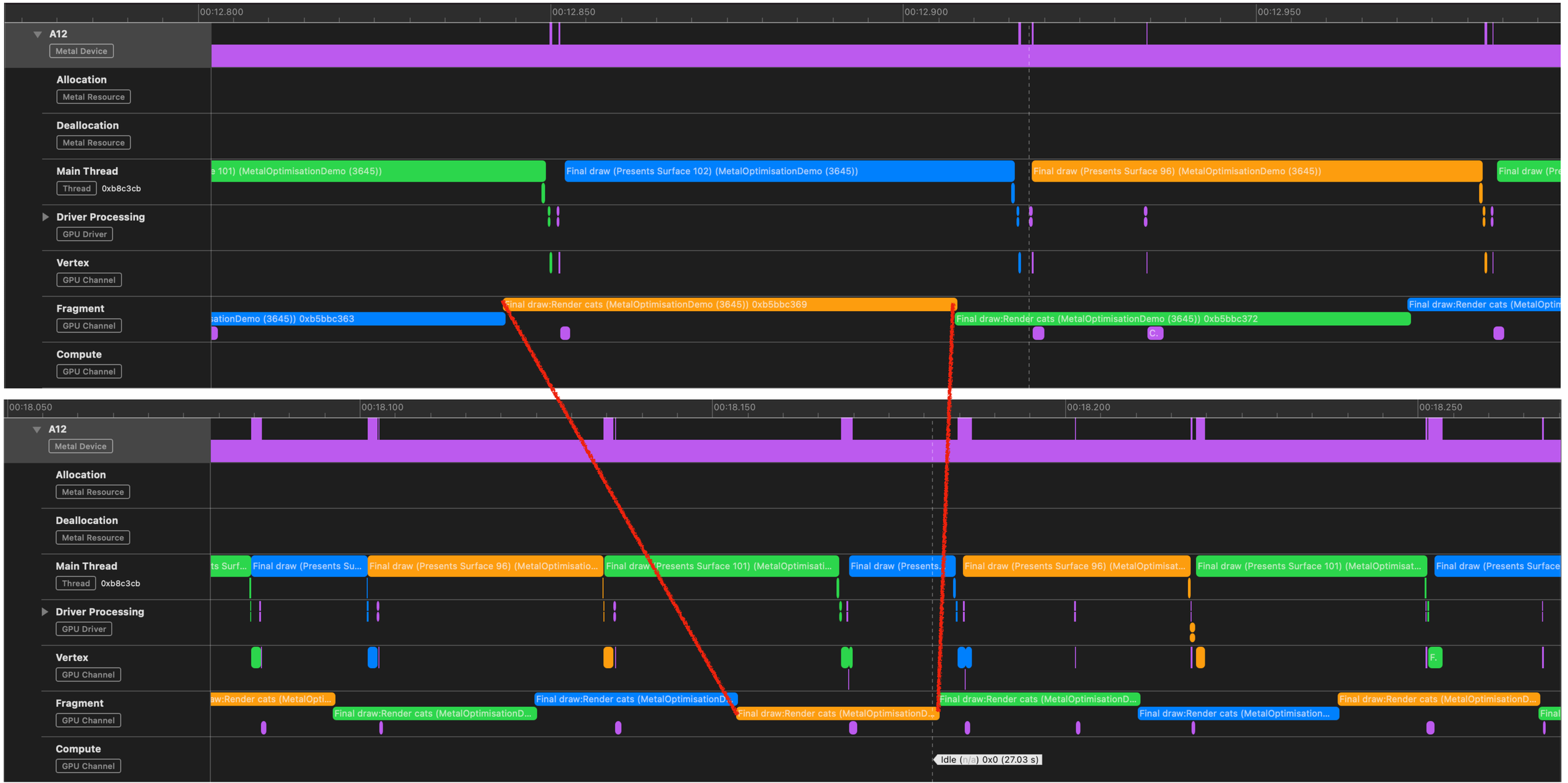


9 FPS

14 FPS

Fillrate

Fillrate

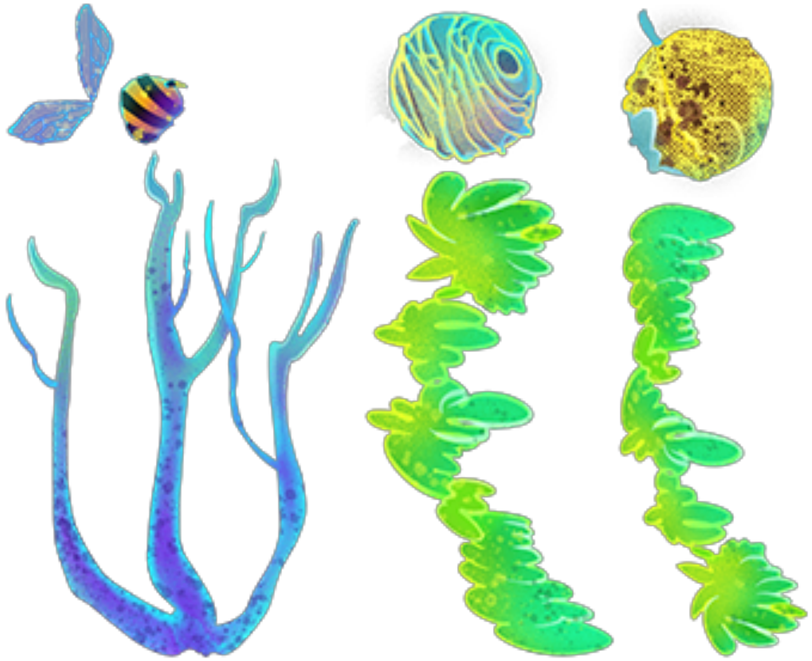


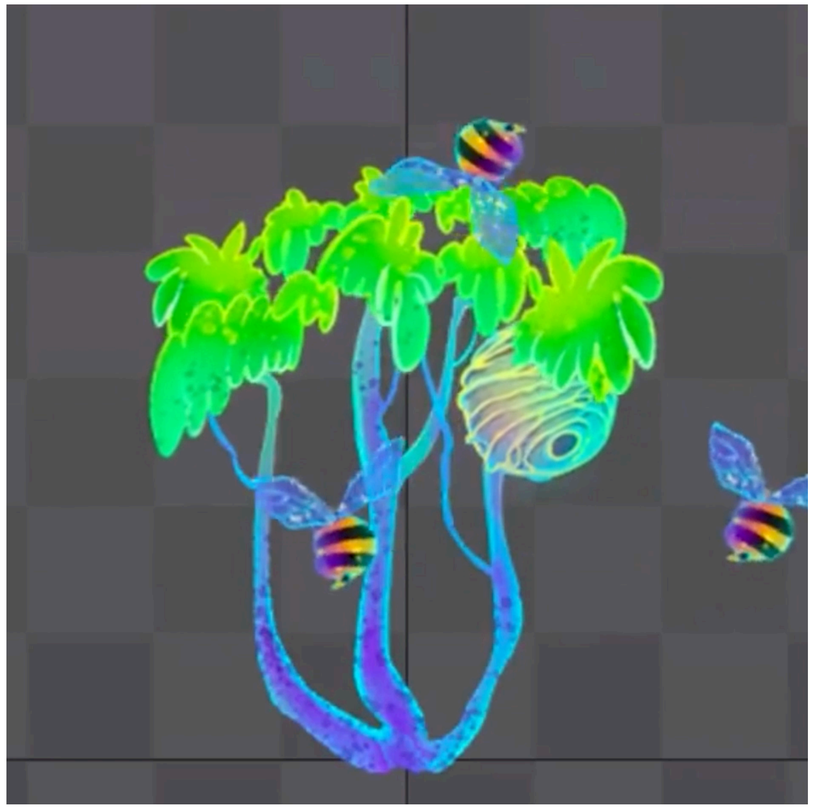
14 FPS

37 FPS

Number of draw calls

Texture atlases





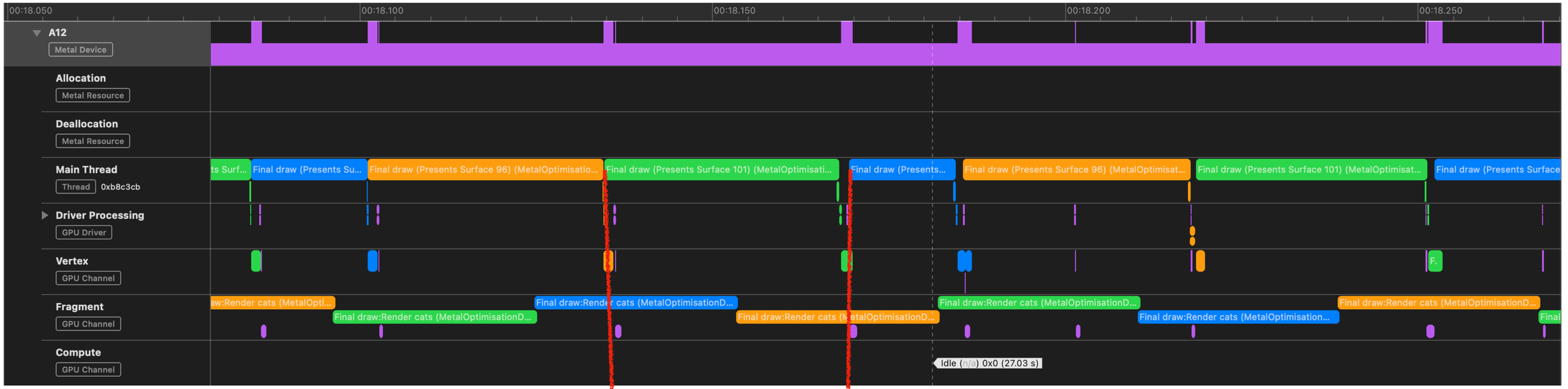
Number of draw calls

Instances

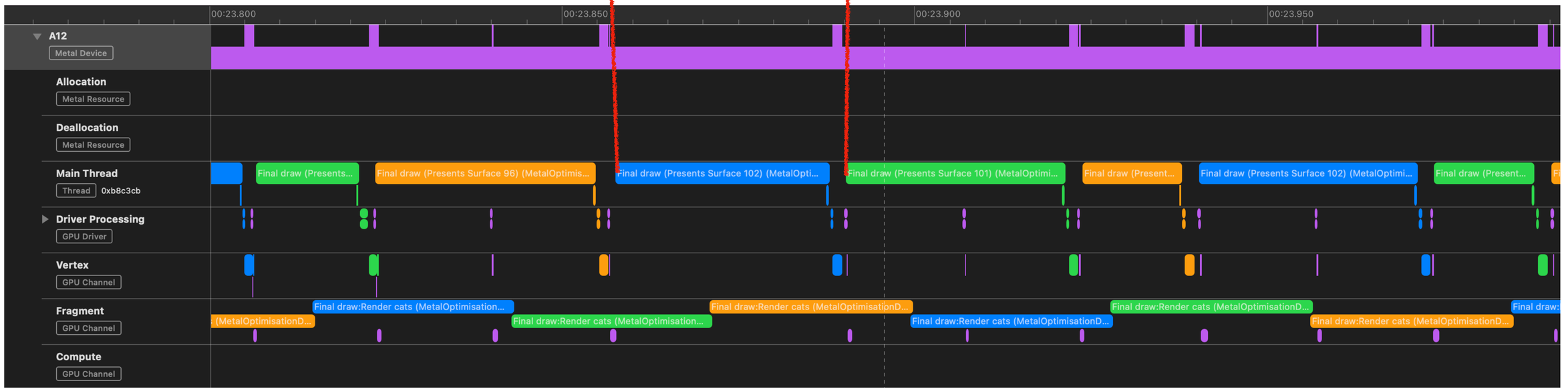
```
for (int i = 0; i < MaxInstances; ++i) {
    [encoder setVertexBytes:&i length:sizeof(int) atIndex:1];
    [encoder drawIndexedPrimitives:MTLPrimitiveTypeTriangle
                        indexCount:_mesh.indices_count
                         indexType:MTLIndexTypeUInt16
                       indexBuffer:_bufferIndices
                 indexBufferOffset:0];
}
```

```
[encoder drawIndexedPrimitives:MTLPrimitiveTypeTriangle
                    indexCount:_mesh.indices_count
                     indexType:MTLIndexTypeUInt16
                   indexBuffer:_bufferIndices
             indexBufferOffset:0
                 instanceCount:_instances];
```

Number of draw calls



37 FPS



39 FPS

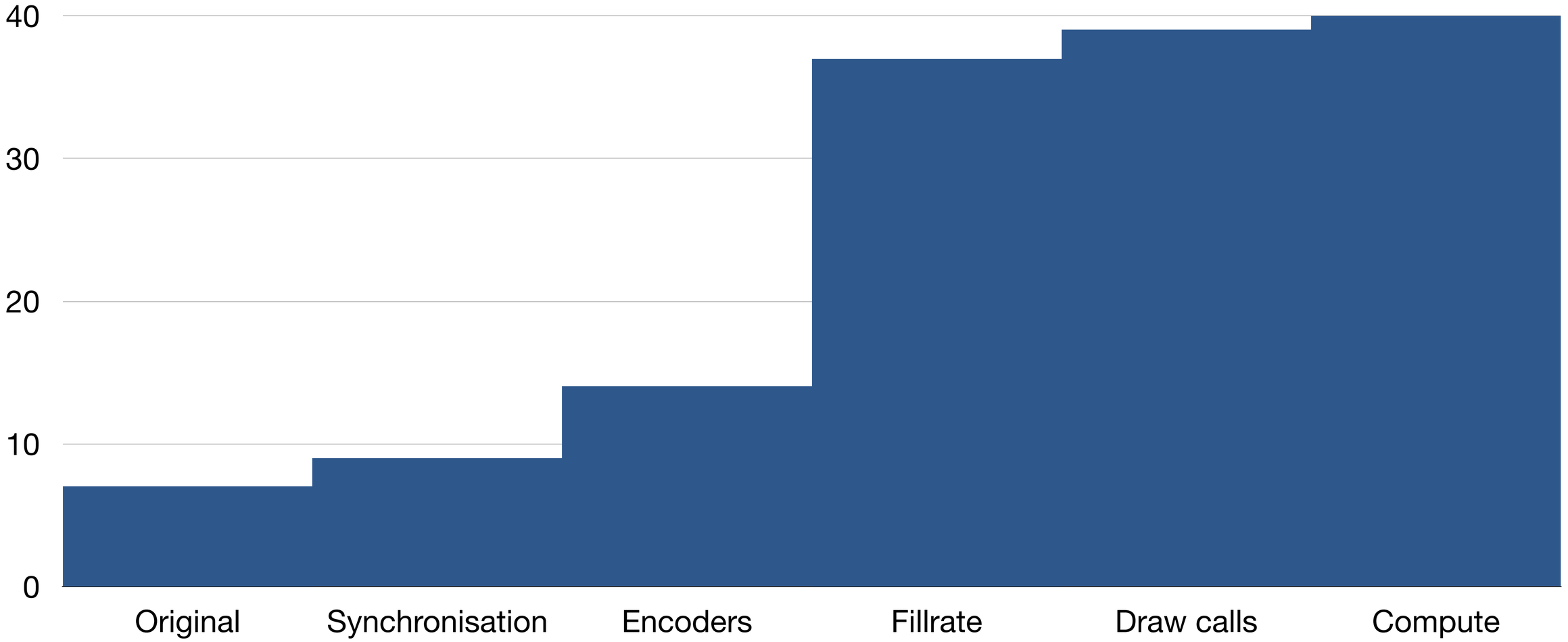
Update state on GPU

39 FPS

40 FPS

Summary (FPS)

Task 1: Render



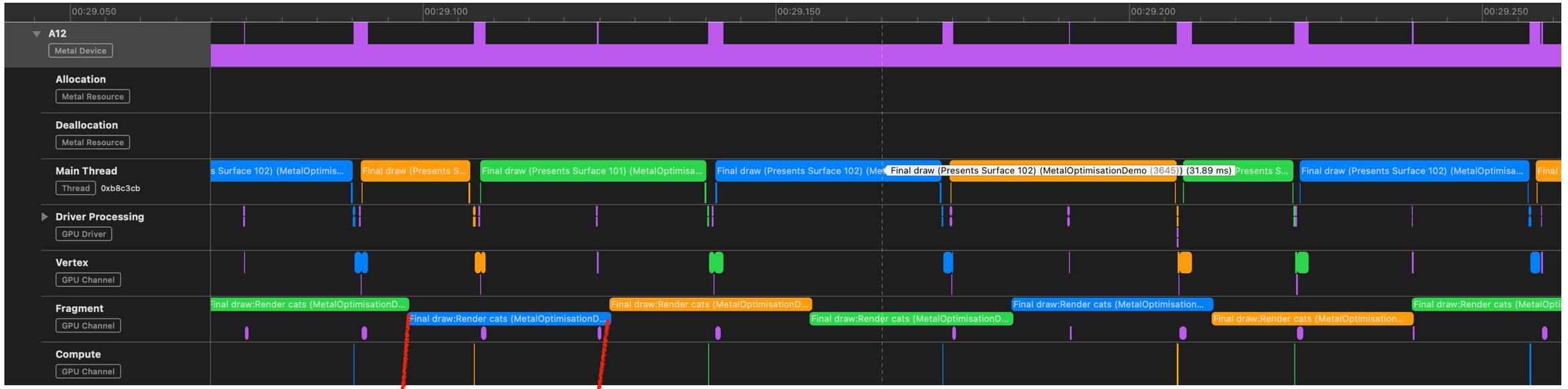
Shaders

Sampling: `nearest` vs `linear`

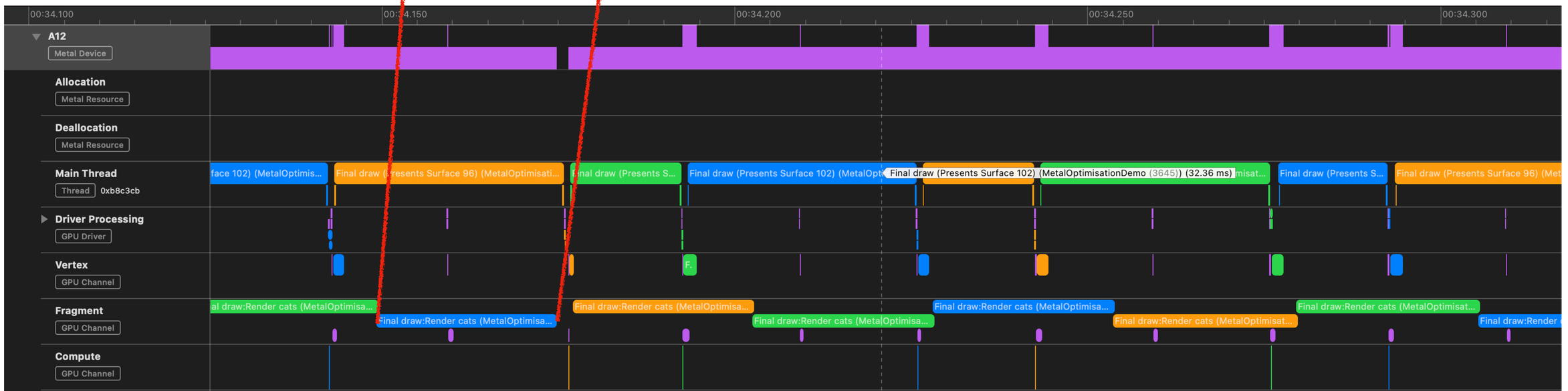
```
// ...
    constexpr sampler imageSampler(address::clamp_to_zero, filter::linear);
// ...
```

```
// ...
    constexpr sampler imageSampler(address::clamp_to_zero, filter::nearest);
// ...
```

Sampling: `nearest` vs `linear`

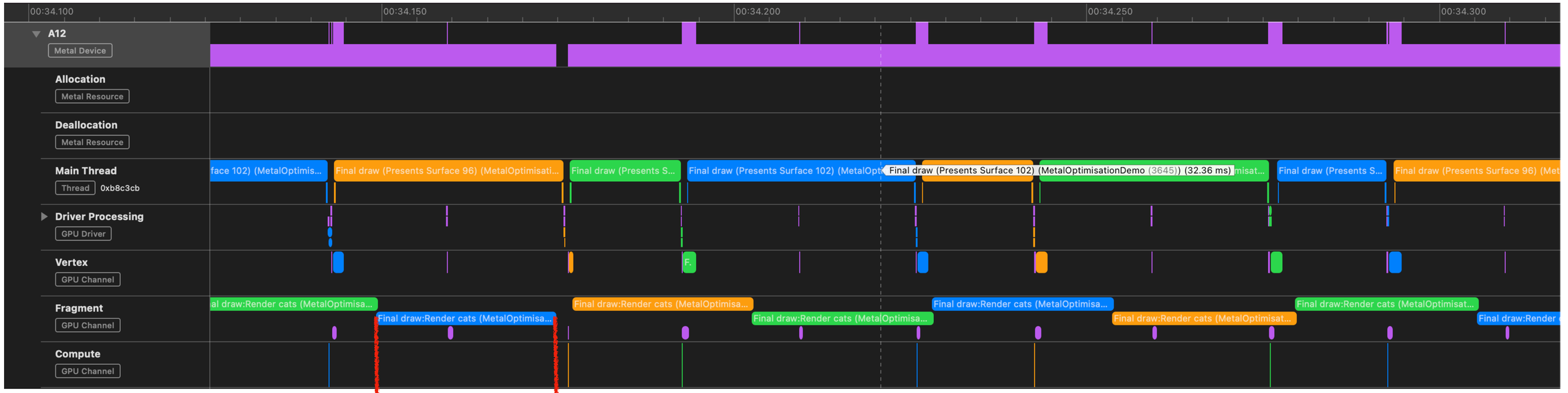


40 FPS

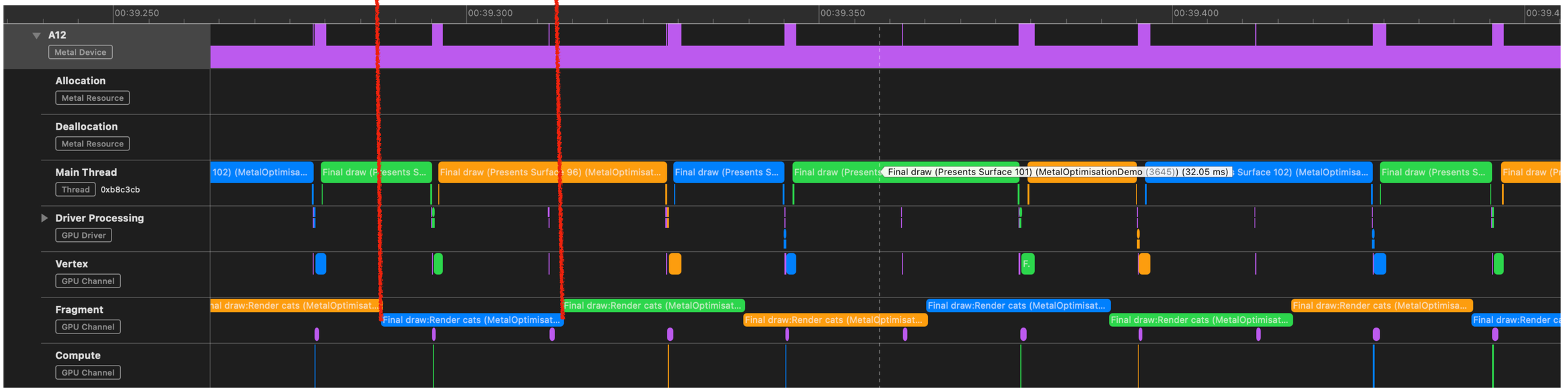


41 FPS

`half` vs `float`



41 FPS



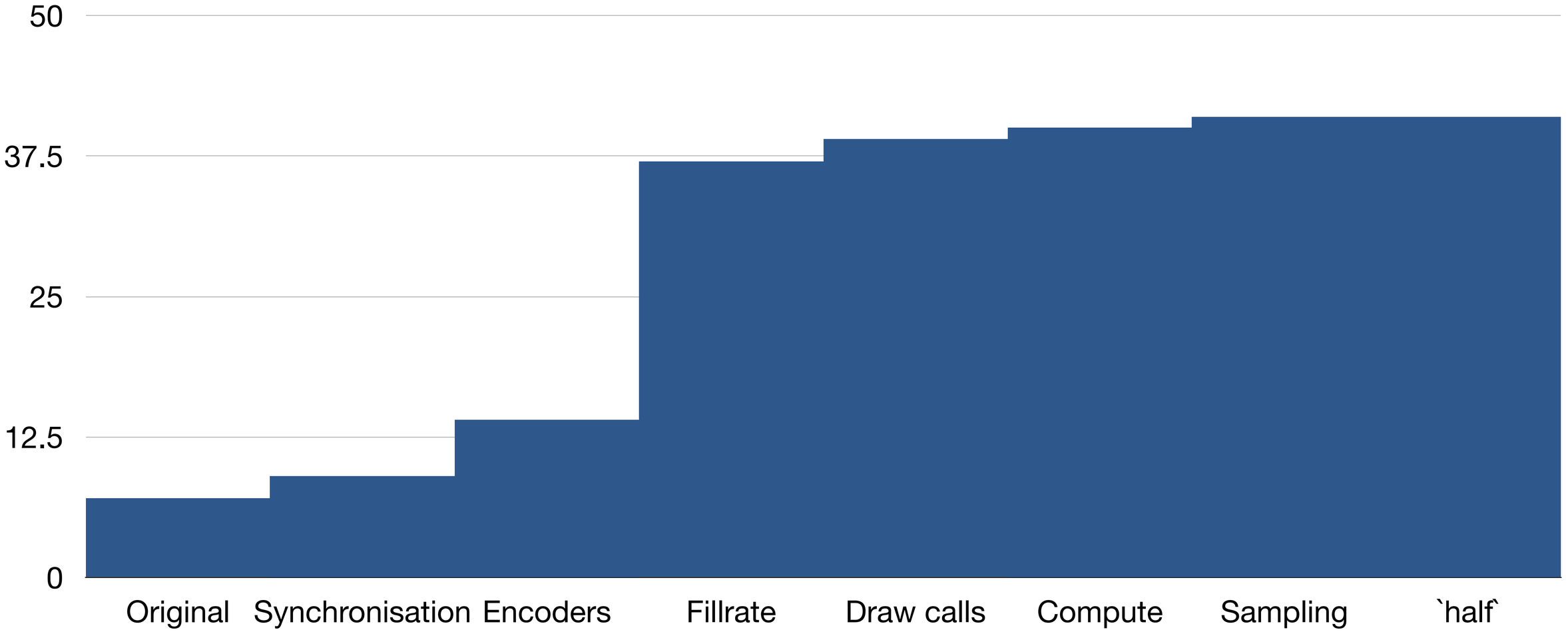
41 FPS

`half` vs `float`

```
fragment float4 fshRenderer07(ColorInOut in [[stage_in]],
                              float4 back [[ color(0) ]],
                              texture2d<float> image [[ texture(0) ]])
{
  constexpr sampler imageSampler(address::clamp_to_zero, filter::linear);

  float4 res = image.sample(imageSampler, in.texCoord);

  res.rgb = mix(back.rgb, res.rgb, res.a);

  return res;
}
```

Summary (FPS)

Task 1: Render + Shader



Summary

Task 1: Render + Shader

Another example

heavy shader

- two input textures
  - with transparent parts
  - without transparency
- sharpen
- color transform (RGB-HSV-RGB)
- displacement transform
- target texture size: 8192 x 8192 px

Another example

heavy shader

10 FPS

`half` vs `float`

```
static float3 rgb2hsv(float3 c) {
   float M = max(c.x, max(c.y, c.z));
   float m = min(c.x, min(c.y, c.z));
   float C = M - m;

   float h = 0;
   if (C > 0) {
      if (c.r == M) {
         h = fmod((c.g - c.b) / C, 6.0);
      } else if (c.g == M) {
         h = (c.b - c.r) / C + 2;
      } else {
         h = (c.r - c.g) / C + 4;
      }
   }
   h /= 6.0;

   float v = M;

   float s = 0;
   if (v > 0) {
      s = C / v;
   }

   return clamp(float3(h, s, v), 0.0, 1.0);
}
```

```
static half3 rgb2hsv(half3 c) {
   half M = max(c.x, max(c.y, c.z));
   half m = min(c.x, min(c.y, c.z));
   half C = M - m;

   half h = 0.0h;
   if (C > 0.0h) {
      if (c.r == M) {
         h = fmod((c.g - c.b) / C, 6.0h);
      } else if (c.g == M) {
         h = (c.b - c.r) / C + 2.0h;
      } else {
         h = (c.r - c.g) / C + 4.0h;
      }
   }
   h /= 6.0h;

   half v = M;

   half s = 0.0h;
   if (v > 0.0h) {
      s = C / v;
   }

   return clamp(half3(h, s, v), 0.0h, 1.0h);
}
```

`half` vs `float`

10 FPS

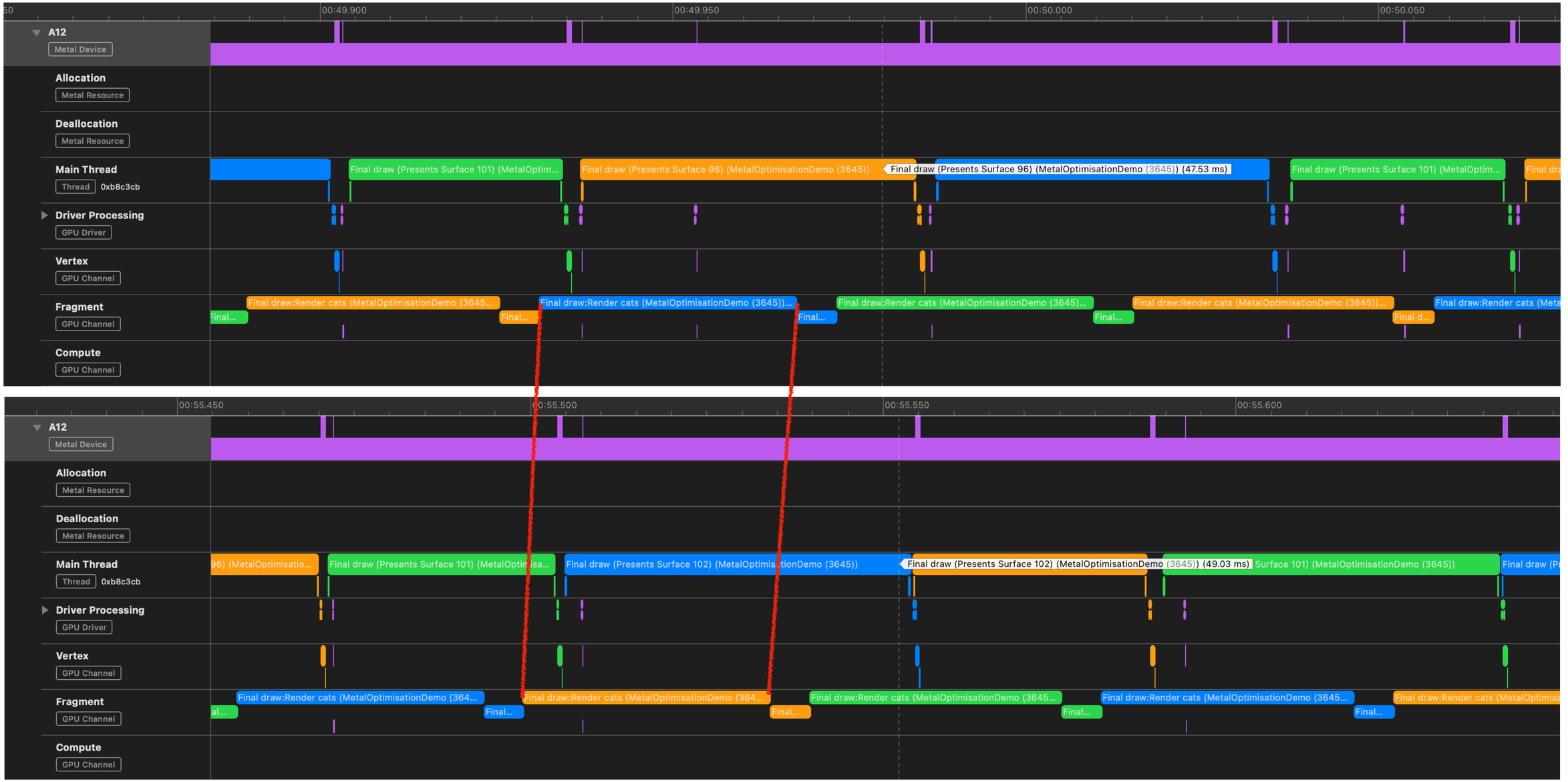
13 FPS

Reduce number of operations

```
half4 dst_pixels[9];
for (char i = 0; i < 9; ++i) {
   float2 dp = pixel * float2(i % 3 - 1, i / 3 - 1);
   dst_pixels[i] = image1.sample(image1Sampler, uv1 + dp);
}
half4 outline = 8 * dst_pixels[4];
for (char i = 0; i < 9; ++i) {
   if (i == 4) {
      continue;
   }
   outline -= dst_pixels[i];
}
```

```
half4 dst_pixels[9] = {
   image1.sample(image1Sampler, uv1 + pixel * float2(-1.0, -1.0)),
   image1.sample(image1Sampler, uv1 + pixel * float2(00.0, -1.0)),
   image1.sample(image1Sampler, uv1 + pixel * float2(+1.0, -1.0)),
   image1.sample(image1Sampler, uv1 + pixel * float2(-1.0, 00.0)),
   image1.sample(image1Sampler, uv1 + pixel * float2(00.0, 00.0)),
   image1.sample(image1Sampler, uv1 + pixel * float2(+1.0, 00.0)),
   image1.sample(image1Sampler, uv1 + pixel * float2(-1.0, +1.0)),
   image1.sample(image1Sampler, uv1 + pixel * float2(00.0, +1.0)),
   image1.sample(image1Sampler, uv1 + pixel * float2(+1.0, +1.0)),
};

half4 outline = 8.0h * dst_pixels[4] - (dst_pixels[0] + dst_pixels[1] +
dst_pixels[2] + dst_pixels[3] + dst_pixels[5] + dst_pixels[6] + dst_pixels[7] +
dst_pixels[8]);
```

Reduce number of operations



13 FPS

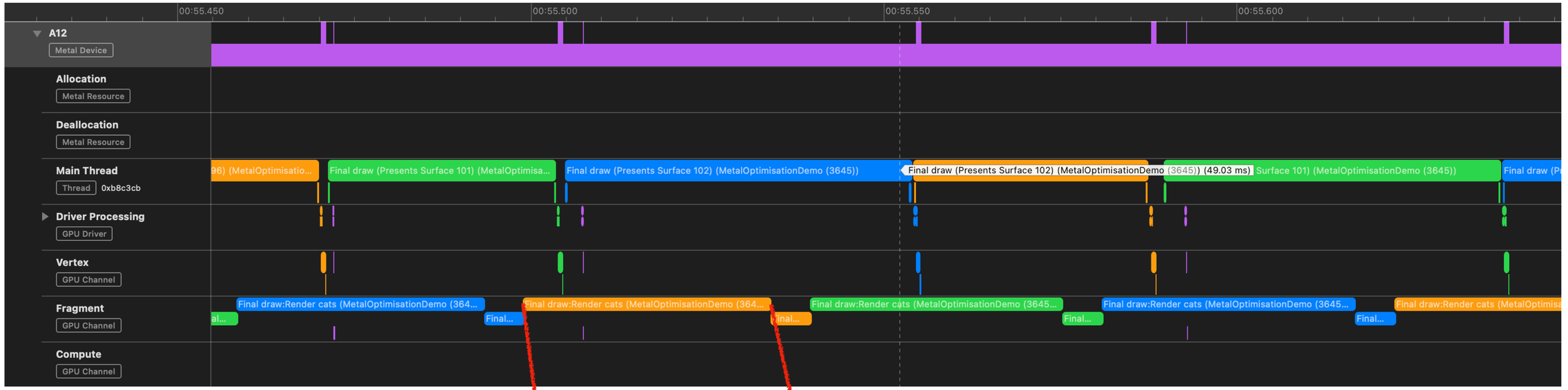
13 FPS

Use vectorisation and standard methods

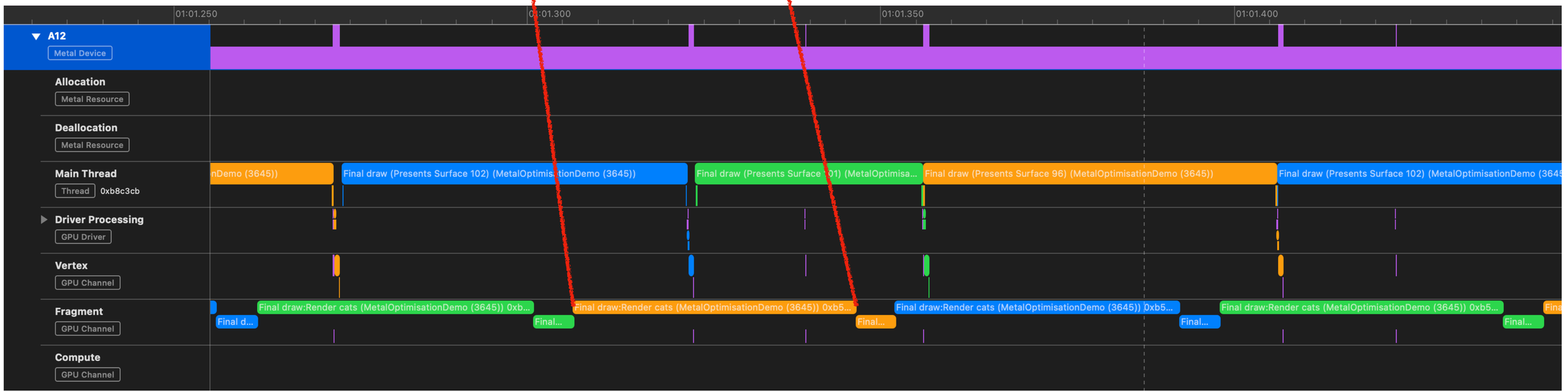
```
static half4 blend(half4 top, half4 btm) {
   half4 res;

   res.rgb = top.rgb + btm.rgb * (1.0h - top.a);
   res.a = top.a + btm.a * (1.0h - top.a);

   return res;
}
```

```
static half4 blend(half4 top, half4 btm) {
   return fma(1.0 - top.a, btm, top);
}
```

Use vectorisation and standard methods



13 FPS



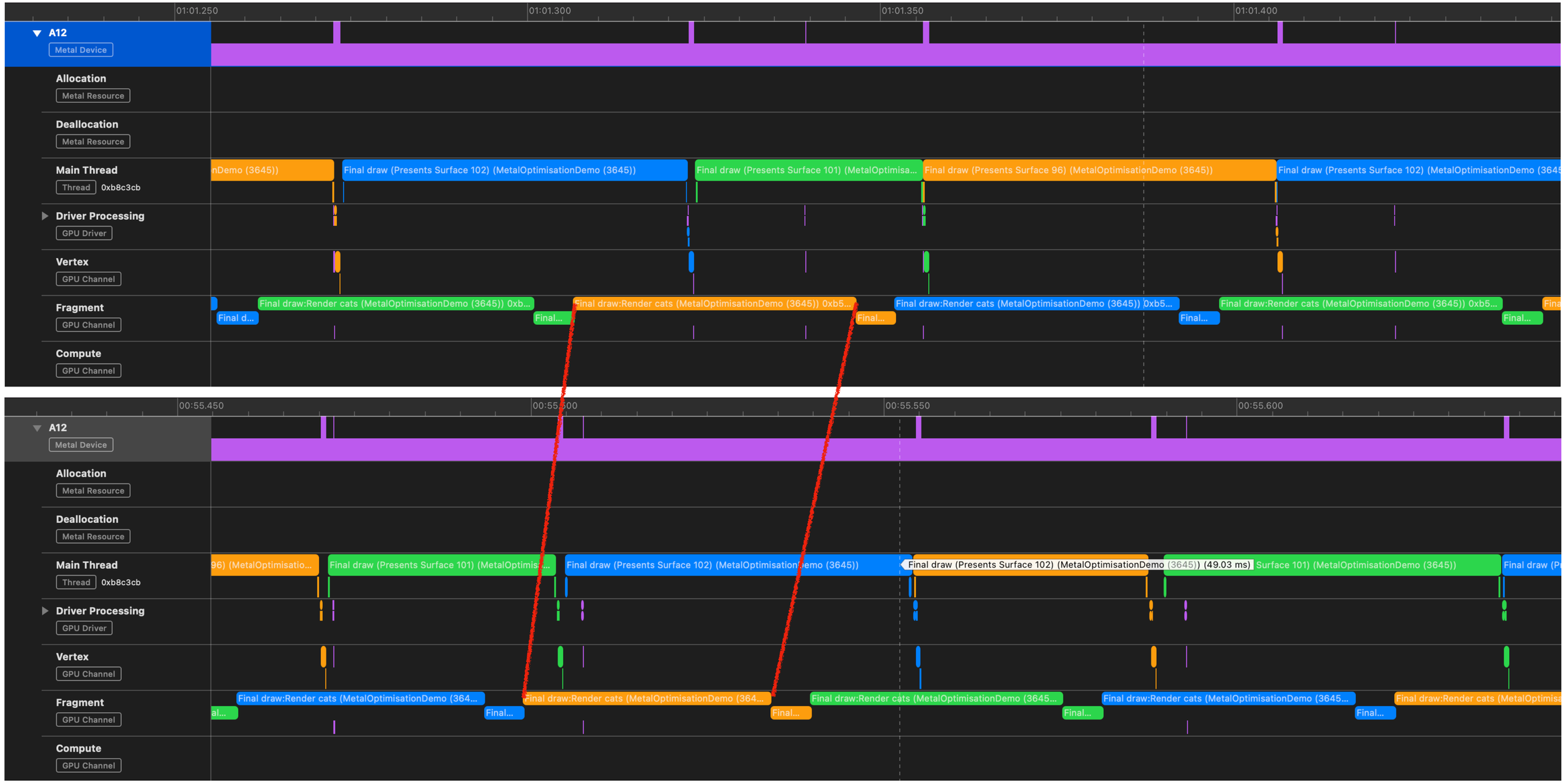
13 FPS

Use alternative algorithms

```
static half3 rgb2hsv(half3 c) {
    half M = max(c.x, max(c.y, c.z));
    half m = min(c.x, min(c.y, c.z));
    half C = M - m;

    half h = 0.0h;
    if (C > 0.0h) {
        if (c.r == M) {
            h = fmod((c.g - c.b) / C, 6.0h);
        } else if (c.g == M) {
            h = (c.b - c.r) / C + 2.0h;
        } else {
            h = (c.r - c.g) / C + 4.0h;
        }
    }
    h /= 6.0h;

    half v = M;

    half s = 0.0h;
    if (v > 0.0h) {
        s = C / v;
    }

    return clamp(half3(h, s, v), 0.0h, 1.0h);
}
```

```
static half3 rgb2hsv(half3 c)
{
    constexpr half4 K = half4(0.0h, -1.0h / 3.0h, 2.0h / 3.0h, -1.0h);
    half4 p = select(half4(c.bg, K.wz), half4(c.gb, K.xy), c.b < c.g);
    half4 q = select(half4(p.xyw, c.r), half4(c.r, p.yzx), p.x < c.r);

    half d = q.x - min(q.w, q.y);
    half e = 6.0e-8h;
    return clamp(half3(abs(q.z + (q.w - q.y) / (6.0h * d + e)),
                       d / (q.x + e), q.x),
                       0.0h,
                       1.0h);
}
```

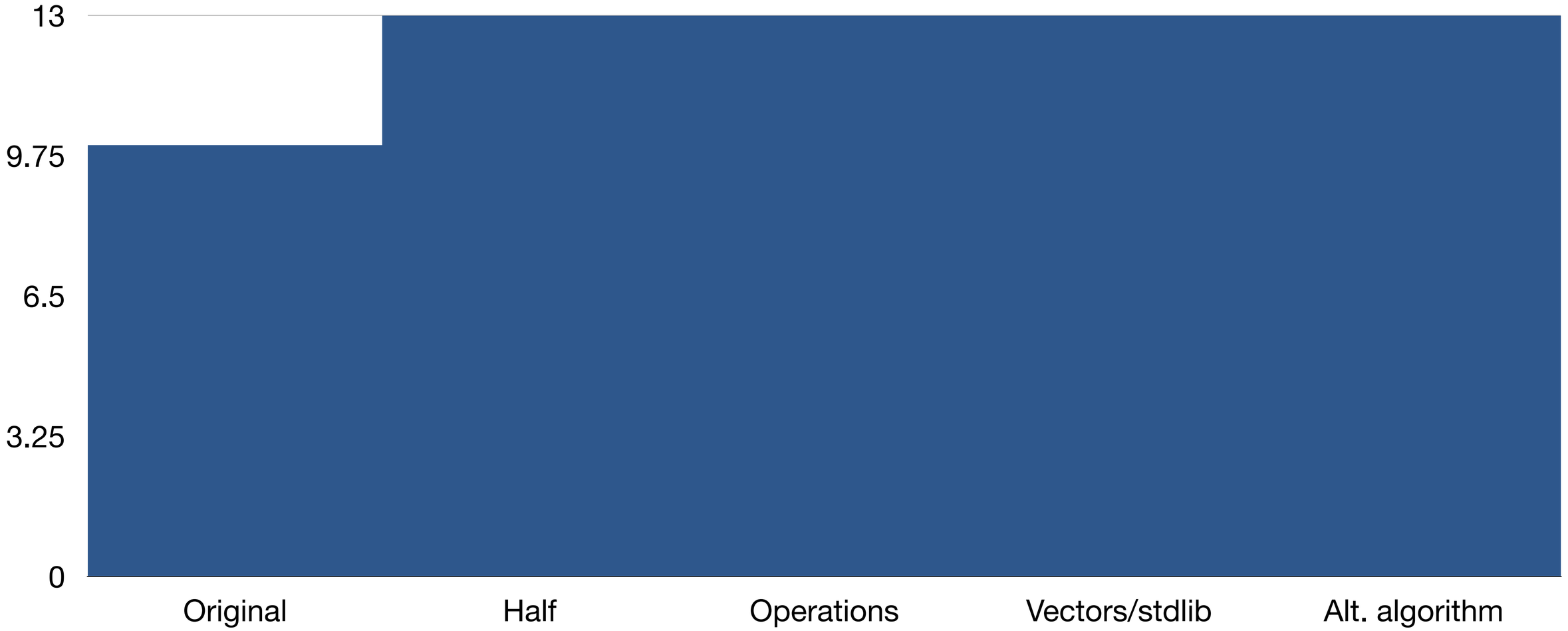
Use alternative algorithms



13 FPS

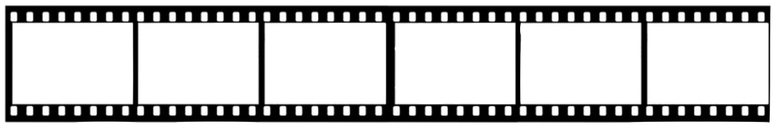
13 FPS

Fiasco? (FPS)


13
9.75
6.5
3.25
0
Original
Half
Operations
Vectors/stdlib
Alt. algorithm

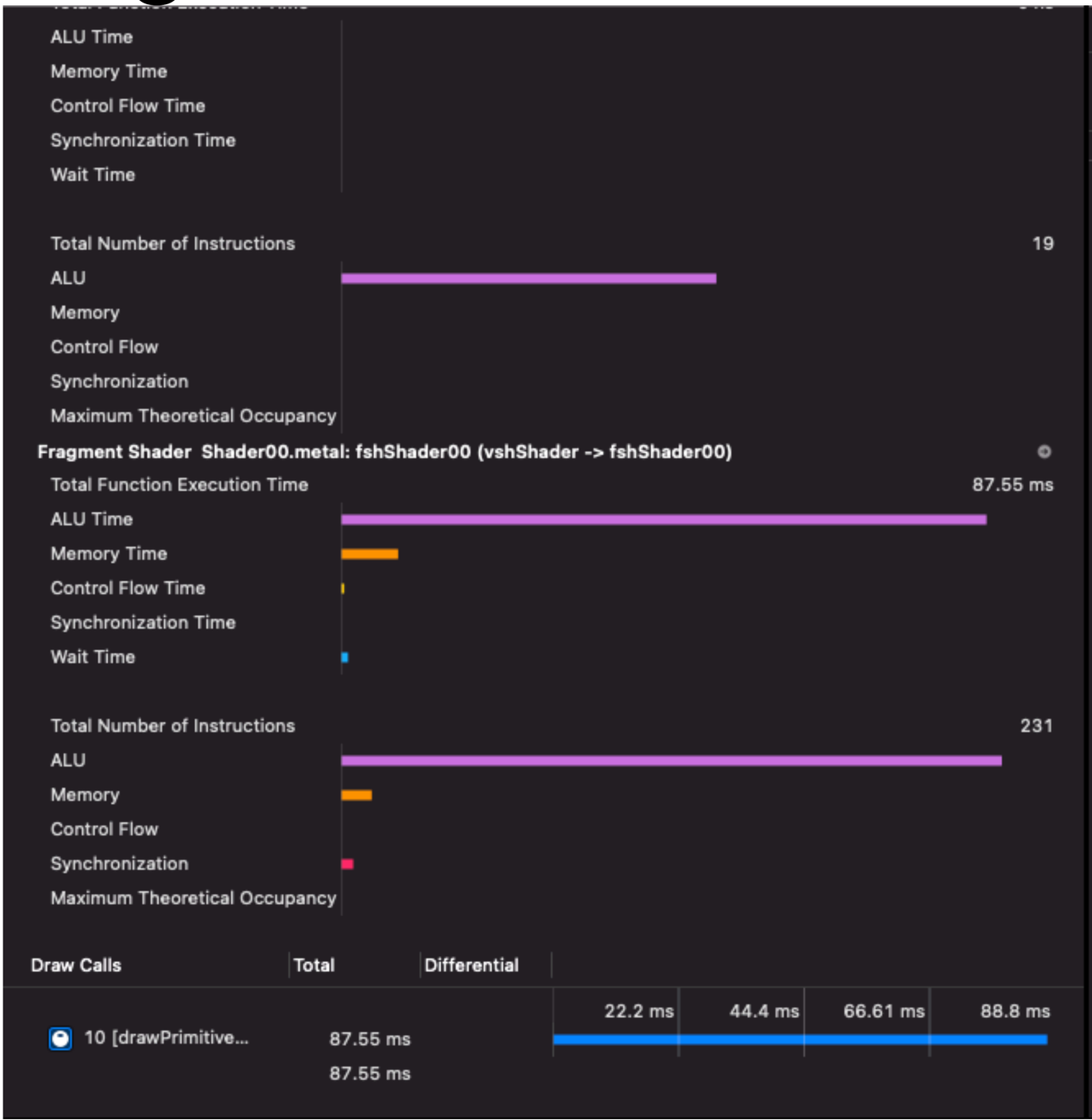
Use right profiling tool

Instruments: Metal System Trace



Xcode: Capture GPU frame

Original shader



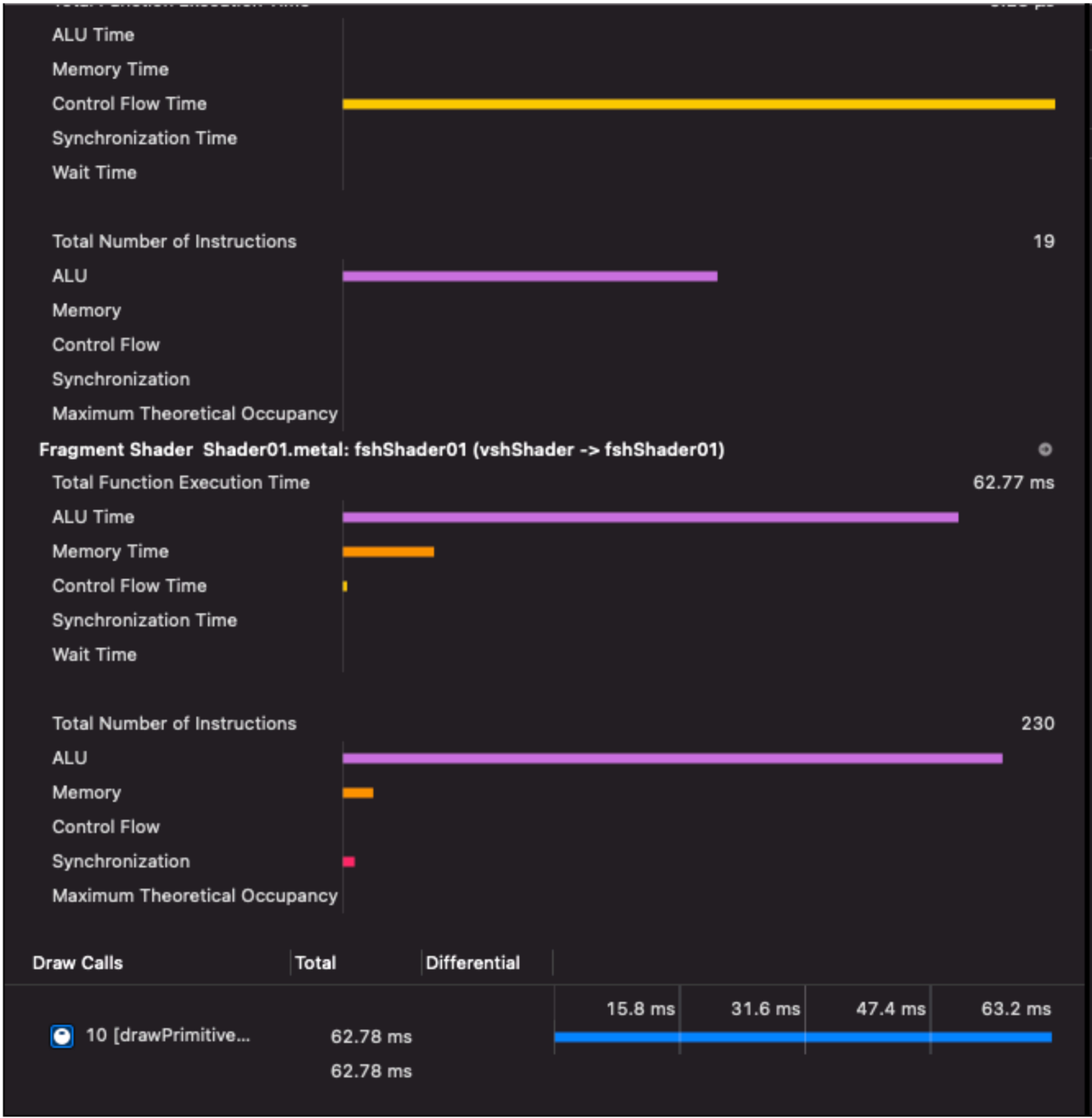
Potential Hotspots/Bottlenecks

No potential bottlenecks detected

Counters	Draw	Encoder
Fragment Shader Shader00.metal:fshShader00		
Fragment Stage Time	100.00%	100.00%
FS Invocations	67,108,860	67,108,860
Samples Shaded Per Tile	N/A	1,024.00
Sampler Calls/FS Invocation	10.00	10.00
FS Sampler Stall Time	N/A	1.63%
FS Texture Cache Miss Rate	N/A	27.93%
FS Occupancy	N/A	1.48%
FS Pixel Write Stall	N/A	0.00%
FS ALU Instructions	14,362,350,000	14,362,350,000
FS ALU Float Instructions	64.02%	64.02%
FS ALU Integer and Conditional Instructions	23.84%	23.84%
FS ALU Integer and Complex Instructions	12.15%	12.15%
FS ALU Inefficiency	0.01%	0.01%
FS ALU Limiter	N/A	91.40%
FS Texture Read Limiter	N/A	40.78%
FS Texture Write Limiter	N/A	7.83%
FS Bytes Read From Main Memory	N/A	40.56 MiB
FS Bytes Written To Main Memory	N/A	251.93 MiB
FS Main Memory Bandwidth	N/A	3.34
FS Last Level Cache Bytes Read	N/A	80.80 MiB
FS Last Level Cache Bytes Written	N/A	268.46 MiB
FS Texture L1 Bytes Read	57.30 MiB	56.86 MiB
FS Buffer L1 Miss Rate	N/A	44.45%
FS Buffer L1 Bytes Read	N/A	64.00 Bytes
FS Buffer L1 Bytes Written	N/A	384.00 Bytes
FS Arithmetic Intensity	N/A	49.11
FS ALU Performance	164.06	163.95
Pixels per Second	766,576,000.00	766,064,800.00

GROUP Fragment Shader

`float` to `half`



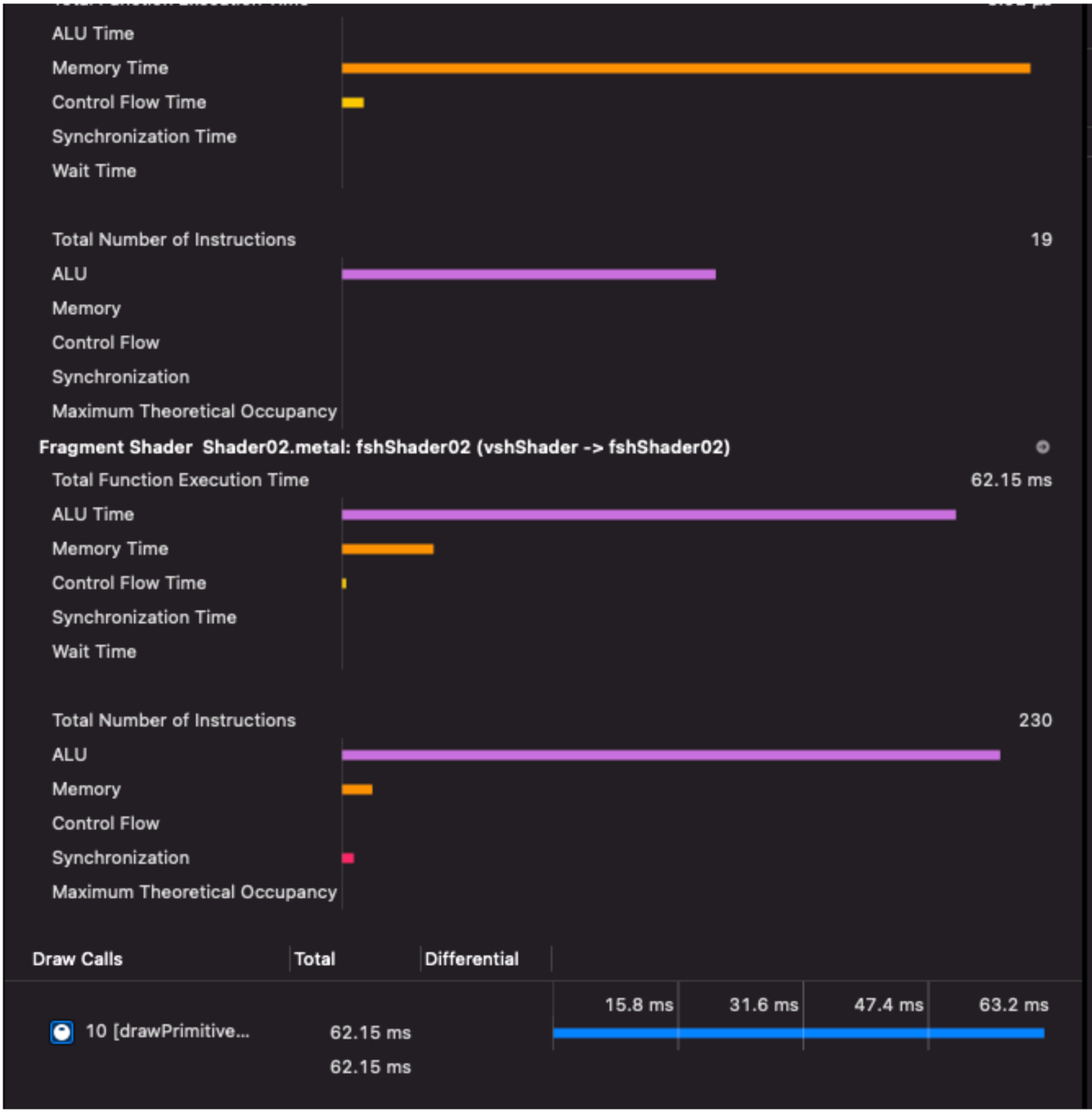
Potential Hotspots/Bottlenecks

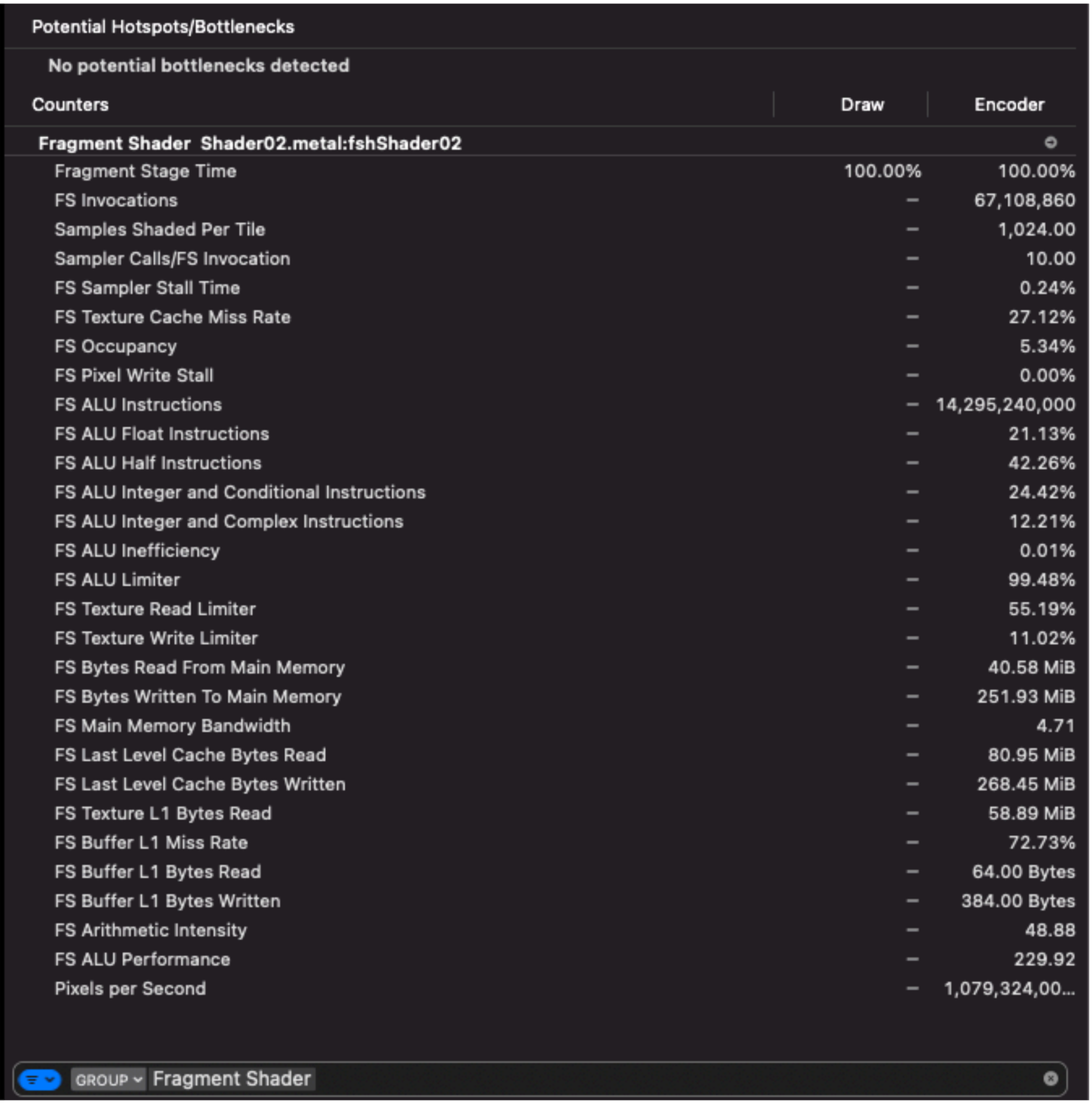
No potential bottlenecks detected

Counters	Draw	Encoder
Fragment Shader Shader01.metal:fshShader01		
Fragment Stage Time	100.00%	100.00%
FS Invocations	67,108,860	67,108,860
Samples Shaded Per Tile	N/A	1,024.00
Sampler Calls/FS Invocation	10.00	10.00
FS Sampler Stall Time	N/A	0.24%
FS Texture Cache Miss Rate	N/A	27.12%
FS Occupancy	N/A	5.79%
FS Pixel Write Stall	N/A	0.00%
FS ALU Instructions	14,295,240,000	14,295,240,000
FS ALU Float Instructions	23.48%	23.48%
FS ALU Half Instructions	39.91%	39.91%
FS ALU Integer and Conditional Instructions	24.42%	24.42%
FS ALU Integer and Complex Instructions	12.21%	12.21%
FS ALU Inefficiency	0.01%	0.01%
FS ALU Limiter	N/A	99.51%
FS Texture Read Limiter	N/A	54.63%
FS Texture Write Limiter	N/A	10.91%
FS Bytes Read From Main Memory	N/A	40.58 MiB
FS Bytes Written To Main Memory	N/A	251.93 MiB
FS Main Memory Bandwidth	N/A	4.66
FS Last Level Cache Bytes Read	N/A	81.32 MiB
FS Last Level Cache Bytes Written	N/A	268.46 MiB
FS Texture L1 Bytes Read	59.76 MiB	58.89 MiB
FS Buffer L1 Miss Rate	N/A	44.45%
FS Buffer L1 Bytes Read	N/A	64.00 Bytes
FS Buffer L1 Bytes Written	N/A	384.00 Bytes
FS Arithmetic Intensity	N/A	48.88
FS ALU Performance	227.75	227.62
Pixels per Second	1,069,147,00...	1,068,526,00...

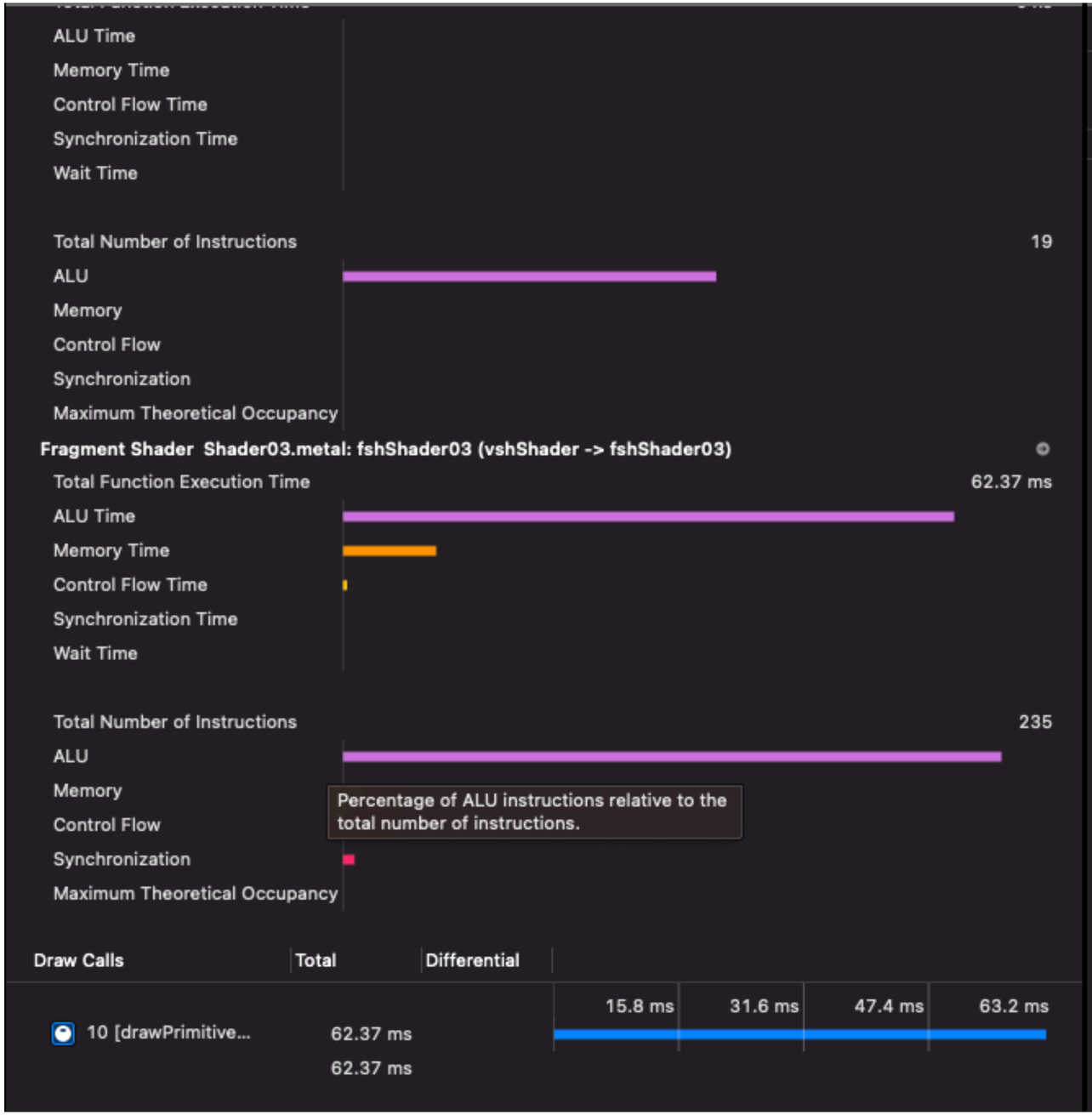
GROUP Fragment Shader

Reduce number of operations





Vectorisation + stdlib



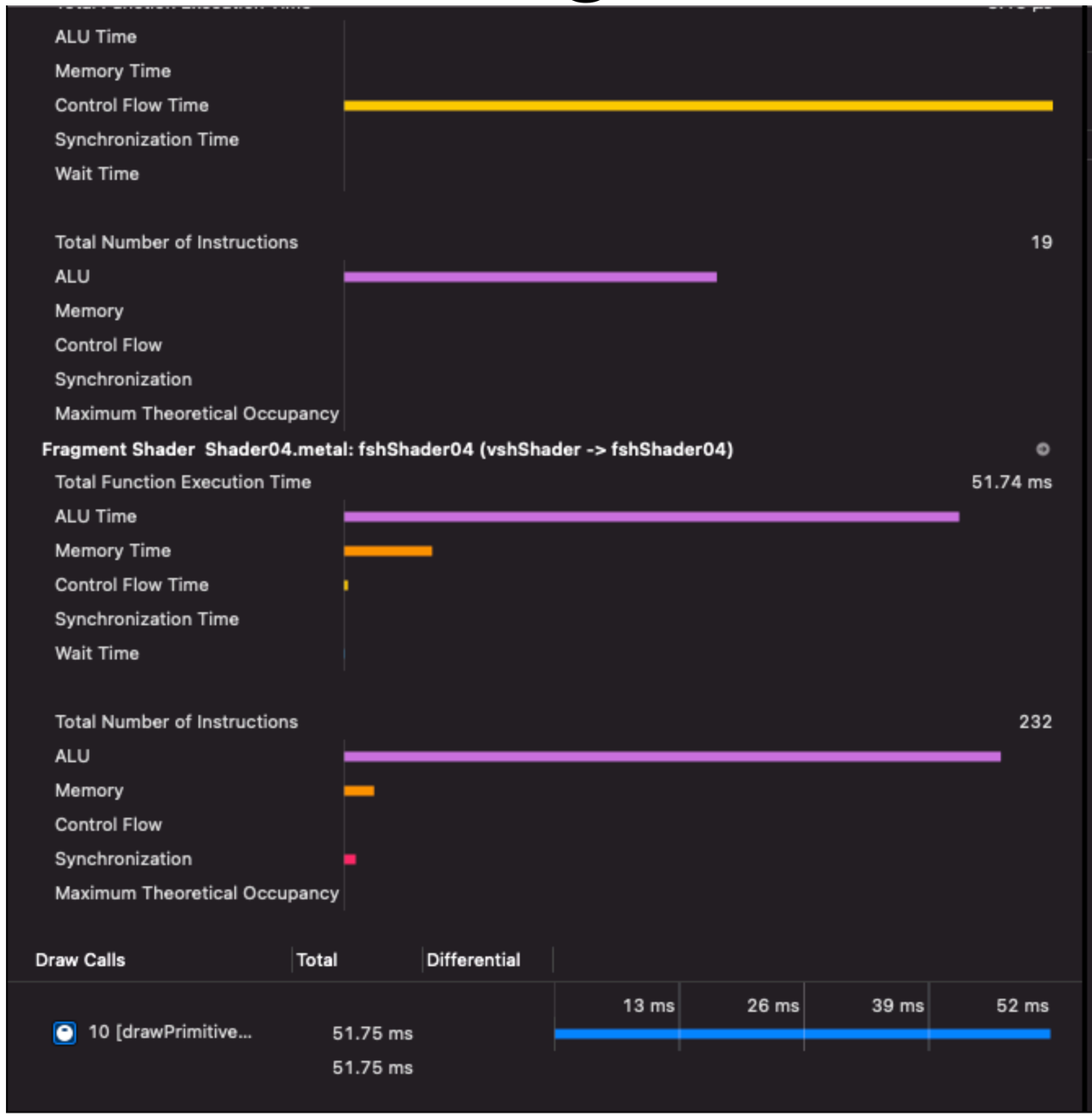
Potential Hotspots/Bottlenecks

No potential bottlenecks detected

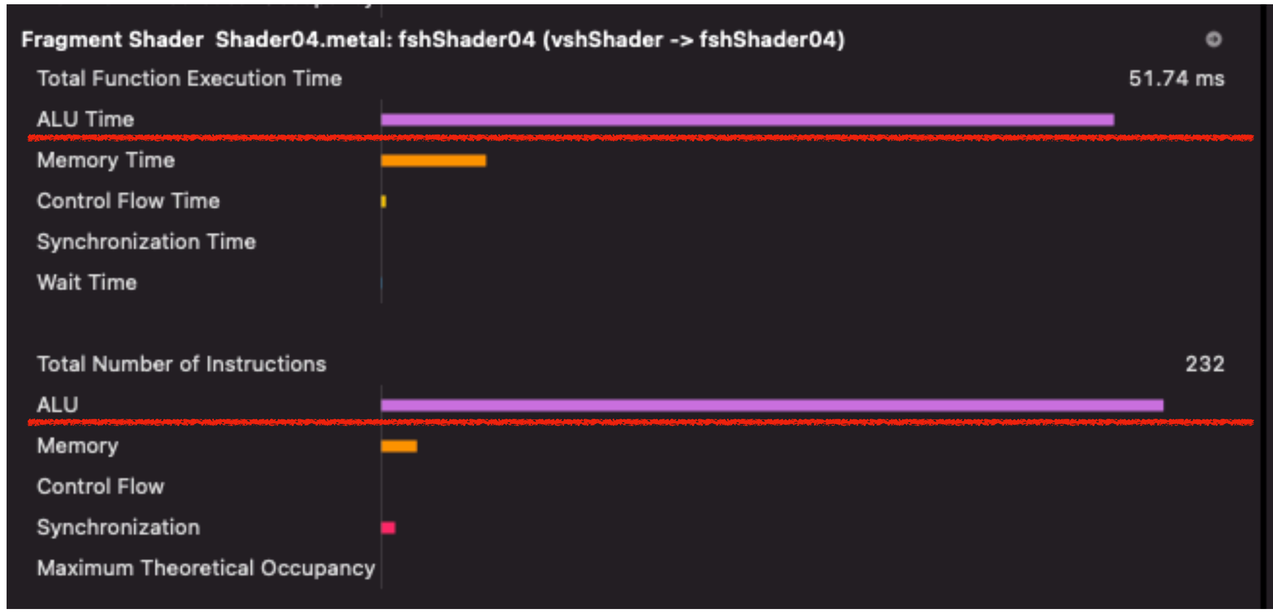
Counters	Draw	Encoder
Fragment Shader Shader03.metal:fshShader03		
Fragment Stage Time	100.00%	100.00%
FS Invocations	–	67,108,860
Samples Shaded Per Tile	–	1,024.00
Sampler Calls/FS Invocation	–	10.00
FS Sampler Stall Time	–	0.23%
FS Texture Cache Miss Rate	–	26.87%
FS Occupancy	–	5.50%
FS Pixel Write Stall	–	0.00%
FS ALU Instructions	–	14,630,780,000
FS ALU Float Instructions	–	21.56%
FS ALU Half Instructions	–	42.66%
FS ALU Integer and Conditional Instructions	–	23.86%
FS ALU Integer and Complex Instructions	–	11.93%
FS ALU Inefficiency	–	0.01%
FS ALU Limiter	–	99.57%
FS Texture Read Limiter	–	54.99%
FS Texture Write Limiter	–	10.99%
FS Bytes Read From Main Memory	–	40.57 MiB
FS Bytes Written To Main Memory	–	251.93 MiB
FS Main Memory Bandwidth	–	4.69
FS Last Level Cache Bytes Read	–	81.37 MiB
FS Last Level Cache Bytes Written	–	268.45 MiB
FS Texture L1 Bytes Read	–	59.37 MiB
FS Buffer L1 Miss Rate	–	64.29%
FS Buffer L1 Bytes Read	–	64.00 Bytes
FS Buffer L1 Bytes Written	–	384.00 Bytes
FS Arithmetic Intensity	–	50.03
FS ALU Performance	–	234.46
Pixels per Second	–	1,075,394,00...

GROUP Fragment Shader

Alternative algorithm



Recognize bottleneck



FS ALU Float Instructions	–	19.54%
FS ALU Half Instructions	–	46.98%
FS ALU Integer and Conditional Instructions	–	21.86%
FS ALU Integer and Complex Instructions	–	11.63%
FS ALU Inefficiency	–	0.01%
FS ALU Limiter	–	98.34%
FS Texture Read Limiter	–	54.83%
FS Texture Write Limiter	–	10.95%
FS Bytes Read From Main Memory	–	40.57 MiB
FS Bytes Written To Main Memory	–	251.93 MiB
FS Main Memory Bandwidth	–	5.65
FS Last Level Cache Bytes Read	–	83.02 MiB
FS Last Level Cache Bytes Written	–	268.46 MiB
FS Texture L1 Bytes Read	–	63.07 MiB
FS Buffer L1 Miss Rate	–	0.00%
FS Buffer L1 Bytes Read	–	64.00 Bytes
FS Buffer L1 Bytes Written	–	384.00 Bytes

Recognize bottleneck

```
half4 dst = dst_pixels[4] * smoothstep(0.8h, 1.0h, 1.0h -          9.91%
    length(outline));

// - HSV
half3 hsv = rgb2hsv(dst.rgb);                                      3.86%
half3 col = time_color(half2(uv1), time);                          16.95%
hsv.x = fract(fma(half(angle), 0.5h / M_PI_H, 0.5h) -              10.77%
    fma(time, 0.25h, -rgb2hsv(col).x));

half k = smoothstep(0.0h, 1.0h, length(half2(shift)));             2.89%
hsv.y = fma(hsv.y, 1.0h - k, k);                                   1.21%
dst.rgb = hsv2rgb(hsv);                                            18.04%
```

Reducing weights

Use your own functions

```
template <typename T>
inline static T pos_fract(T x) {
   return x - floor(x);
}
```

Reducing weights

Reduce access to vector items

```
static half3 hsv2rgb(half3 c) {
   constexpr half4 K = half4(1.0, 2.0 / 3.0, 1.0 / 3.0, 3.0);
   half3 p = abs(fract(c.xxx + K.xyz) * 6.0h - K.www);
   return clamp(c.z * mix(K.xxx, clamp(p - K.xxx, 0.0h, 1.0h), c.y), 0.0h, 1.0h);
}
```

```
static half3 hsv2rgb(half3 c) {
   constexpr half3 K = half3(1.0h, 2.0h / 3.0h, 1.0h / 3.0h);
   half3 p = abs(pos_fract(c.x + K) * 6.0h - 3.0h);
   return c.z * fma(clamp(p - 1.0h, 0.0h, 1.0h), c.y, 1.0h - c.y);
}
```

Reducing weights

Use school math :)

```
static half3 time_color(half2 uv, half time) {
   return fma(abs(half3(
     sin(fma(cos(fma(3.0h, uv.y, time)), 2.0h * uv.x, time)),
     cos(fma(sin(fma(2.0h, uv.x, time)), 3.0h * uv.y, time)),
     0.4h / 0.9h)), 0.9h, 0.1h);
}
```

$$\sin\left(\frac{\pi}{2} + \theta\right) = \cos\theta$$

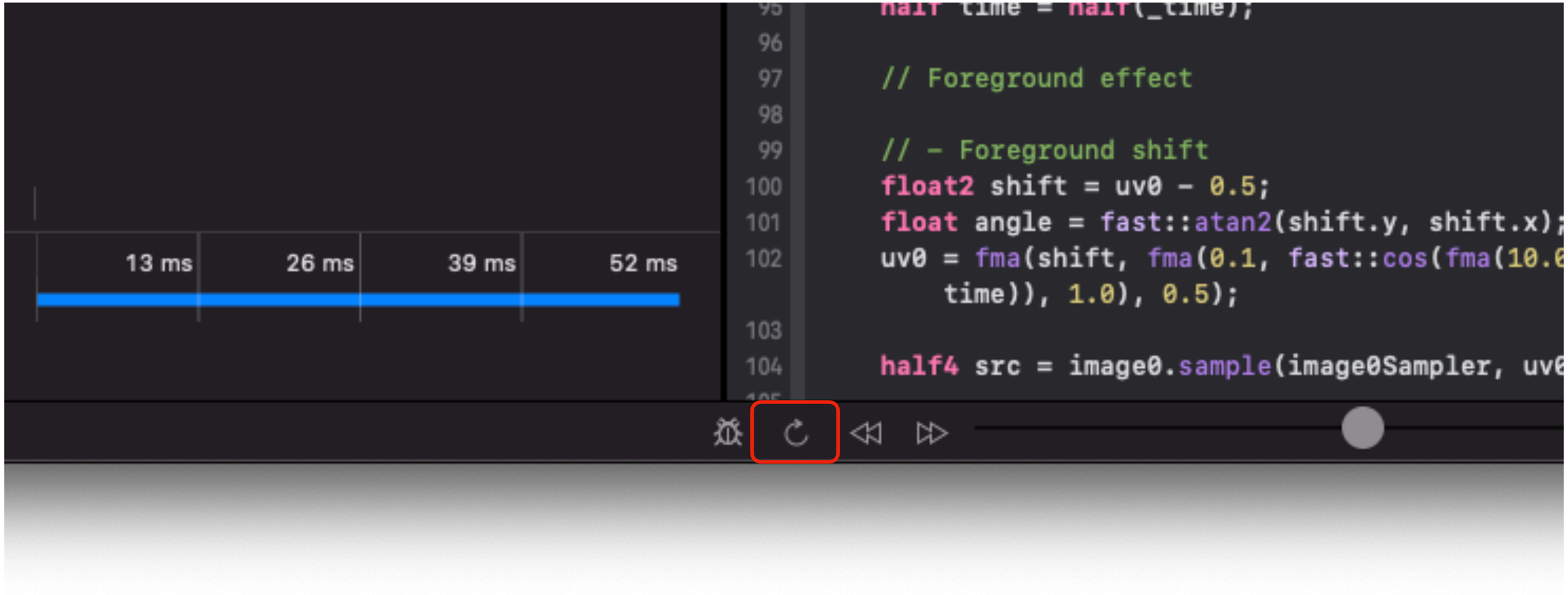
Reducing weights

Use school math :)

```
static half3 time_color(half2 uv, half time) {
   return fma(abs(half3(
     sin(fma(cos(fma(3.0h, uv.y, time)), 2.0h * uv.x, time)),
     cos(fma(sin(fma(2.0h, uv.x, time)), 3.0h * uv.y, time)),
     0.4h / 0.9h)), 0.9h, 0.1h);
}
```

```
static half3 time_color(half2 uv, half time) {
   half2 uvl = uv * half2(3.0h, 2.0h);
   half2 tcs = time + half2(M_PI_2_H, 0.0h);
   half2 tsc = time + half2(0.0h, M_PI_2_H);

   half2 a = uvl + tcs;
   half2 a_cs = sin(a);
   half2 b = fma(uvl, a_cs, tsc);
   half2 b_sc = sin(b);

   return fma(abs(half3(b_sc, 0.4h / 0.9h)), 0.9h, 0.1h);
}
```

Update shader on-fly



Reducing weights

Summary

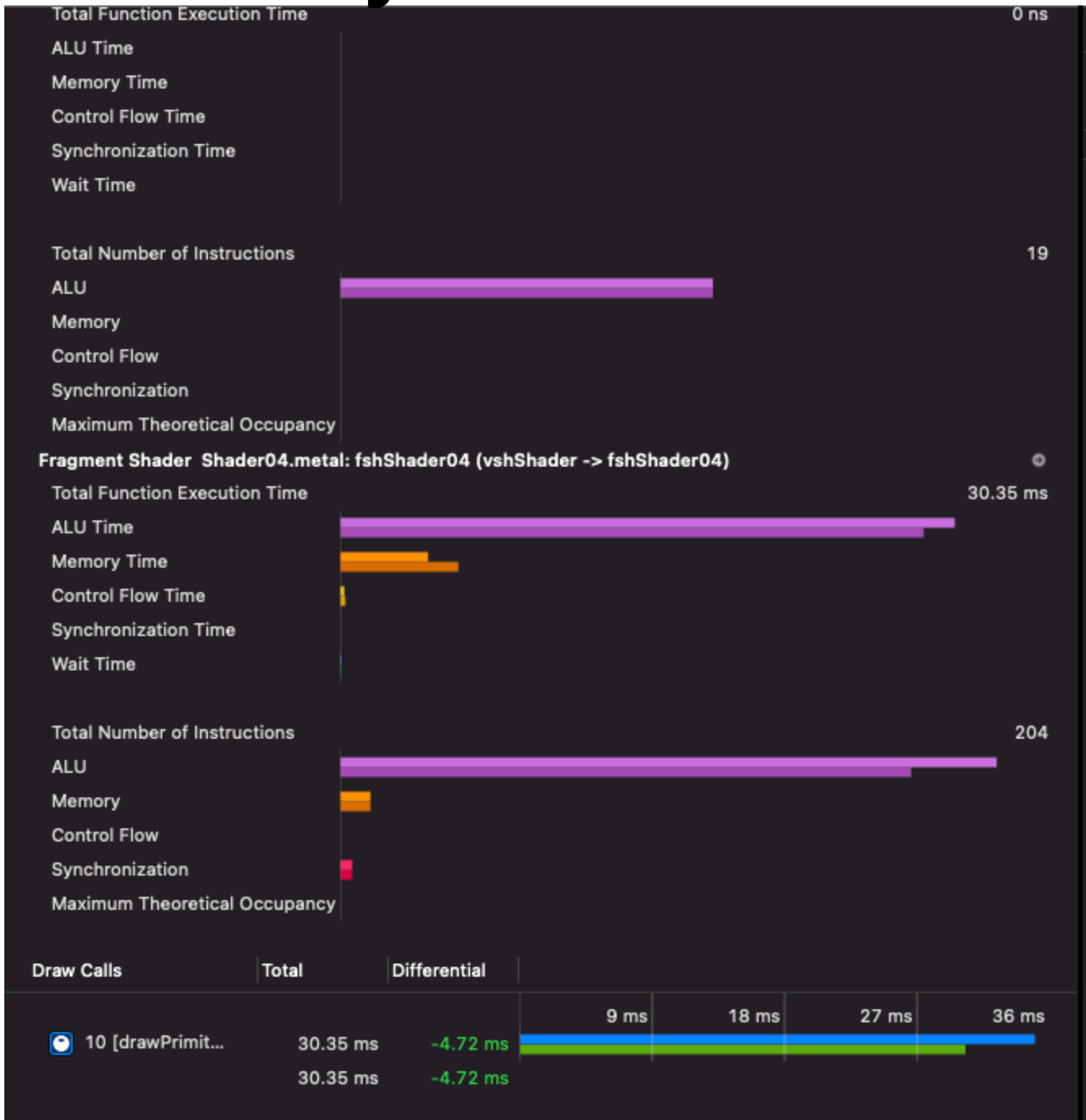
```
half4 dst = dst_pixels[4] * smoothstep(0.8h, 1.0h, 1.0h -
    length(outline));

// - HSV
half3 hsv = rgb2hsv(dst.rgb);
half3 col = time_color(half2(uv1), time);
hsv.x = fract(fma(half(angle), 0.5h / M_PI_H, 0.5h) -
    fma(time, 0.25h, -rgb2hsv(col).x));

half k = smoothstep(0.0h, 1.0h, length(half2(shift)));
hsv.y = fma(hsv.y, 1.0h - k, k);
dst.rgb = hsv2rgb(hsv);
```

```
9.91%
3.86%
16.95%
10.77%
2.89%
1.21%
18.04%
```

```
half4 dst = dst_pixels[4] * smoothstep(0.8h, 1.0h, 1.0h -
    border);

// - HSV
half3 hsv = rgb2hsv(dst.rgb);
half3 col = time_color(half2(uv1), time);
hsv.x = pos_fract(fma(half(angle), 0.5h / M_PI_H, 0.5h) -
    fma(time, 0.25h, -rgb2hsv(col).x));

half k = smoothstep(0.0h, 1.0h, length(half2(shift)));
hsv.y = fma(hsv.y, 1.0h - k, k);
dst.rgb = hsv2rgb(max(hsv, 0.0h));
```

```
4.28%
4.90%
13.96%
9.31%
4.08%
1.21%
16.10%
```

Summary



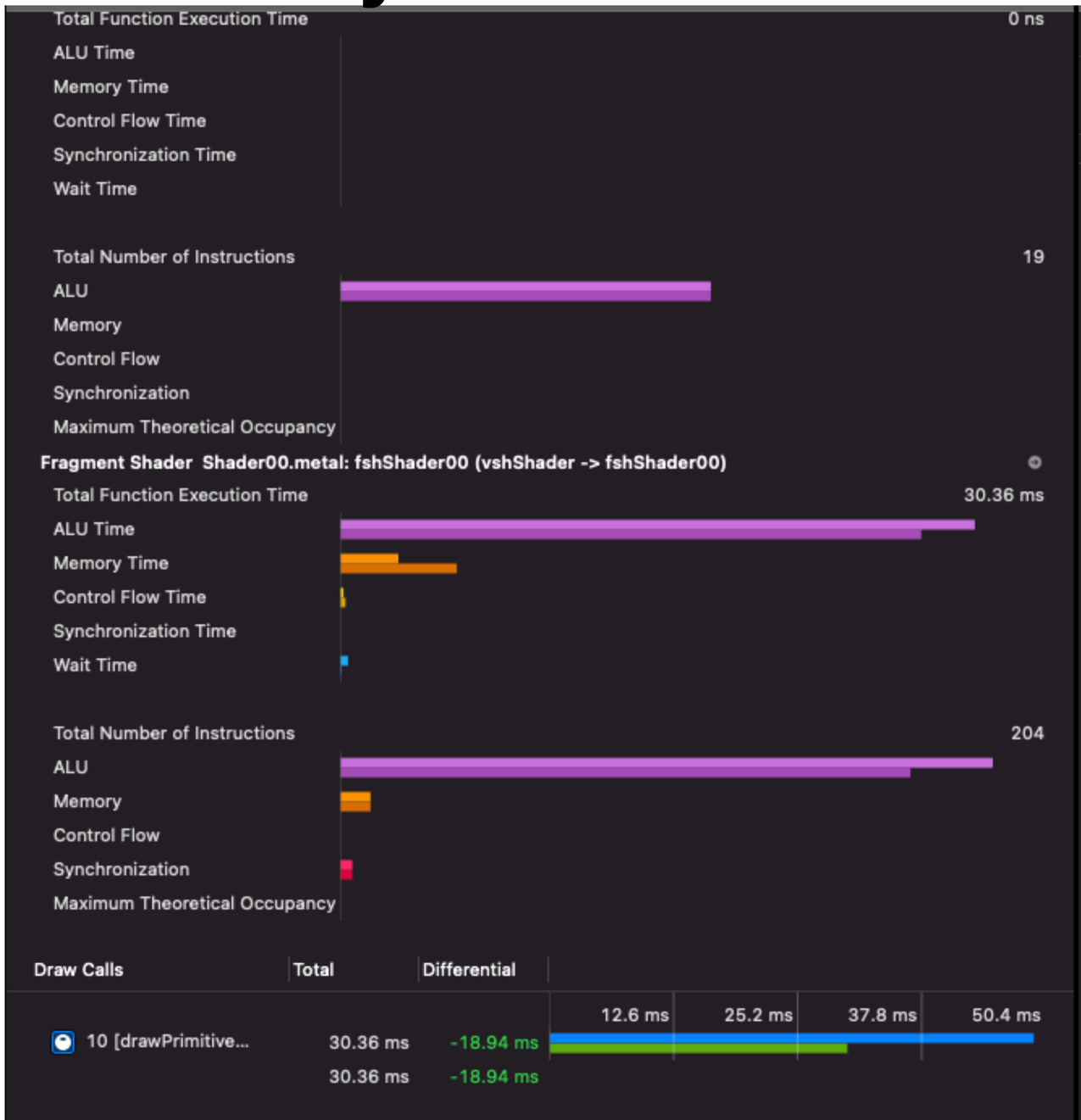
Potential Hotspots/Bottlenecks

No potential bottlenecks detected

Counters	Draw	Encoder
Fragment Shader Shader04.metal:fshShader04		
Fragment Stage Time	100.00%	100.00%
FS Invocations	67,108,860	67,108,860
Samples Shaded Per Tile	N/A	1,024.00
Sampler Calls/FS Invocation	10.00	10.00
FS Sampler Stall Time	N/A	0.98%
FS Texture Cache Miss Rate	N/A	24.27%
FS Occupancy	N/A	15.75%
FS Pixel Write Stall	N/A	0.00%
FS ALU Instructions	12,550,410,000	12,550,410,000
FS ALU Float Instructions	20.32%	20.32%
FS ALU Half Instructions	48.66%	48.66%
FS ALU Integer and Conditional Instructions	18.19%	18.19%
FS ALU Integer and Complex Instructions	12.84%	12.84%
FS ALU Inefficiency	0.01%	0.01%
FS ALU Limiter	N/A	98.85%
FS Texture Read Limiter	N/A	63.60%
FS Texture Write Limiter	N/A	12.58%
FS Bytes Read From Main Memory	N/A	40.59 MiB
FS Bytes Written To Main Memory	N/A	251.93 MiB
FS Main Memory Bandwidth	N/A	9.64
FS Last Level Cache Bytes Read	N/A	81.98 MiB
FS Last Level Cache Bytes Written	N/A	268.46 MiB
FS Texture L1 Bytes Read	66.59 MiB	66.90 MiB
FS Buffer L1 Miss Rate	N/A	56.25%
FS Buffer L1 Bytes Read	N/A	64.00 Bytes
FS Buffer L1 Bytes Written	N/A	384.00 Bytes
FS Arithmetic Intensity	N/A	42.91
FS ALU Performance	413.53	413.29
Pixels per Second	2,211,189,00...	2,209,874,00...

GROUP Fragment Shader

Summary



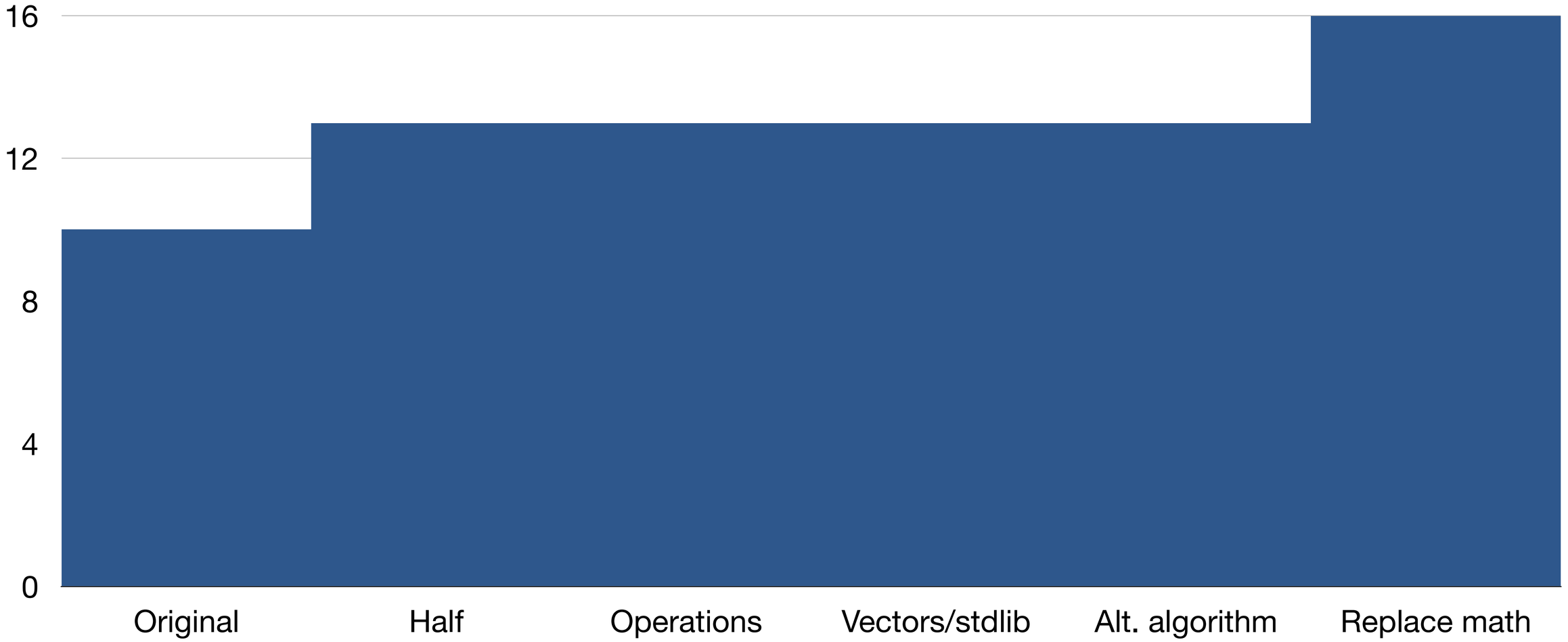
Potential Hotspots/Bottlenecks

No potential bottlenecks detected

Counters	Draw	Encoder
Fragment Shader Shader00.metal:fshShader00		
Fragment Stage Time	100.00%	100.00%
FS Invocations	–	67,108,860
Samples Shaded Per Tile	–	1,024.00
Sampler Calls/FS Invocation	–	10.00
FS Sampler Stall Time	–	0.89%
FS Texture Cache Miss Rate	–	24.38%
FS Occupancy	–	15.78%
FS Pixel Write Stall	–	0.00%
FS ALU Instructions	–	12,550,410,000
FS ALU Float Instructions	–	20.32%
FS ALU Half Instructions	–	48.66%
FS ALU Integer and Conditional Instructions	–	18.19%
FS ALU Integer and Complex Instructions	–	12.84%
FS ALU Inefficiency	–	0.01%
FS ALU Limiter	–	98.86%
FS Texture Read Limiter	–	63.60%
FS Texture Write Limiter	–	12.58%
FS Bytes Read From Main Memory	–	40.59 MiB
FS Bytes Written To Main Memory	–	251.93 MiB
FS Main Memory Bandwidth	–	9.64
FS Last Level Cache Bytes Read	–	81.90 MiB
FS Last Level Cache Bytes Written	–	268.46 MiB
FS Texture L1 Bytes Read	–	66.59 MiB
FS Buffer L1 Miss Rate	–	44.45%
FS Buffer L1 Bytes Read	–	64.00 Bytes
FS Buffer L1 Bytes Written	–	384.00 Bytes
FS Arithmetic Intensity	–	42.91
FS ALU Performance	–	413.28
Pixels per Second	–	2,209,862,00...

GROUP Fragment Shader

Summary (FPS)


16
12
8
4
0
Original
Half
Operations
Vectors/stdlib
Alt. algorithm
Replace math

Classes in shaders

- Encapsulate context
- Cache computation results
- Reduce passing objects
- More understandable code

Textures

Another example

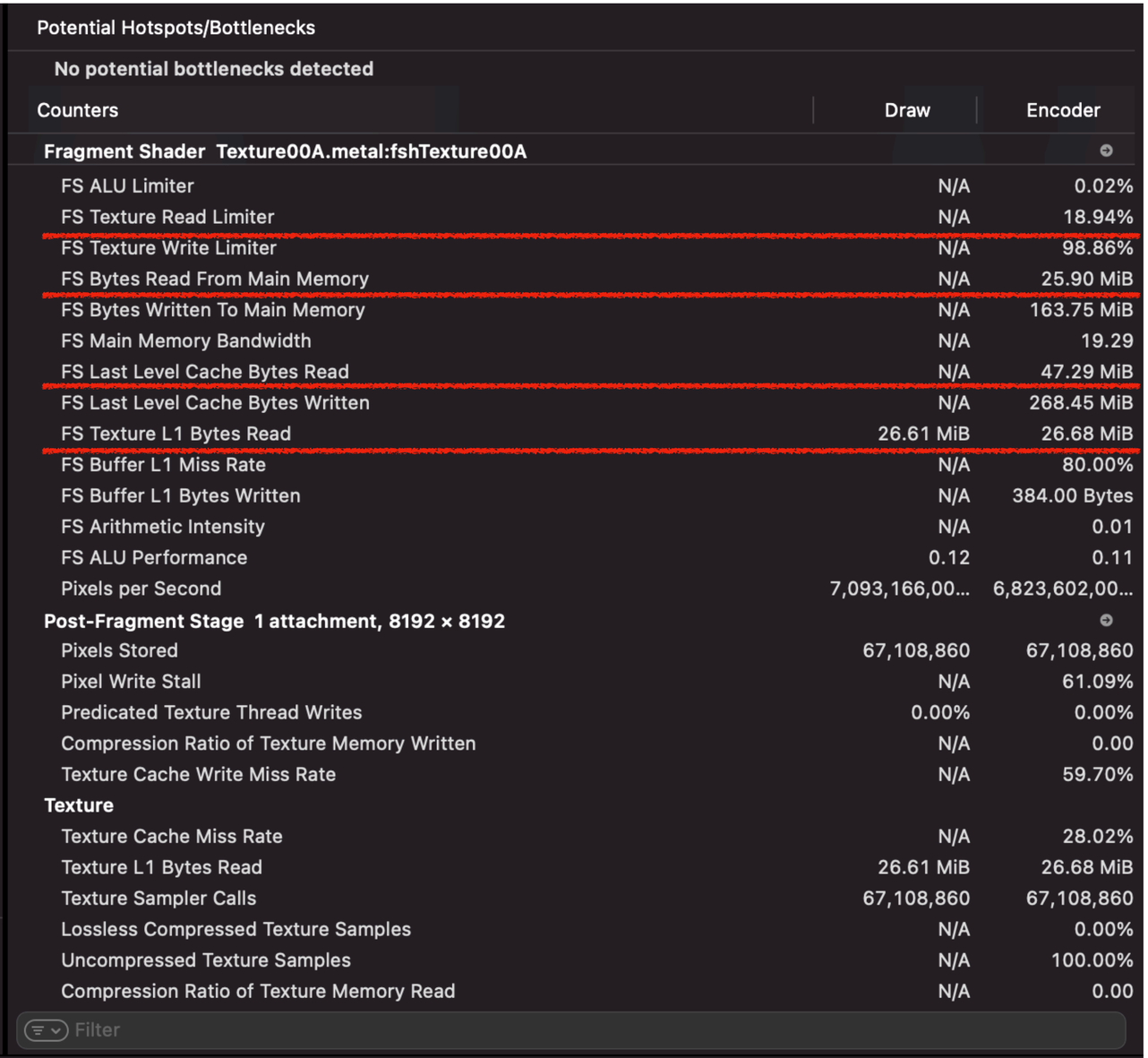
displacing

- one input textures
- displacement transforms
  - random tile
  - `sin` offset
- target texture size: 8192 x 8192 px

Localisation (0)

~ 10.46 ms

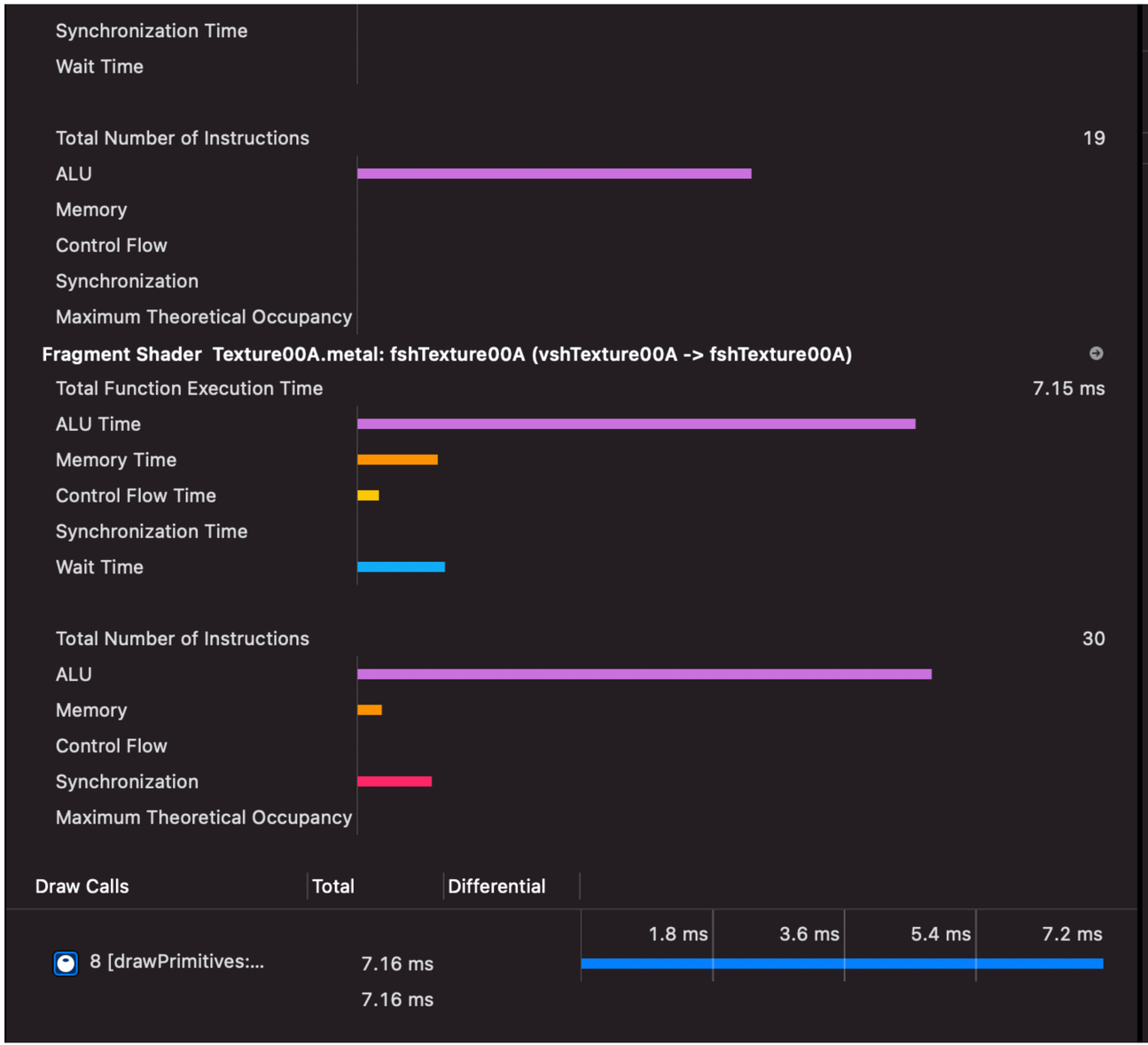
Localisation (0)

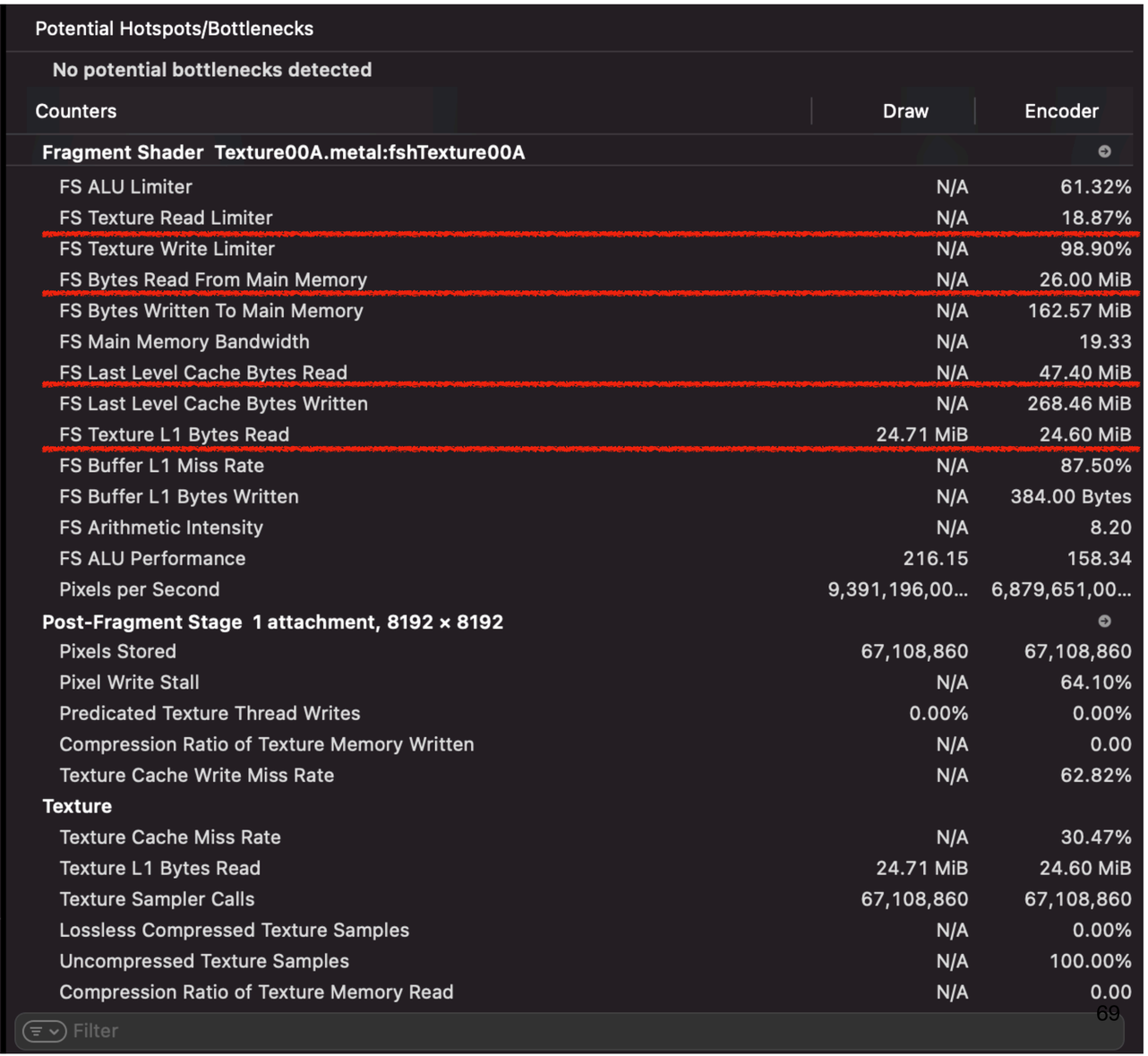


Localisation (1)

~ 10.24 ms

Localisation (1)



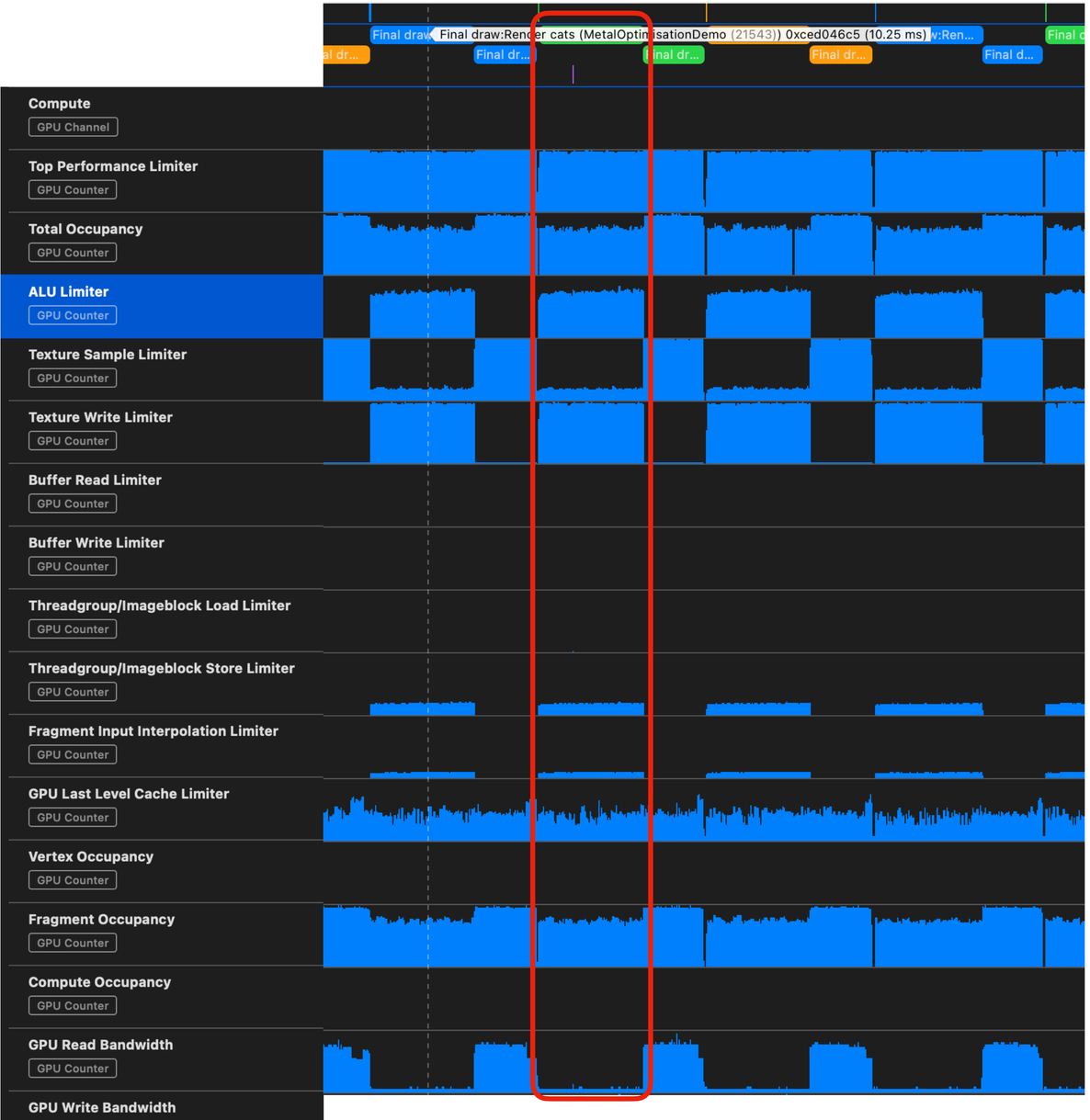


Localisation (2)

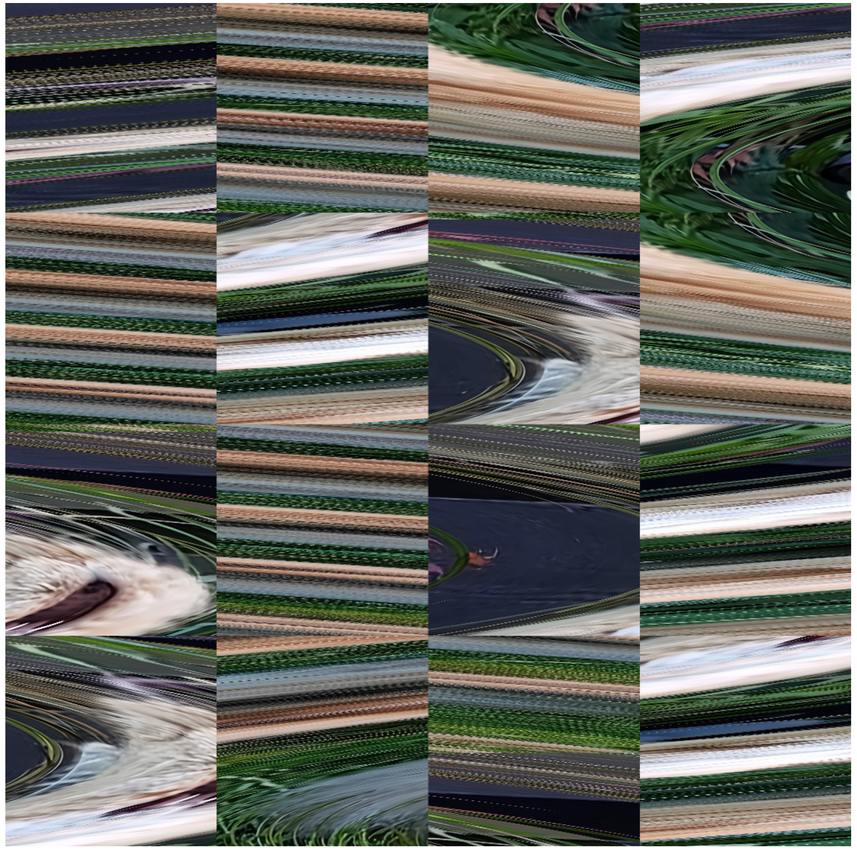
~ 10.01 ms

Localisation (2)

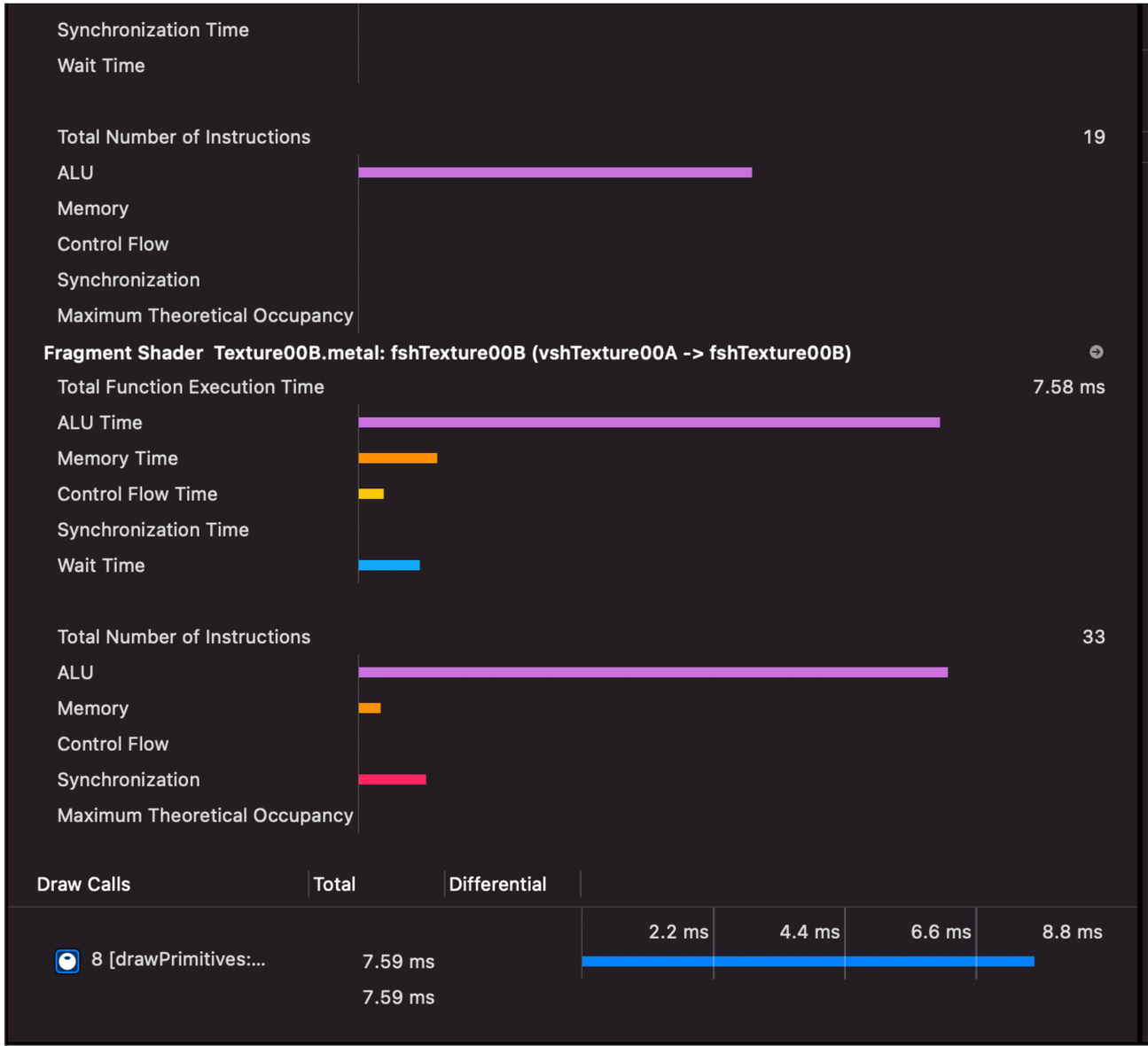
Localisation (4)

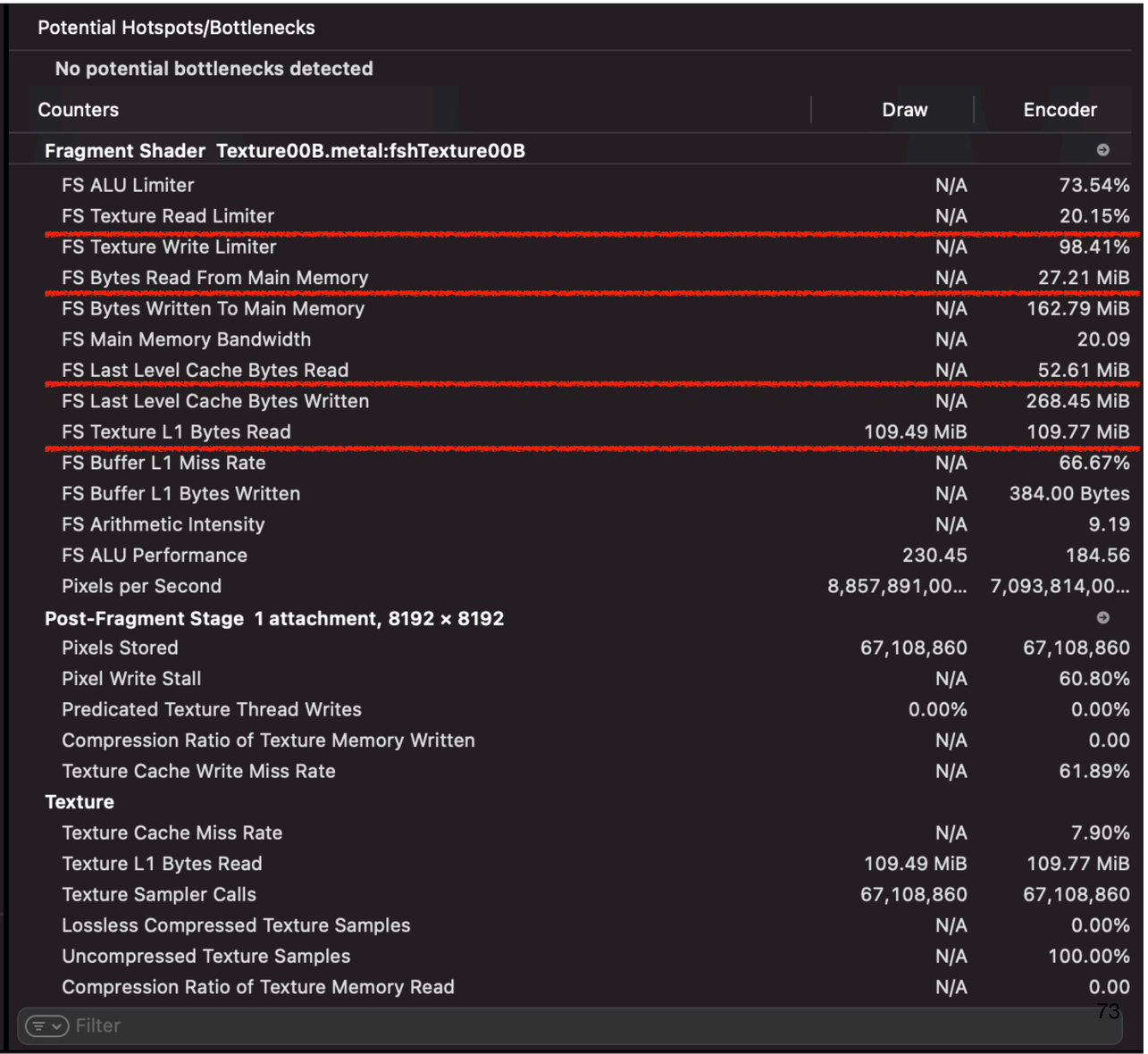


~ 10.25 ms



Localisation (4)

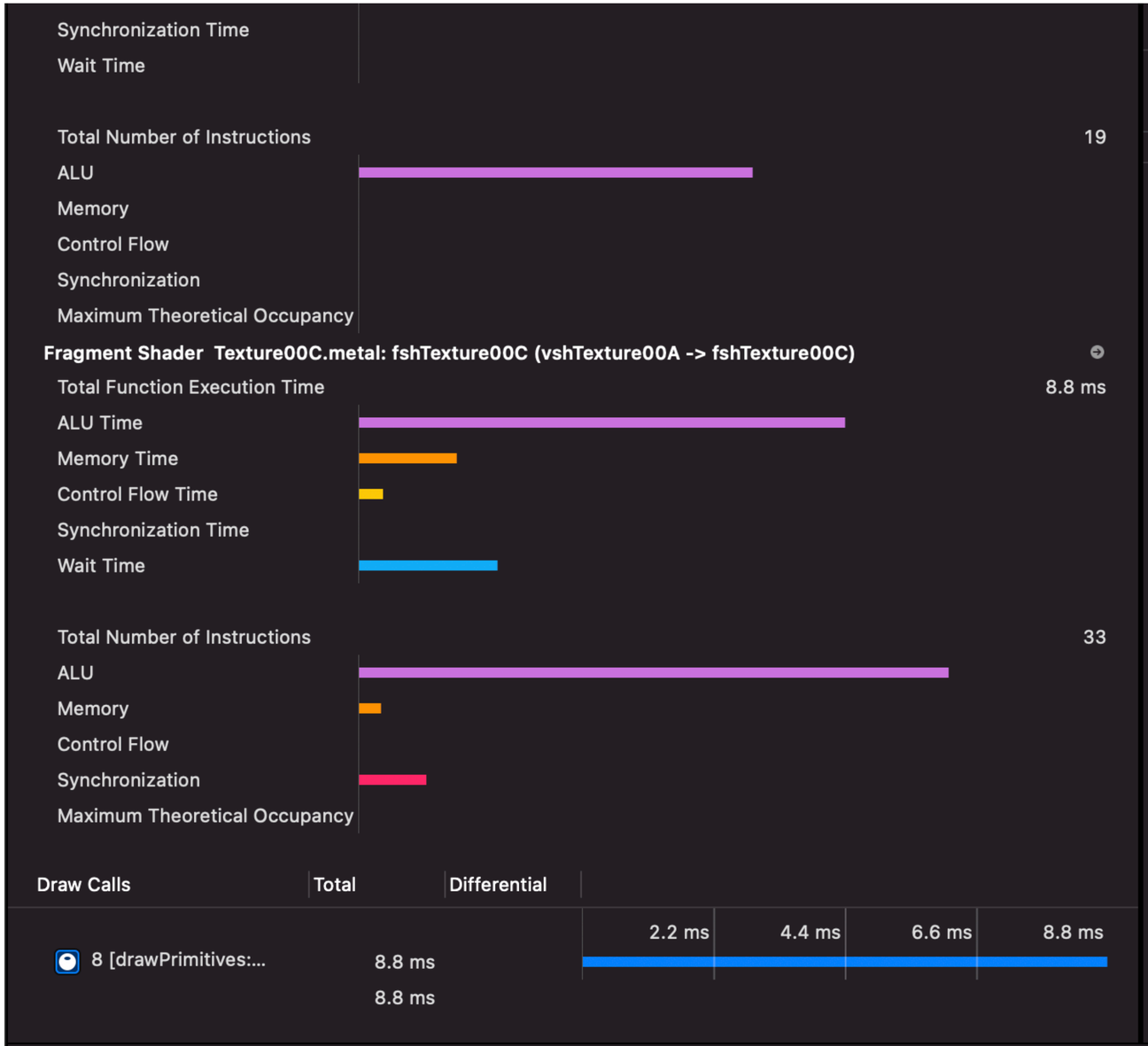


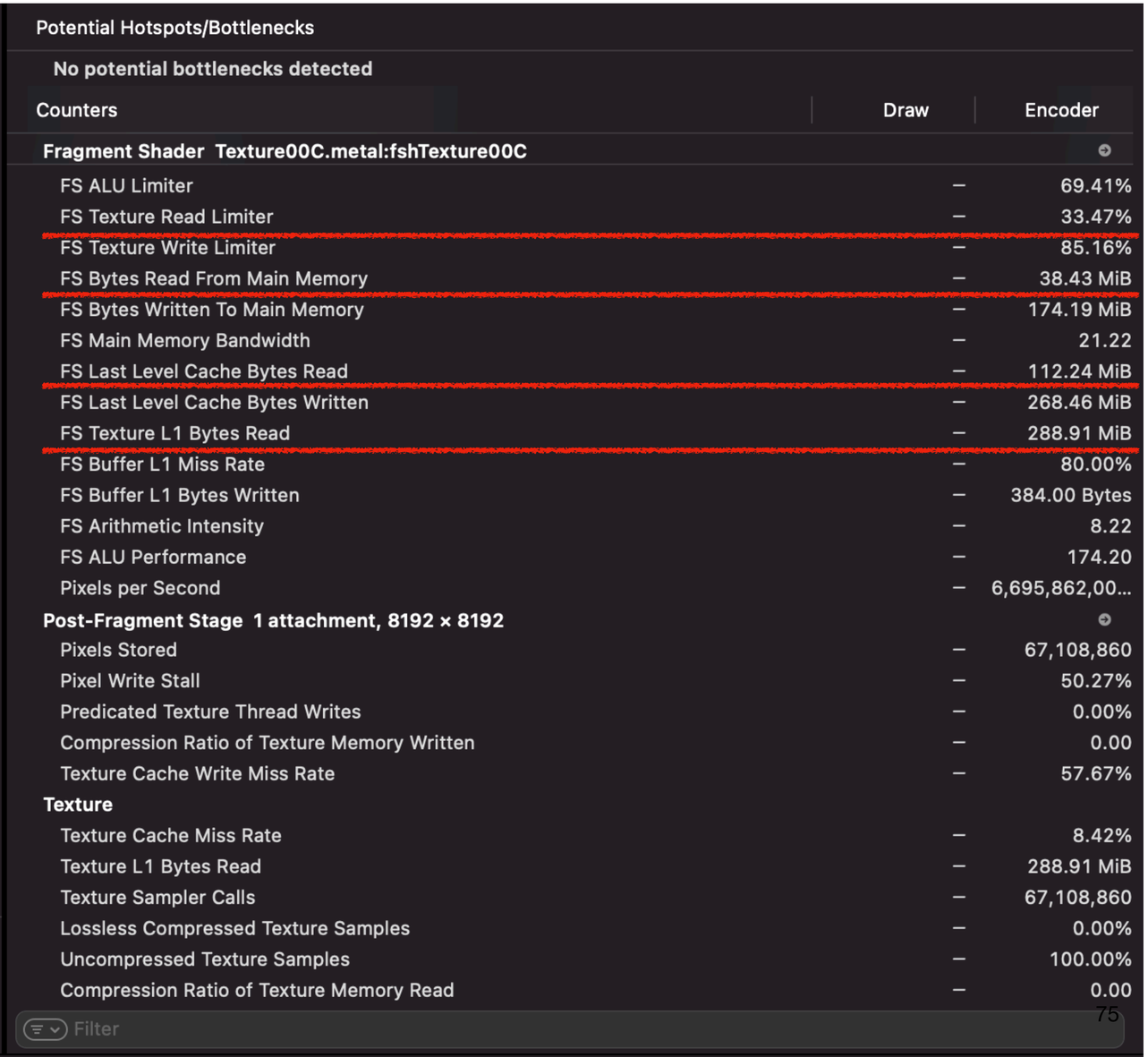


Localisation (8)

~ 10.64 ms

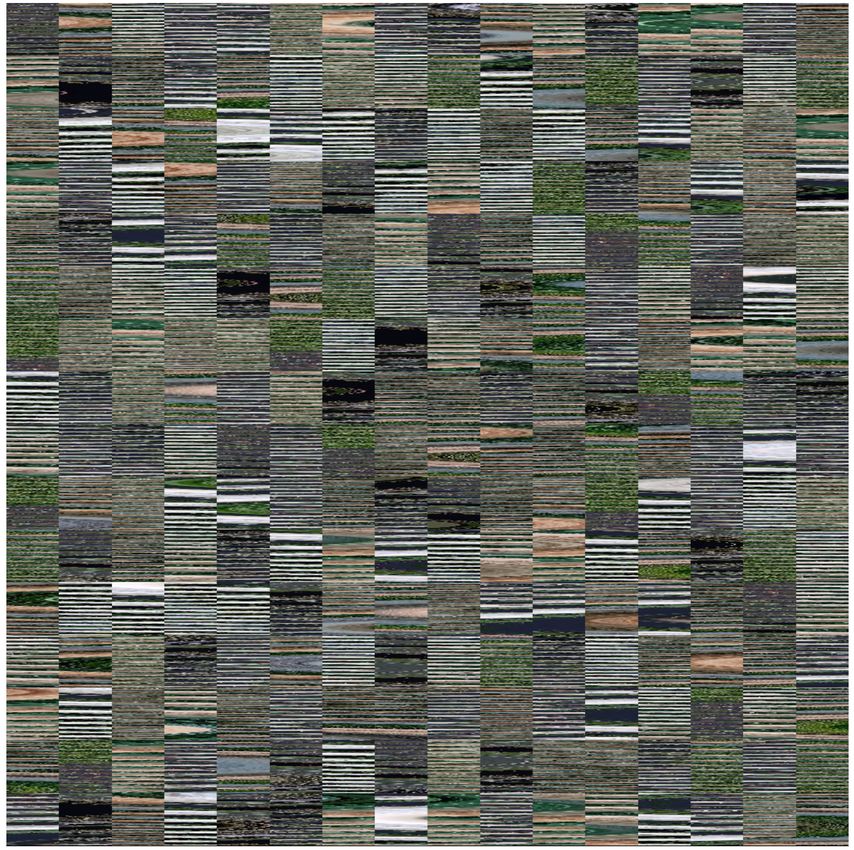
Localisation (8)



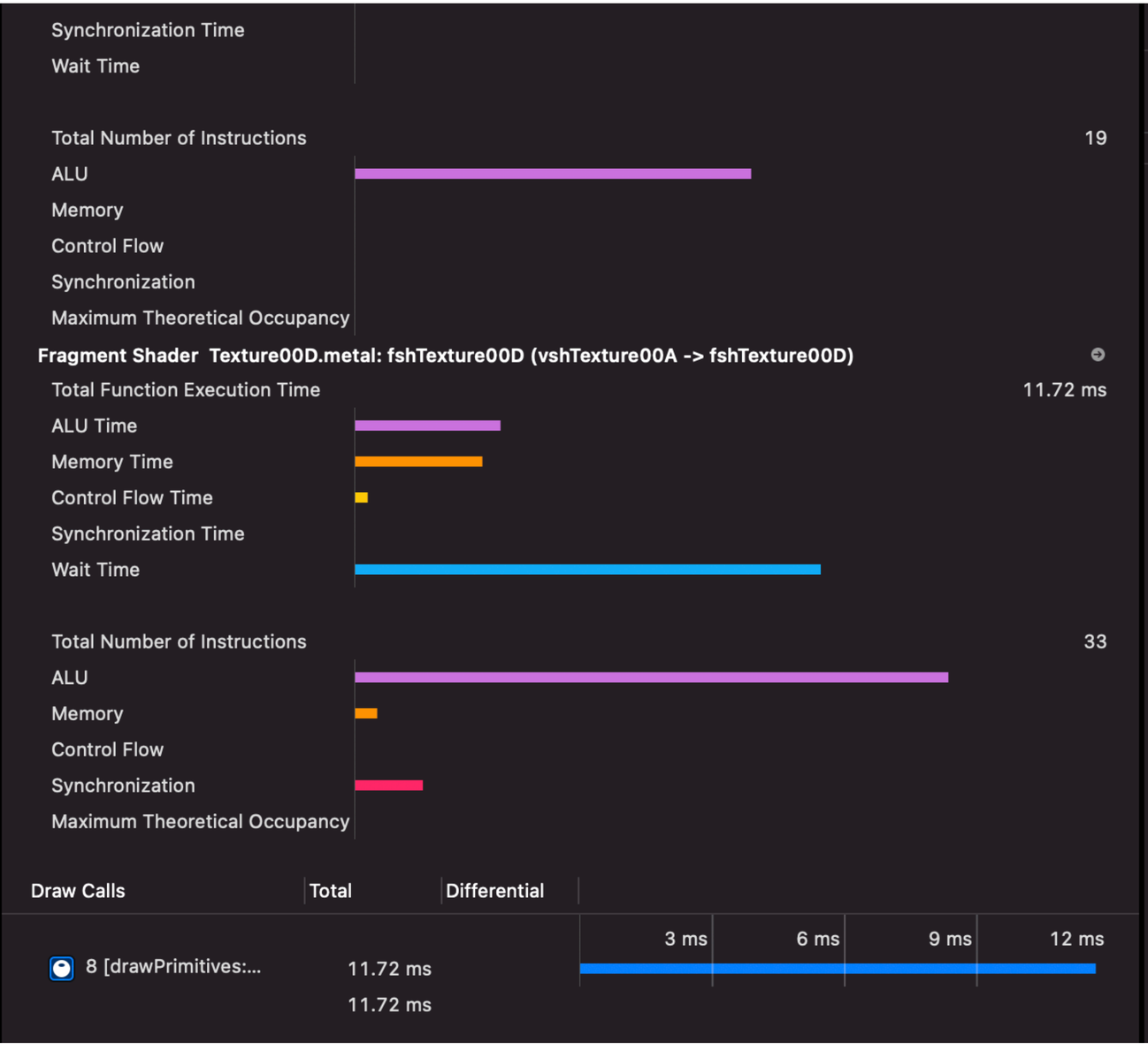


Localisation (16)

~ 13.91 ms



Localisation (16)



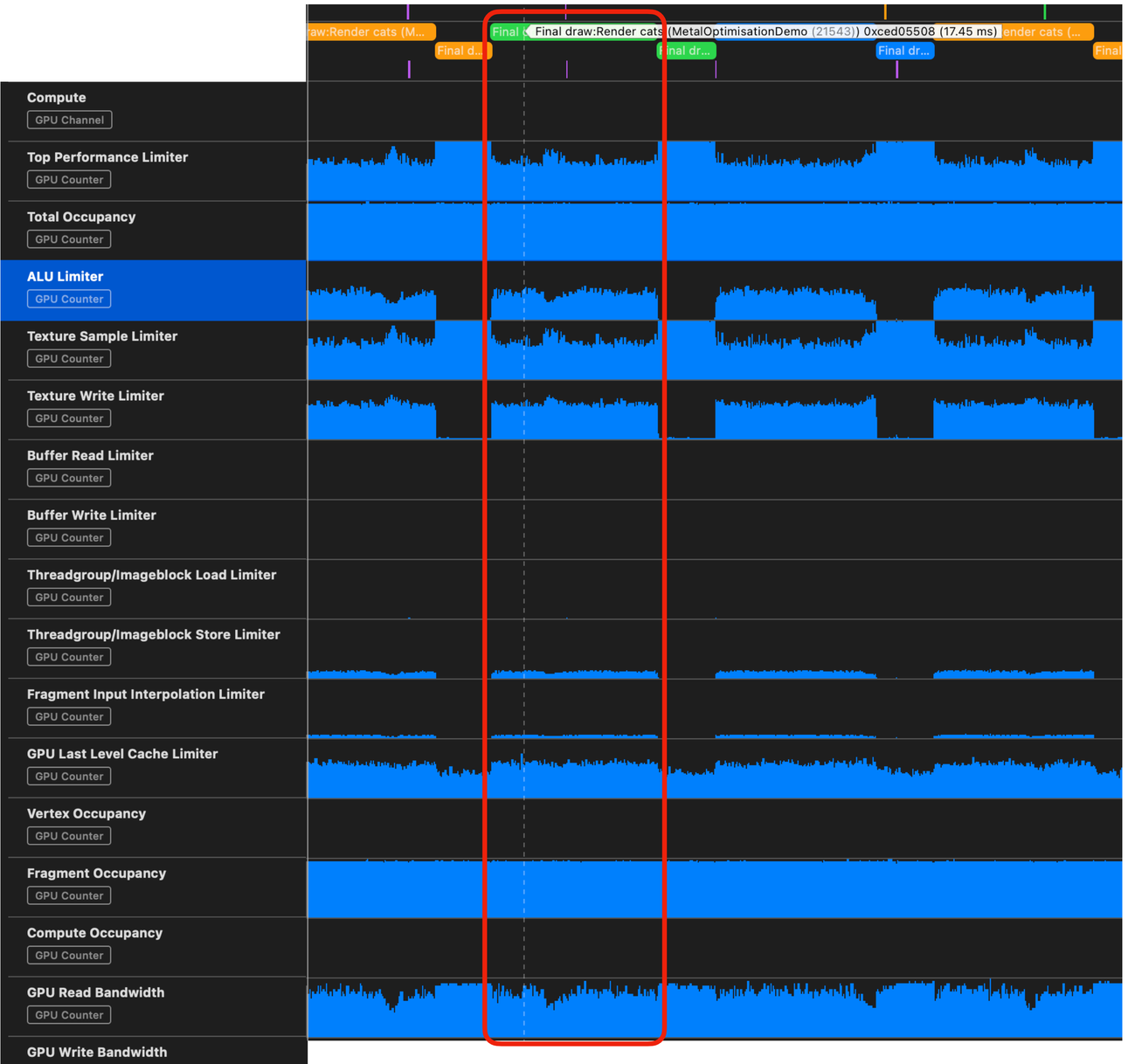
Potential Hotspots/Bottlenecks

No potential bottlenecks detected

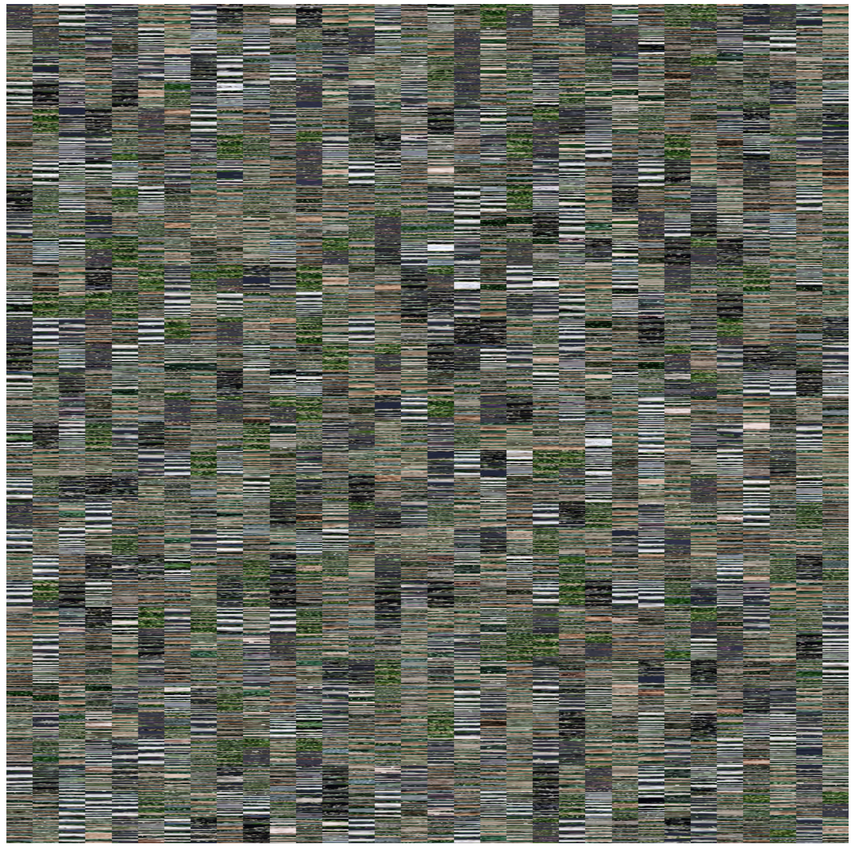
Counters	Draw	Encoder
Fragment Shader Texture00D.metal:fshTexture00D		
FS ALU Limiter	N/A	60.39%
FS Texture Read Limiter	N/A	60.36%
FS Texture Write Limiter	N/A	60.88%
FS Bytes Read From Main Memory	N/A	126.04 MiB
FS Bytes Written To Main Memory	N/A	196.89 MiB
FS Main Memory Bandwidth	N/A	27.07
FS Last Level Cache Bytes Read	N/A	322.04 MiB
FS Last Level Cache Bytes Written	N/A	268.46 MiB
FS Texture L1 Bytes Read	330.74 MiB	327.83 MiB
FS Buffer L1 Miss Rate	N/A	83.34%
FS Buffer L1 Bytes Written	N/A	384.00 Bytes
FS Arithmetic Intensity	N/A	5.41
FS ALU Performance	149.04	146.35
Pixels per Second	5,728,499,00...	5,625,432,00...
Post-Fragment Stage 1 attachment, 8192 × 8192		
Pixels Stored	67,108,860	67,108,860
Pixel Write Stall	N/A	29.83%
Predicated Texture Thread Writes	0.00%	0.00%
Compression Ratio of Texture Memory Written	N/A	0.00
Texture Cache Write Miss Rate	N/A	51.12%
Texture		
Texture Cache Miss Rate	N/A	23.32%
Texture L1 Bytes Read	330.74 MiB	327.83 MiB
Texture Sampler Calls	67,108,860	67,108,860
Lossless Compressed Texture Samples	N/A	0.00%
Uncompressed Texture Samples	N/A	100.00%
Compression Ratio of Texture Memory Read	N/A	0.00

Filter

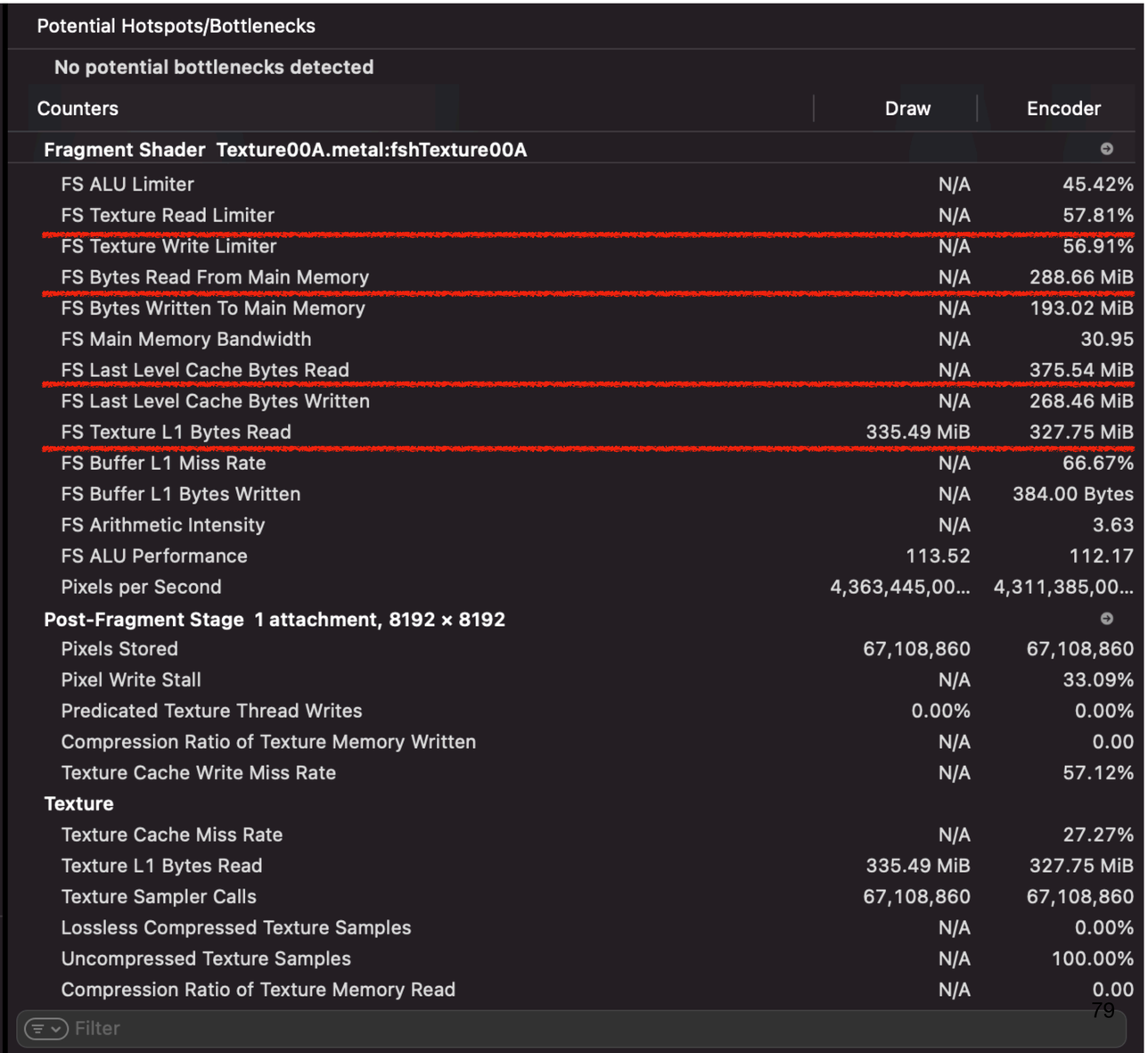
Localisation (32)



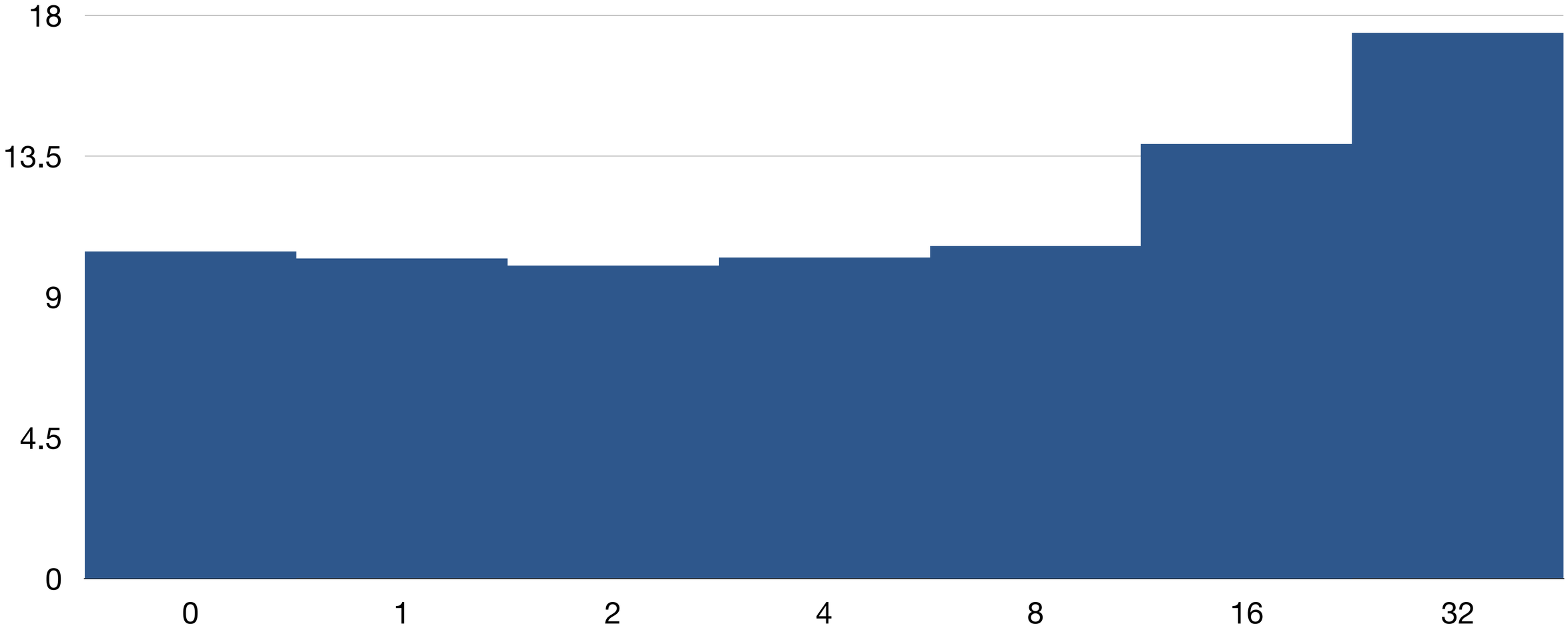
~ 17.45 ms



Localisation (32)



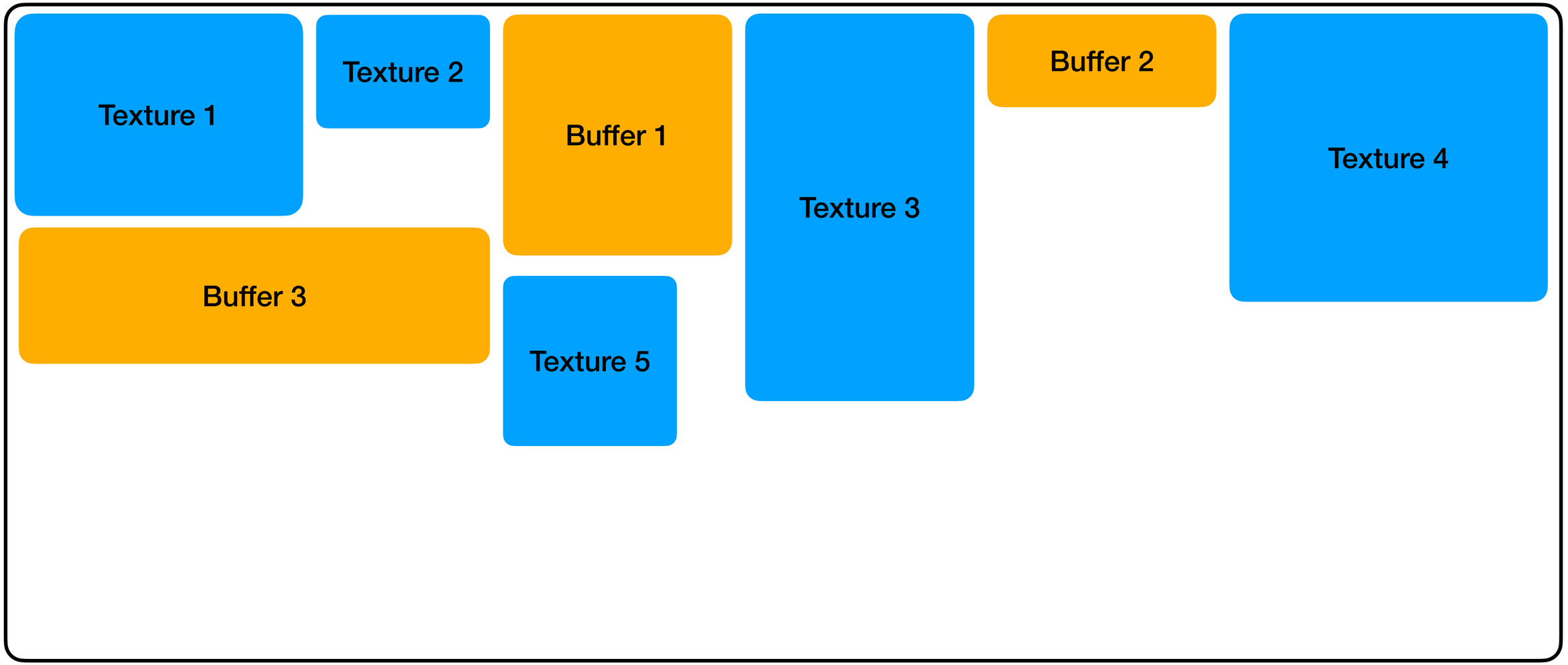
Summary (ms)



Memory

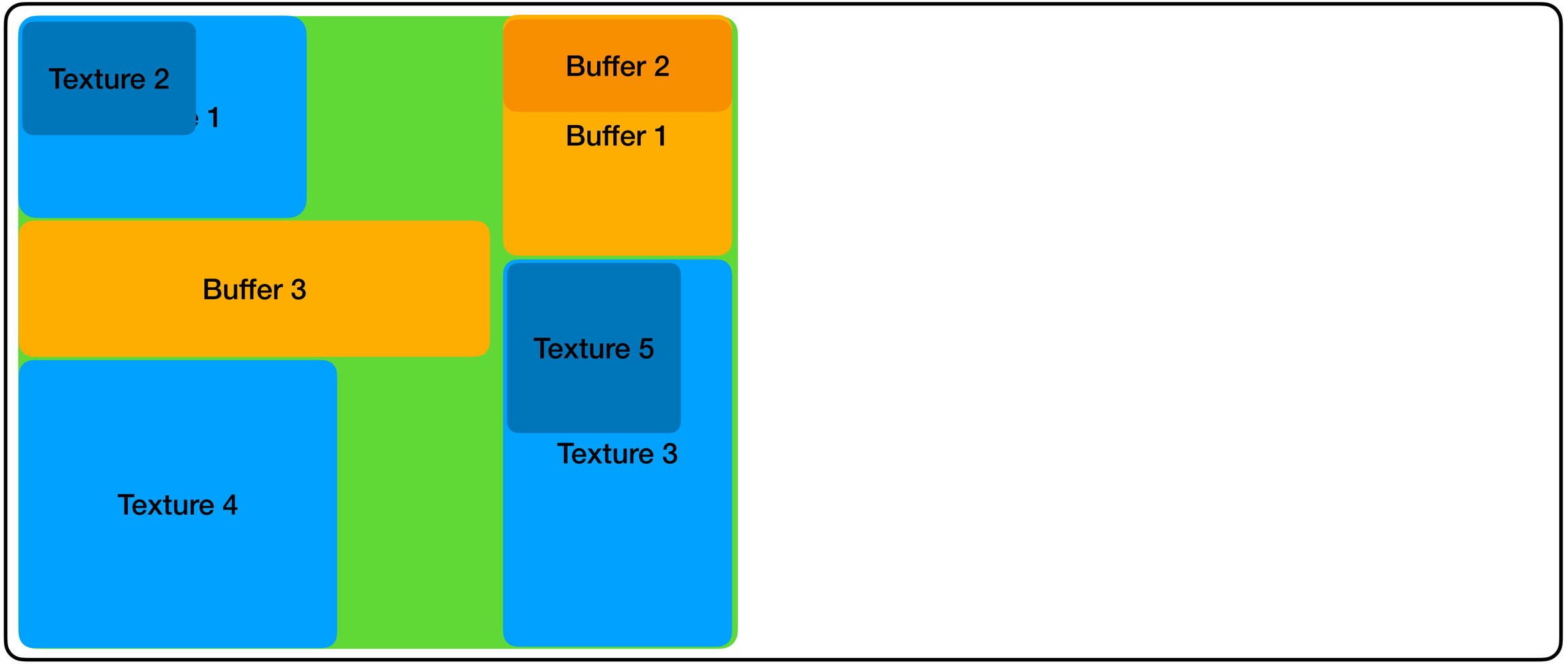
Heap and device

Device: N x (allocate + place)

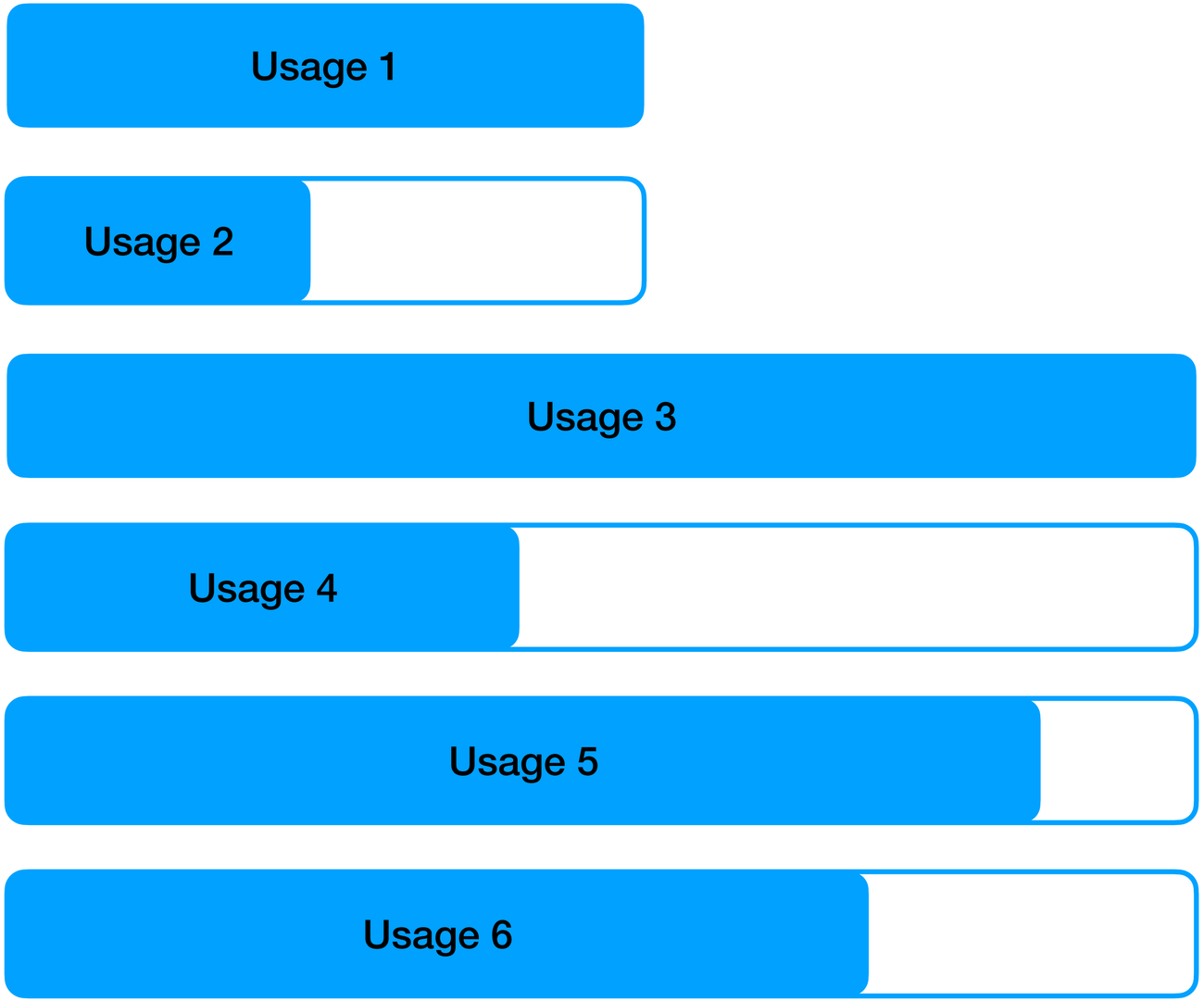


Heap and device

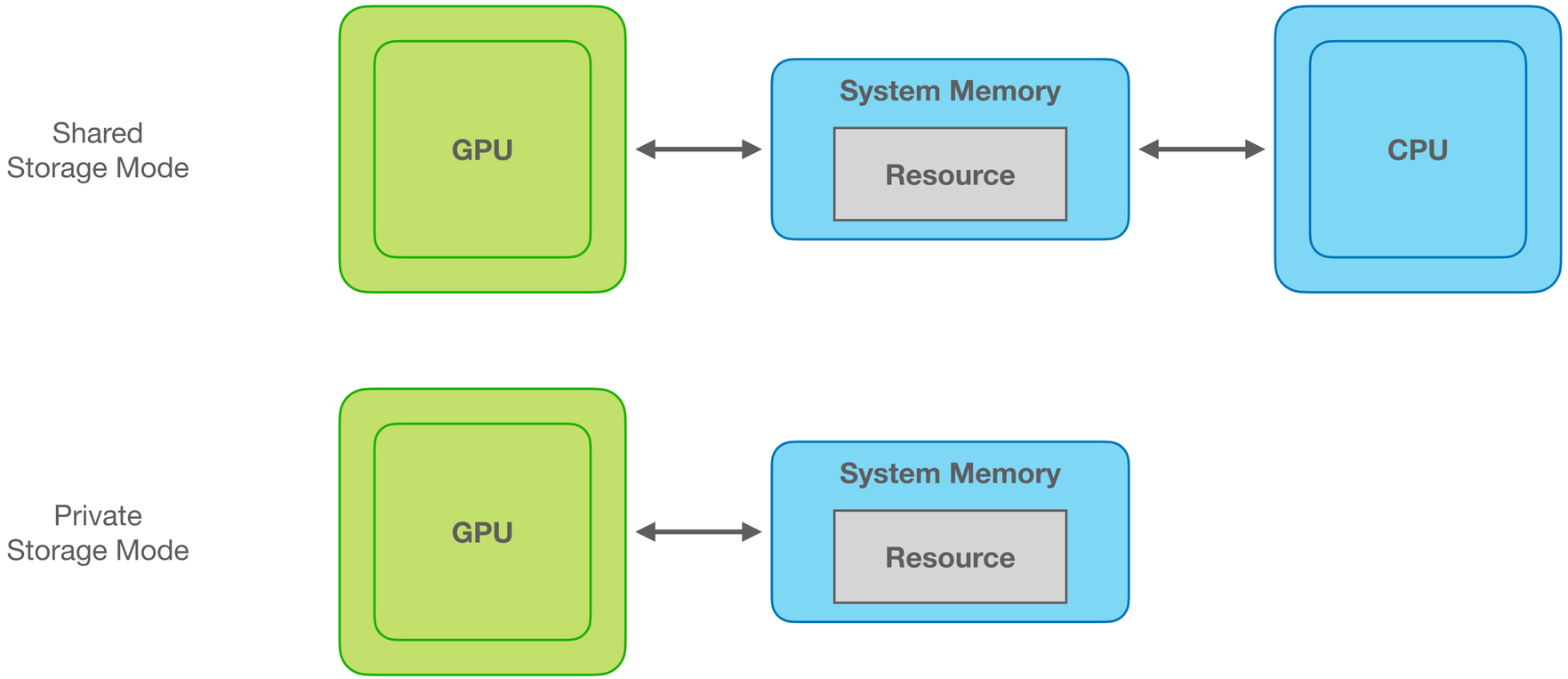
Heap: allocate + N x place



Reuse buffers



Private and shared



Several passes

Several passes

`sample`

```
// Pass 1
renderPassDescriptor.colorAttachments[0].texture = _textureTmp;
renderPassDescriptor.colorAttachments[0].loadAction = MTLLoadActionLoad;
id <MTLRenderCommandEncoder> encoder =
[commandBuffer renderCommandEncoderWithDescriptor:renderPassDescriptor];
[encoder setRenderPipelineState:_pipelineRender1];
[encoder setFragmentTexture:_texture atIndex:0];
[encoder drawPrimitives:MTLPrimitiveTypeTriangleStrip vertexStart:0 vertexCount:4];
[encoder endEncoding];

// Pass 2
renderPassDescriptor = [MTLRenderPassDescriptor new];
renderPassDescriptor.colorAttachments[0].texture = _textureOut;
renderPassDescriptor.colorAttachments[0].loadAction = MTLLoadActionLoad;
id <MTLRenderCommandEncoder> encoder =
[commandBuffer renderCommandEncoderWithDescriptor:renderPassDescriptor];
[encoder setRenderPipelineState:_pipelineRender2];
[encoder setFragmentTexture:_textureTmp atIndex:0];
[encoder drawPrimitives:MTLPrimitiveTypeTriangleStrip vertexStart:0 vertexCount:4];
[encoder endEncoding];
```

Several passes

`sample`

```
fragment float4 fshTexture01A_1(ColorInOut in [[stage_in]], texture2d<float> image [[ texture(0) ]])
{
    constexpr sampler imageSampler(address::repeat, filter::nearest);
    float2 uv = in.texCoord;
    float4 src = image.sample(imageSampler, uv);
    return src * src;
}
```

```
fragment float4 fshTexture01A_2(ColorInOut in [[stage_in]], texture2d<float> image [[ texture(0) ]])
{
    constexpr sampler imageSampler(address::repeat, filter::nearest);
    float2 uv = in.texCoord;
    float4 src = image.sample(imageSampler, uv);
    return sqrt(src);
}
```

Several passes

`sample`

Several passes

attachment

```
MTLRenderPassDescriptor* renderPassDescriptor = [MTLRenderPassDescriptor new];
renderPassDescriptor.colorAttachments[0].texture = _textureOut;
renderPassDescriptor.colorAttachments[0].loadAction = MTLLoadActionLoad;
renderPassDescriptor.colorAttachments[1].texture = _textureTmp;
renderPassDescriptor.colorAttachments[1].loadAction = MTLLoadActionLoad;
if (renderPassDescriptor != nil) {
    id <MTLRenderCommandEncoder> encoder =
    [commandBuffer renderCommandEncoderWithDescriptor:renderPassDescriptor];
    encoder.label = @"Render cats";

    [encoder setRenderPipelineState:_pipelineRender1];
    [encoder setFragmentTexture:_texture atIndex:0];
    [encoder drawPrimitives:MTLPrimitiveTypeTriangleStrip vertexStart:0 vertexCount:4];

    [encoder setRenderPipelineState:_pipelineRender2];
    [encoder drawPrimitives:MTLPrimitiveTypeTriangleStrip vertexStart:0 vertexCount:4];

    [encoder endEncoding];
}
```

Several passes

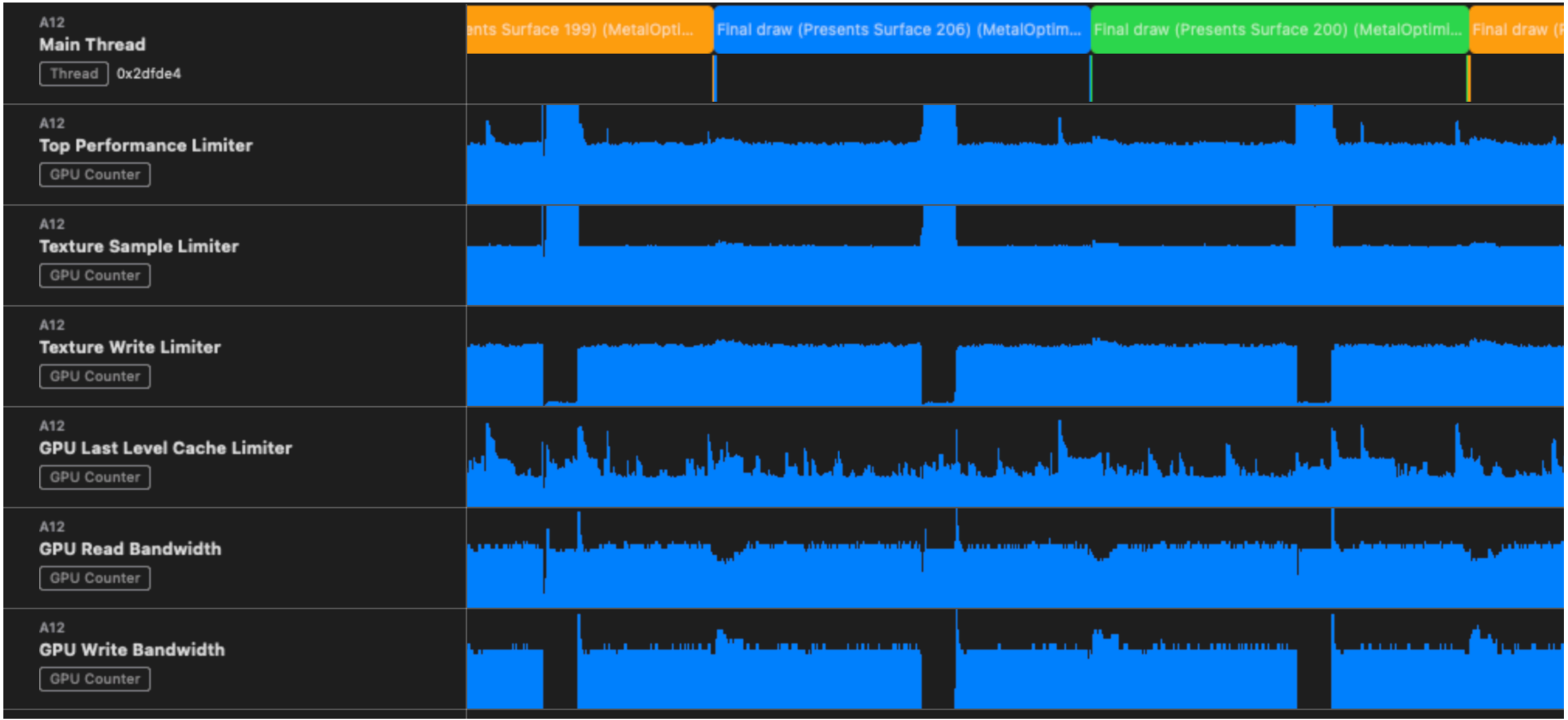
attachment

```
typedef struct {
   float4 dst [[ color(1) ]];
} FragmentOut1;


typedef struct {
   float4 dst [[ color(0) ]];
} FragmentOut2;
```

```
fragment FragmentOut1 fshTexture01B_1(
   ColorInOut in [[stage_in]],
   texture2d<float> image [[ texture(0) ]])
{
    constexpr sampler imageSampler(address::repeat, filter::nearest);

    float2 uv = in.texCoord;
    float4 src = image.sample(imageSampler, uv);
    return {src * src};
}



fragment FragmentOut2 fshTexture01B_2(
   ColorInOut in [[stage_in]],
   const float4 image [[ color(1) ]])
{
    float4 src = image;
    return {sqrt(src)};
}
```

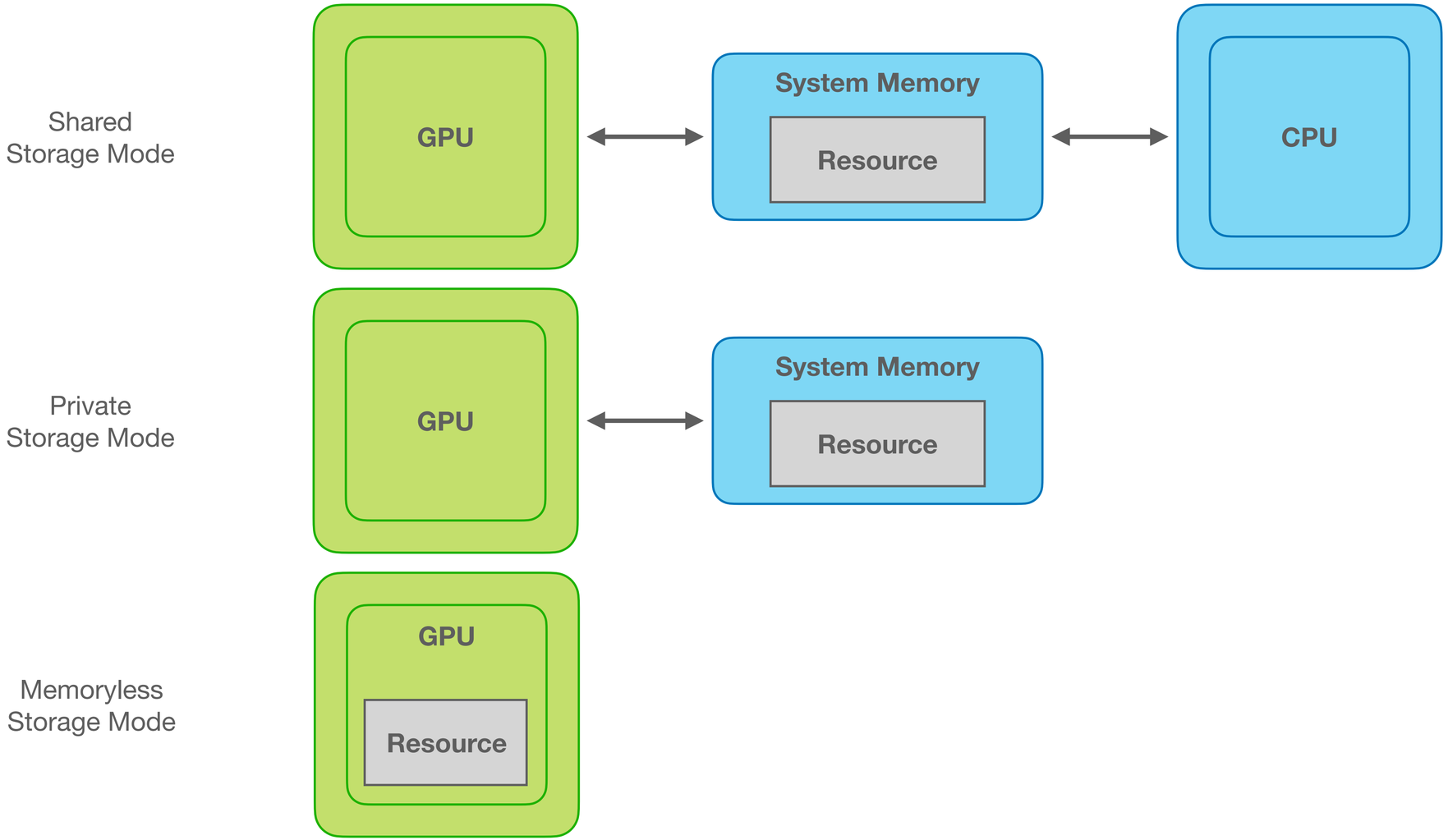
Several passes

attachment



Several passes

memoryless textures



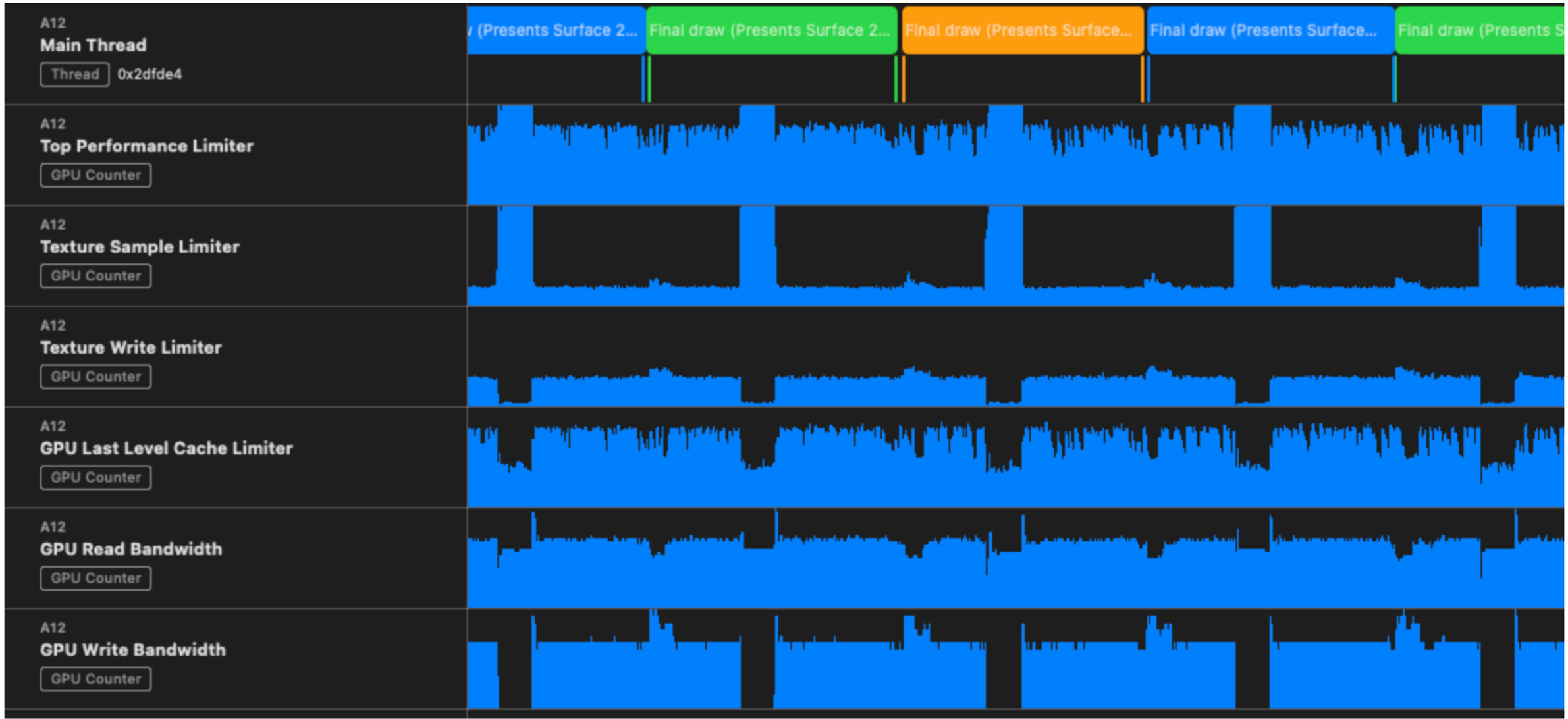
Several passes

memoryless textures

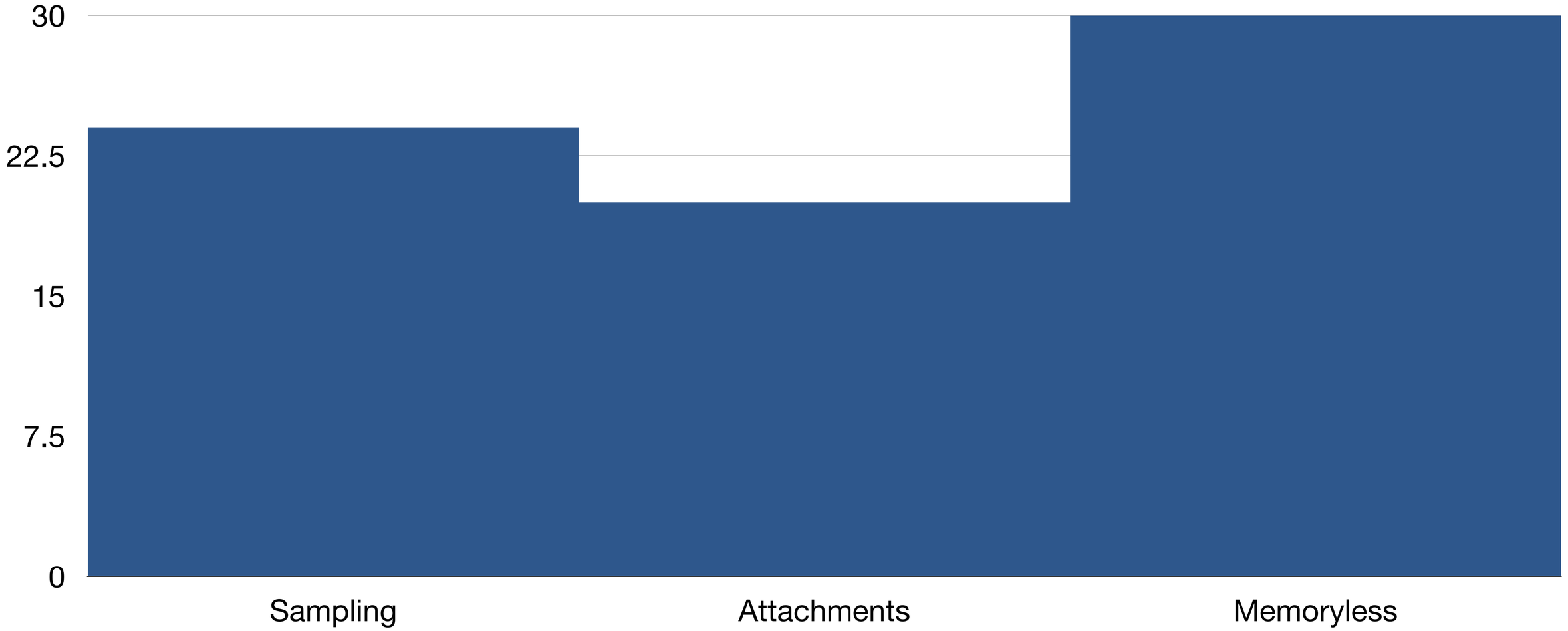
```
MTLRenderPassDescriptor* renderPassDescriptor = [MTLRenderPassDescriptor new];
renderPassDescriptor.colorAttachments[0].texture = _textureOut;
renderPassDescriptor.colorAttachments[0].loadAction = MTLLoadActionLoad;
renderPassDescriptor.colorAttachments[1].texture = _textureTmp;
renderPassDescriptor.colorAttachments[1].loadAction = MTLLoadActionDontCare;
renderPassDescriptor.colorAttachments[1].storeAction = MTLStoreActionDontCare;
if (renderPassDescriptor != nil) {
   id <MTLRenderCommandEncoder> encoder =
   [commandBuffer renderCommandEncoderWithDescriptor:renderPassDescriptor];
   encoder.label = @"Render cats";

   [encoder setRenderPipelineState:_pipelineRender1];
   [encoder setFragmentTexture:_texture atIndex:0];
   [encoder drawPrimitives:MTLPrimitiveTypeTriangleStrip vertexStart:0 vertexCount:4];

   [encoder setRenderPipelineState:_pipelineRender2];
   [encoder drawPrimitives:MTLPrimitiveTypeTriangleStrip vertexStart:0 vertexCount:4];

   [encoder endEncoding];
}
```

Several passes

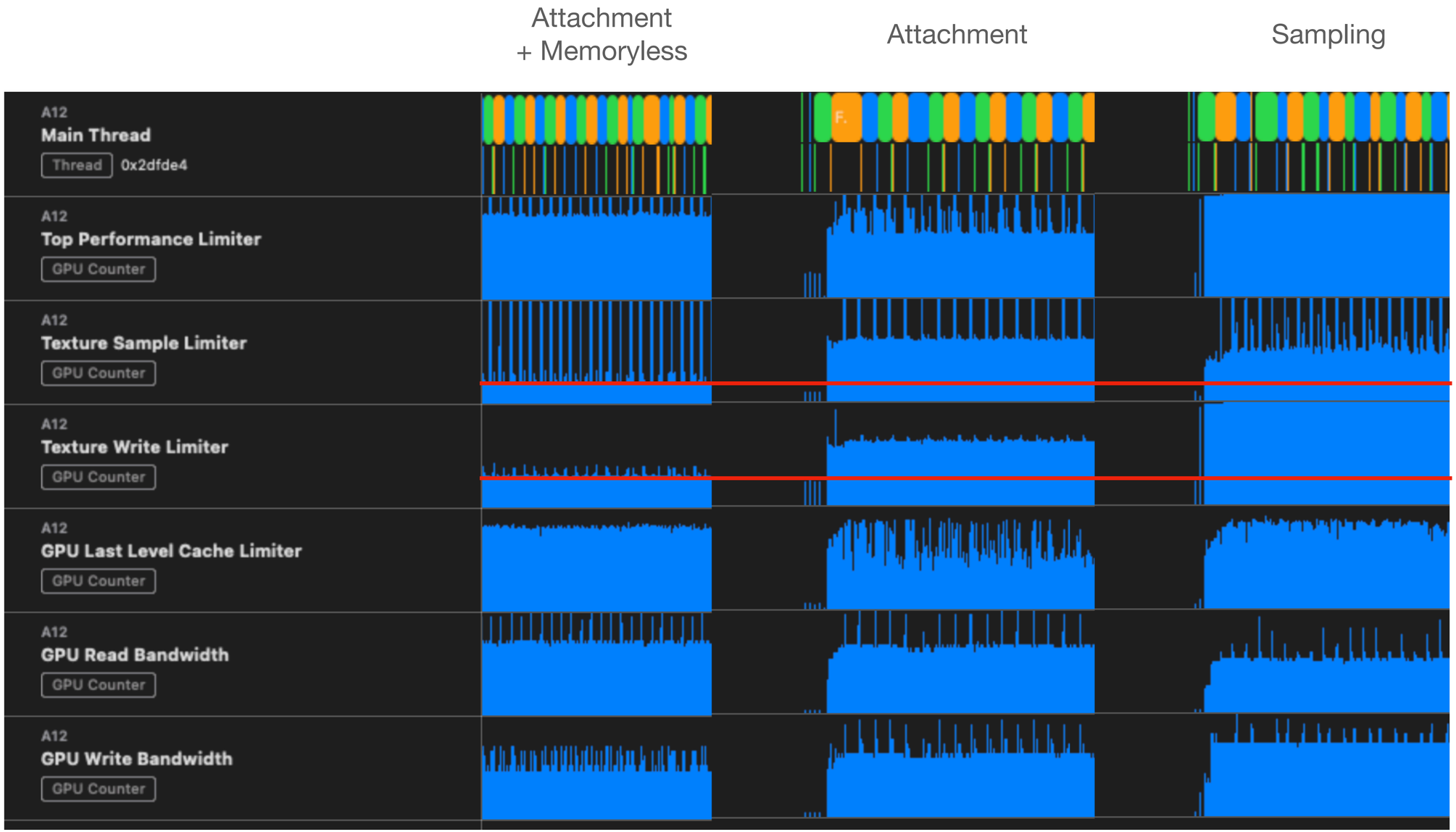
memoryless textures



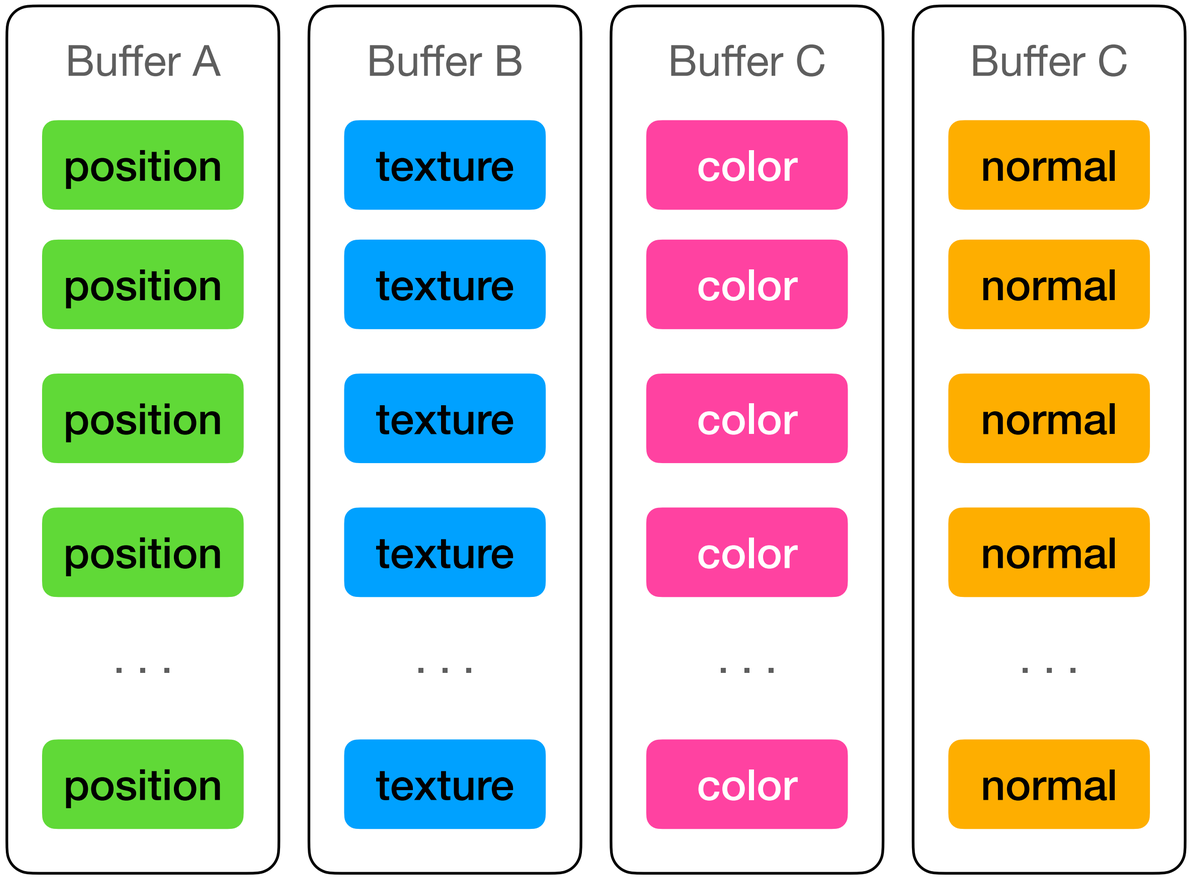
Summary (FPS)



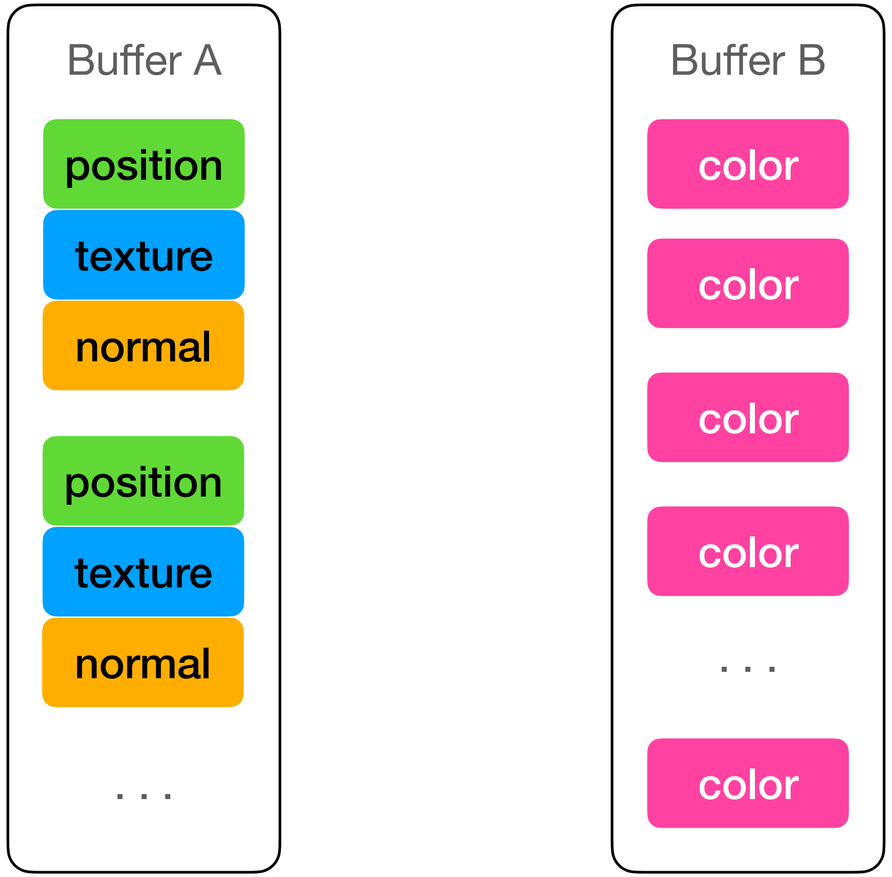
Summary


Attachment
+ Memoryless
Attachment
Sampling
A12
Main Thread
Thread
0x2dfde4
A12
Top Performance Limiter
GPU Counter
A12
Texture Sample Limiter
GPU Counter
A12
Texture Write Limiter
GPU Counter
A12
GPU Last Level Cache Limiter
GPU Counter
A12
GPU Read Bandwidth
GPU Counter
A12
GPU Write Bandwidth
GPU Counter

Pack your data optimally

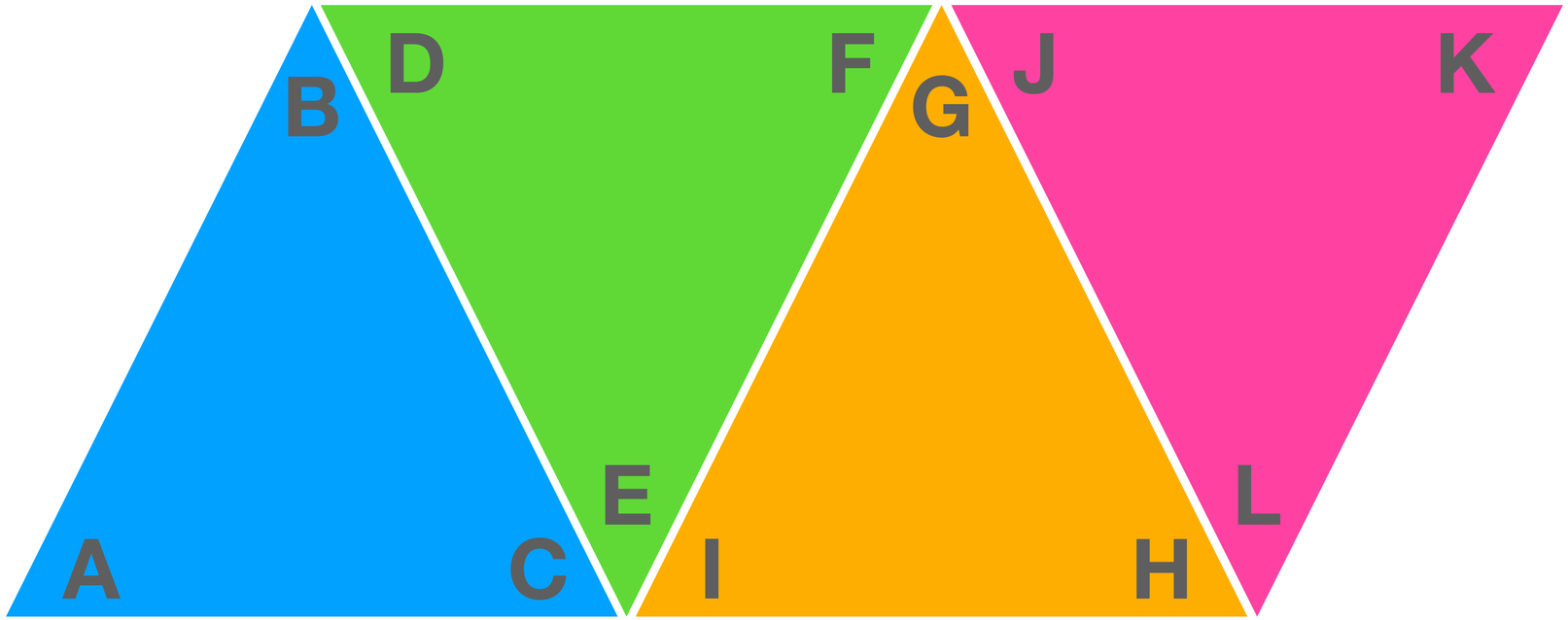


Pack your data optimally



Pack your data optimally

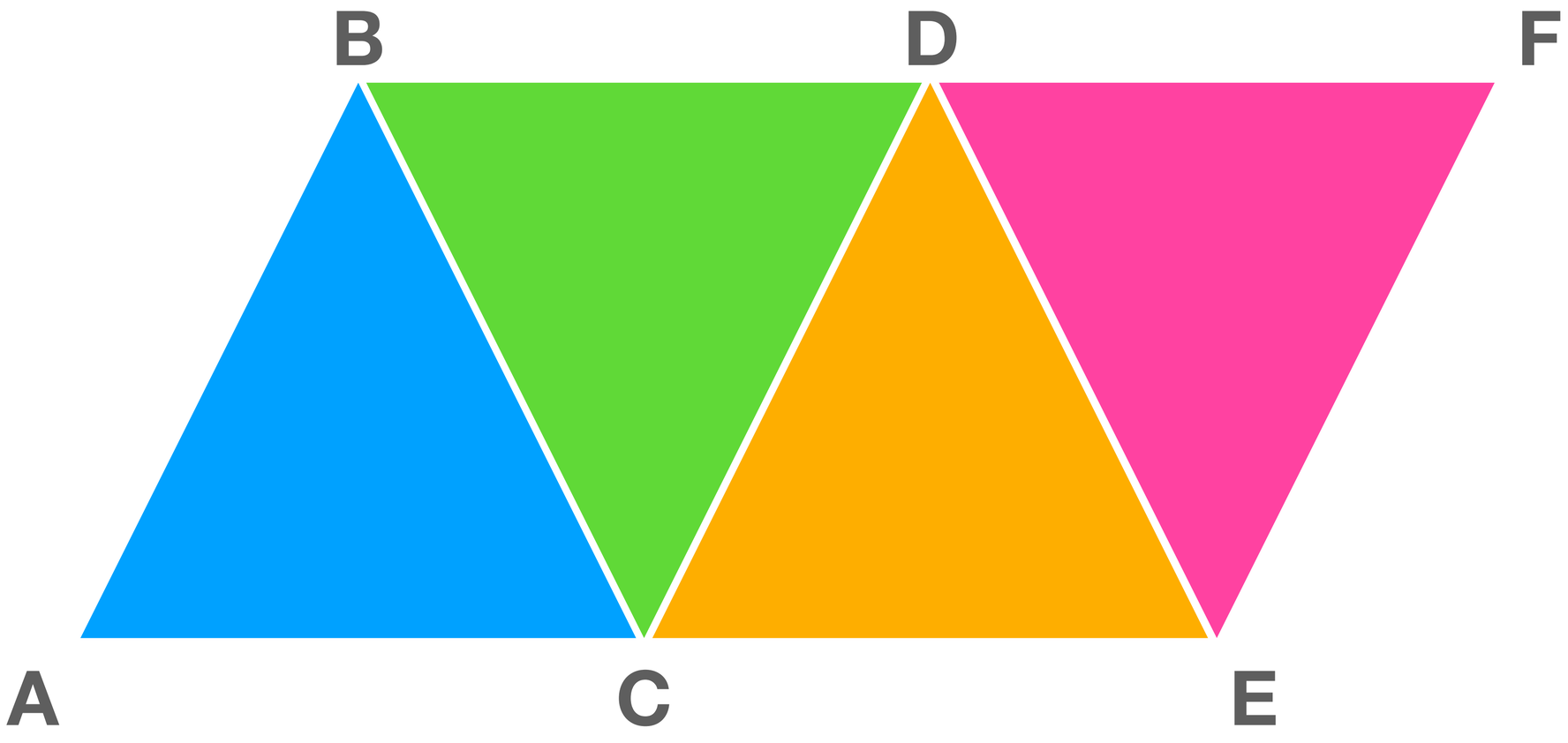
triangles	4
vertices	12



Pack your data optimally

triangles 4

vertices 6

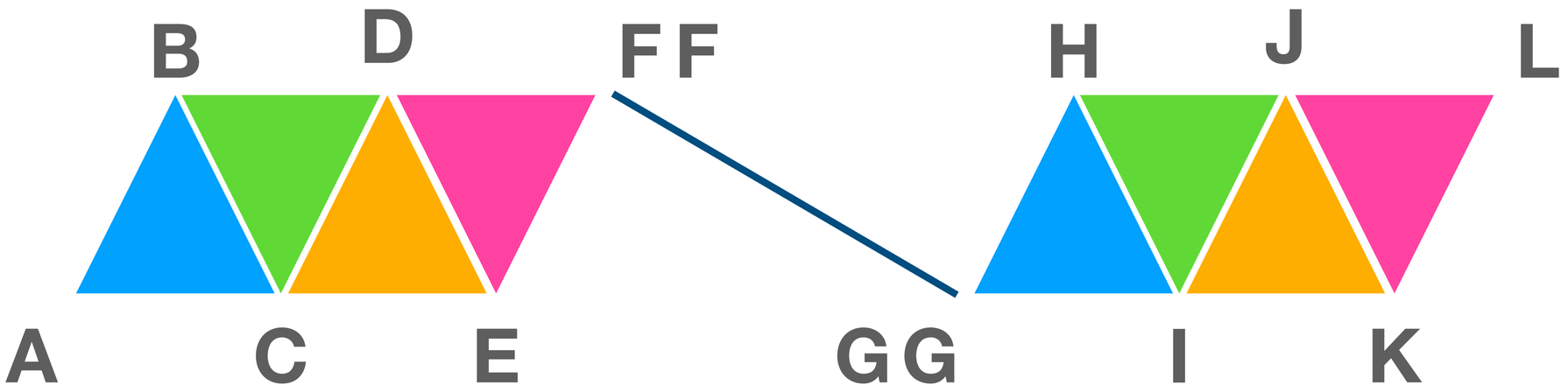


Pack your data optimally

draw calls	2
triangles	8
vertices	12

Pack your data optimally

draw calls	1
triangles	8
vertices	14

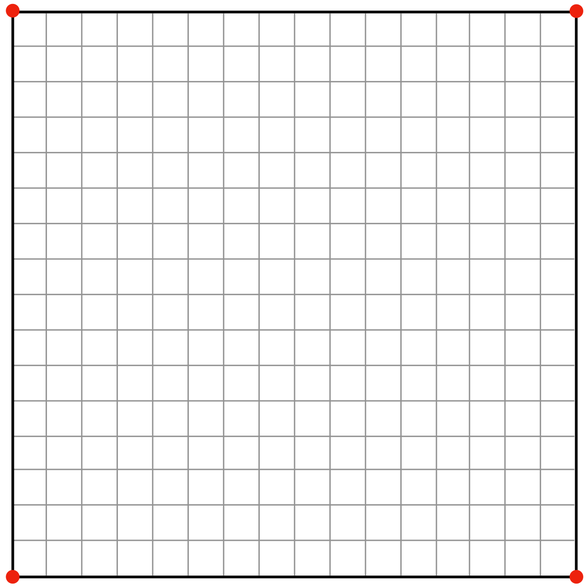


Parallel Computing

Compute - not render

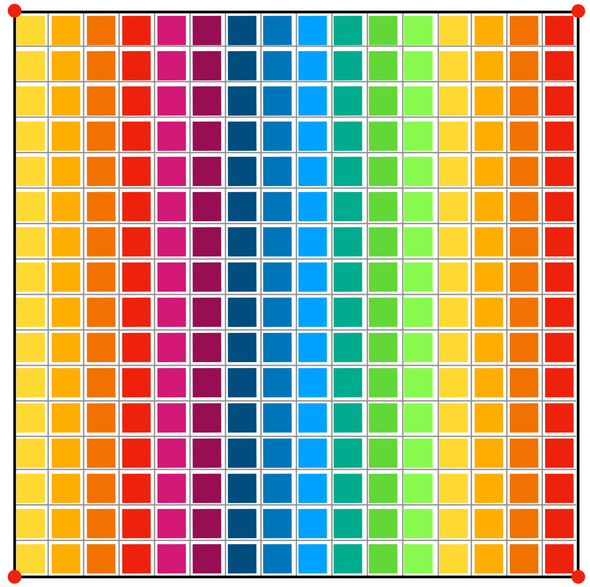
Use compute for pixel to pixel processing

Vertex



Rasterisation

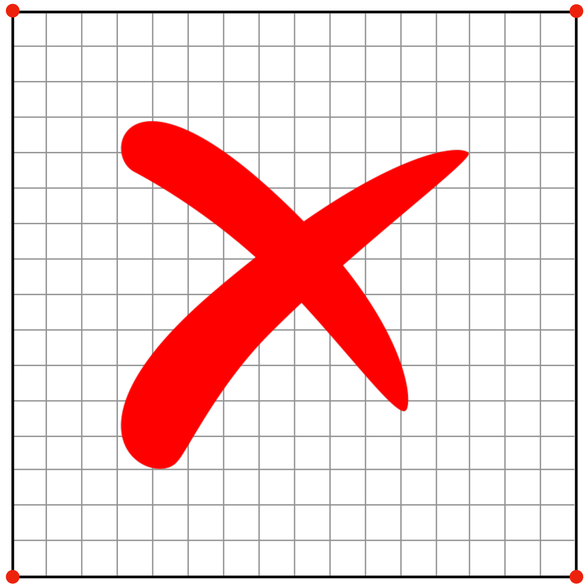
Fragment



Compute - not render

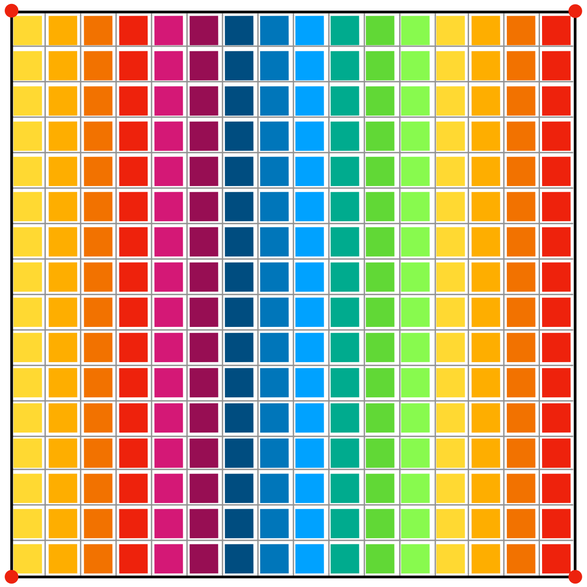
Use compute for pixel to pixel processing

Vertex

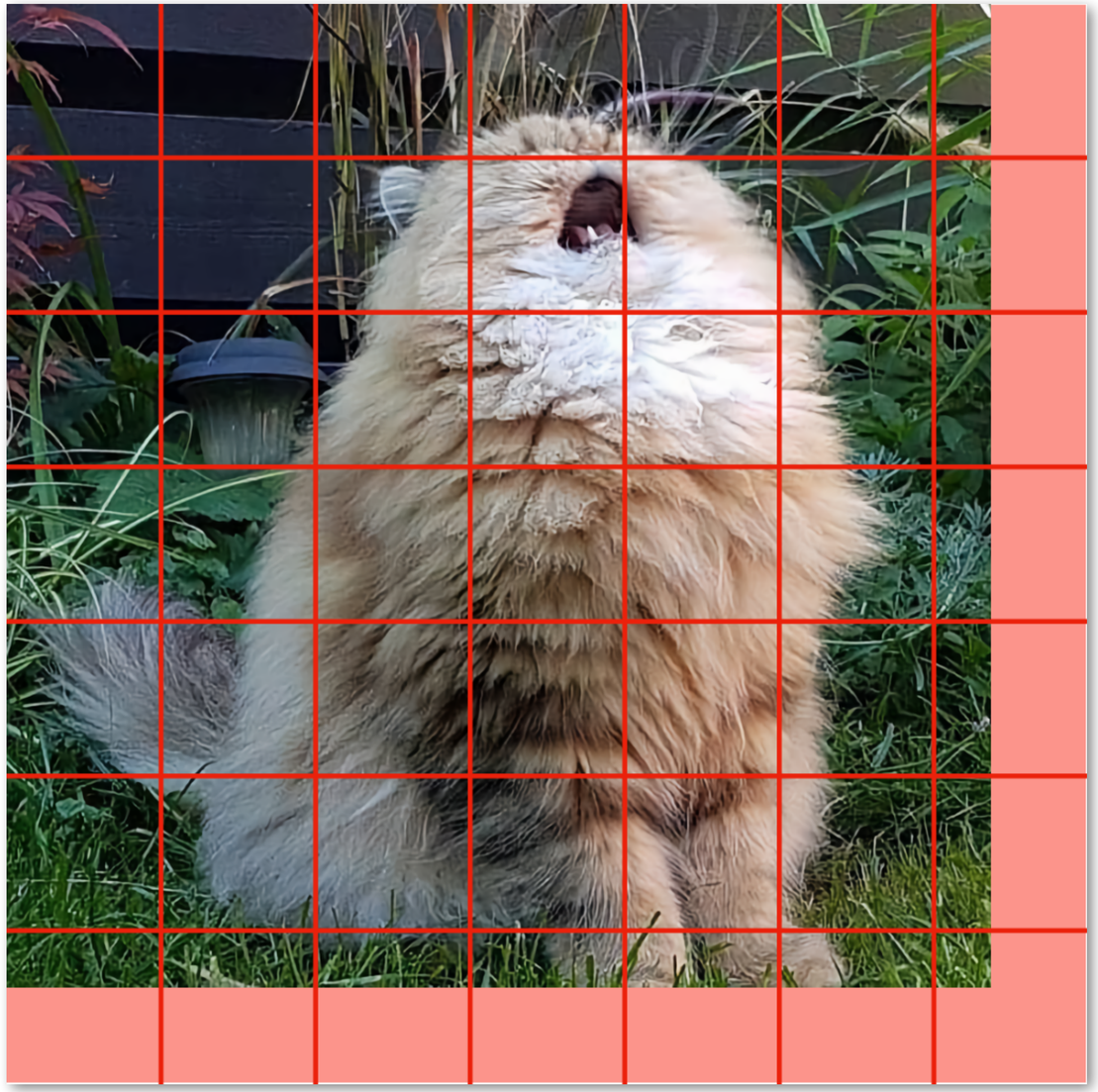


Rasterization

Compute



dispatchThreadgroups

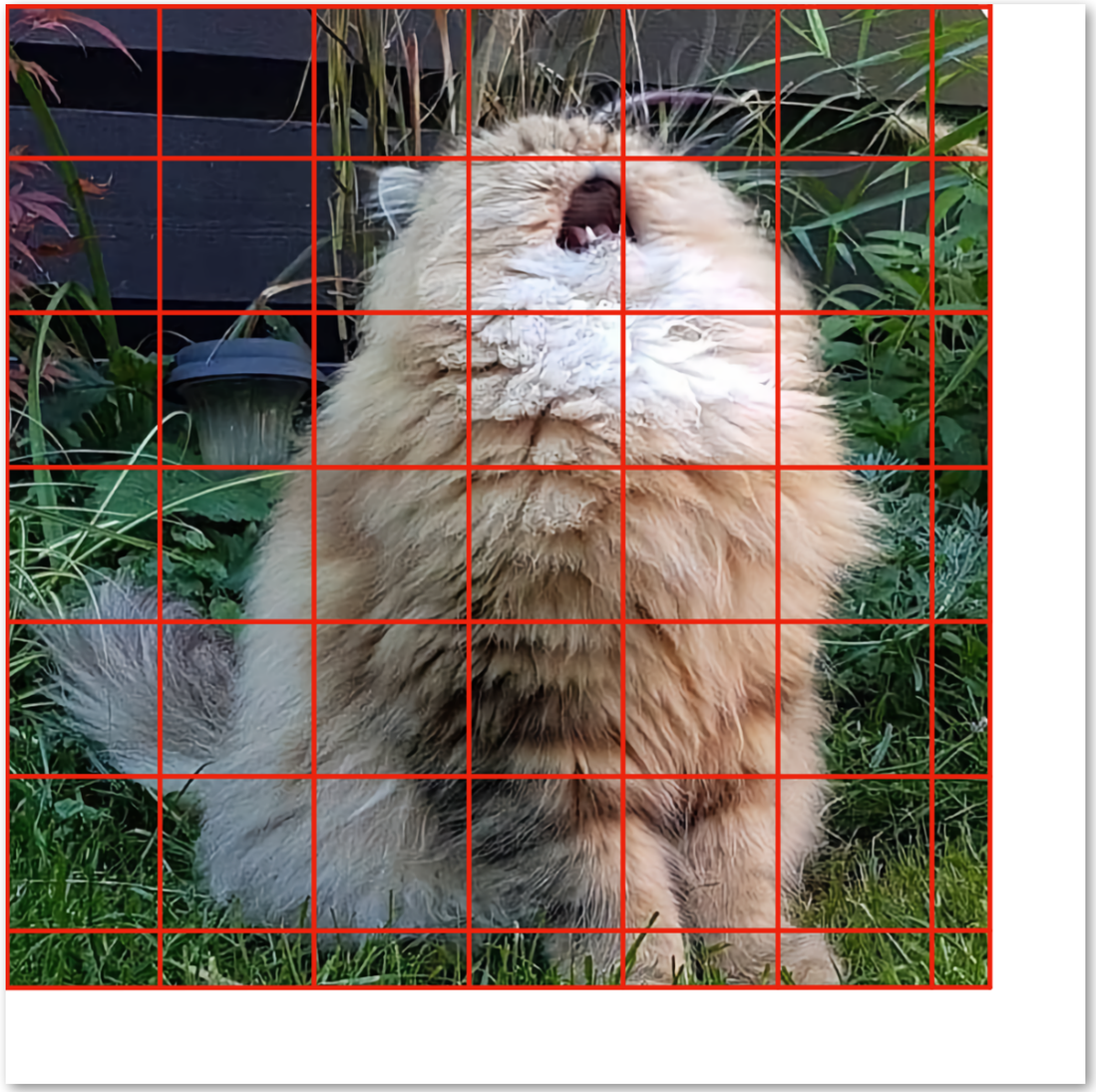


dispatchThreadgroups

```
kernel void krnFunction(texture2d<float, access::read> in [[ texture(0) ]],
                        texture2d<float, access::write> out [[ texture(1) ]],
                       uint2 gid [[thread_position_in_grid]])
{
   uint2 buf_size = uint2(in.get_width(), in.get_height());
   if (any(gid >= buf_size)) {
      return;
   }

   float4 src = in.read(gid);
   out.write(src * src, gid);
}
```

dispatchThreads

since A11 only, but…

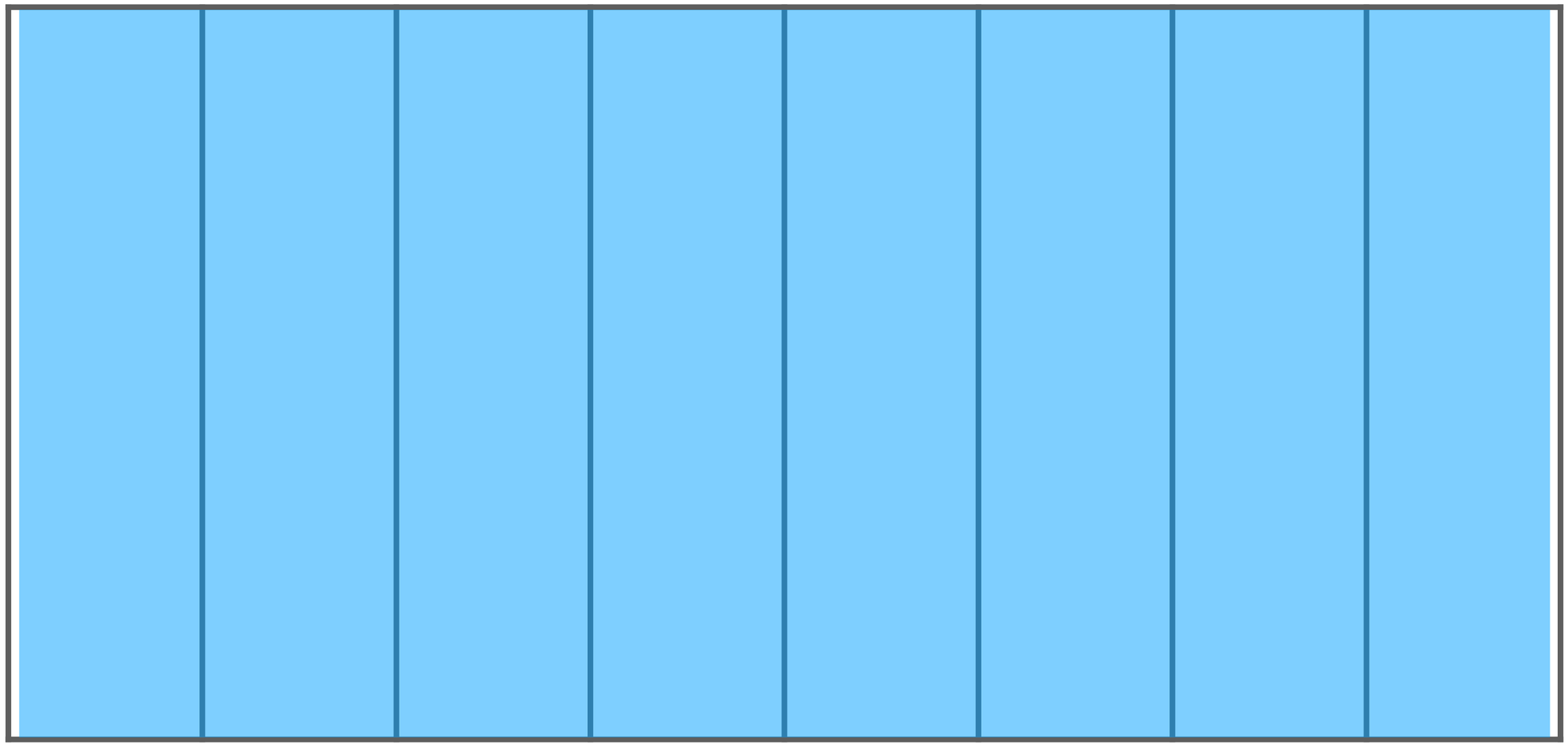


Size of thread group

```
NSUInteger w = _pipelineCompute.threadExecutionWidth;
NSUInteger h = _pipelineCompute.maxTotalThreadsPerThreadgroup / w;
```

Size of thread group

The best 👍🏻

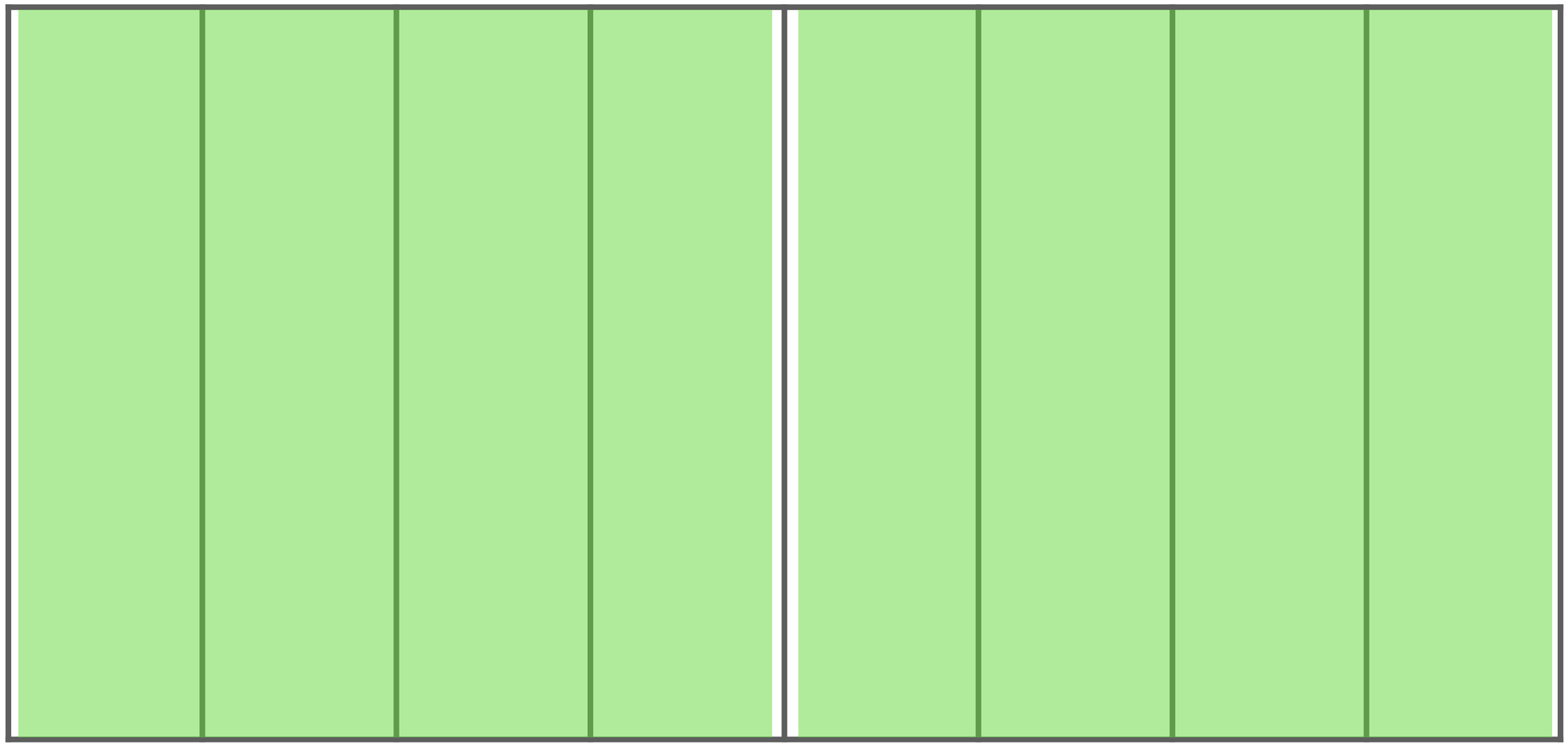


Size of thread group

The best 👍🏻

Size of thread group

OK 🤷🏼

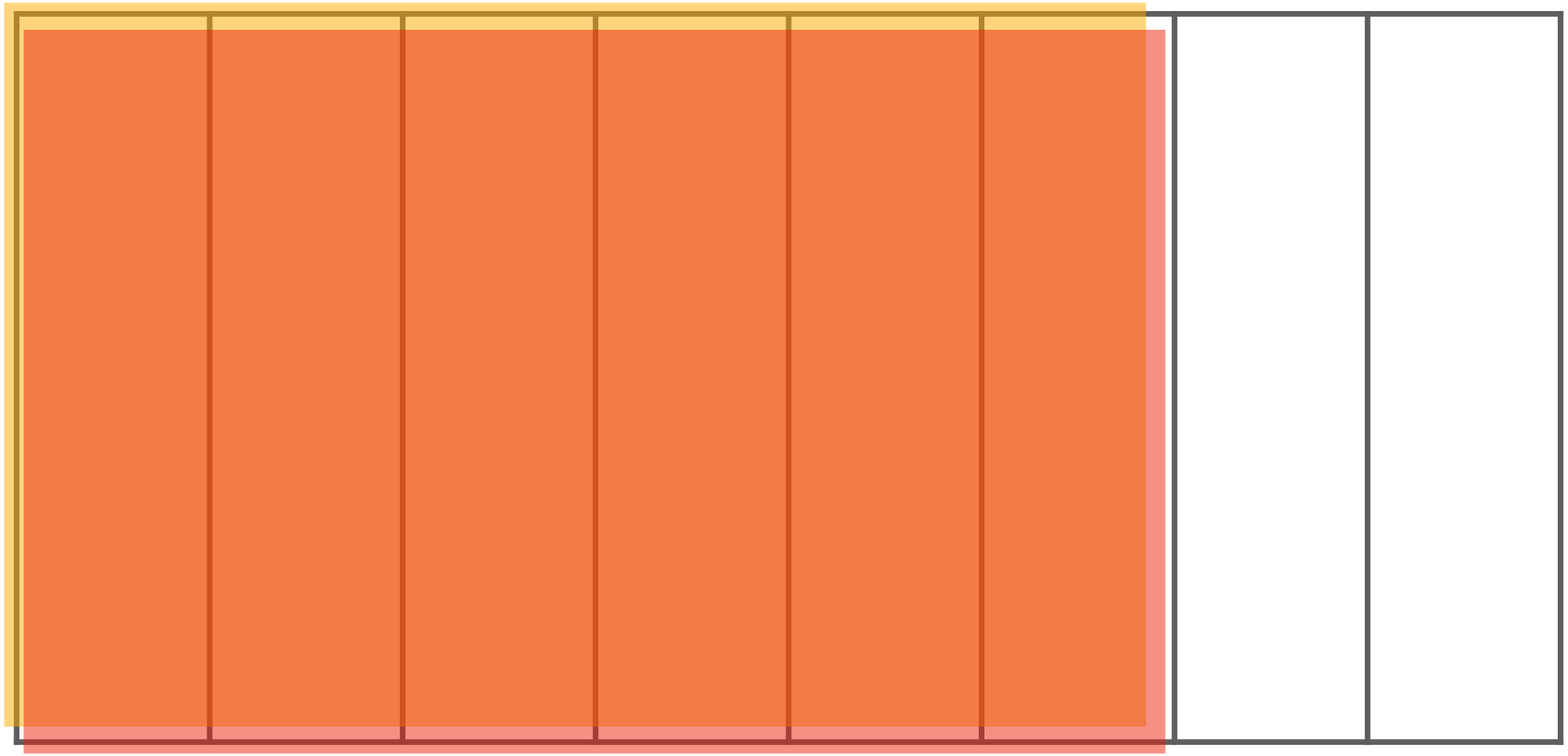


Size of thread group

OK 🤷🏼

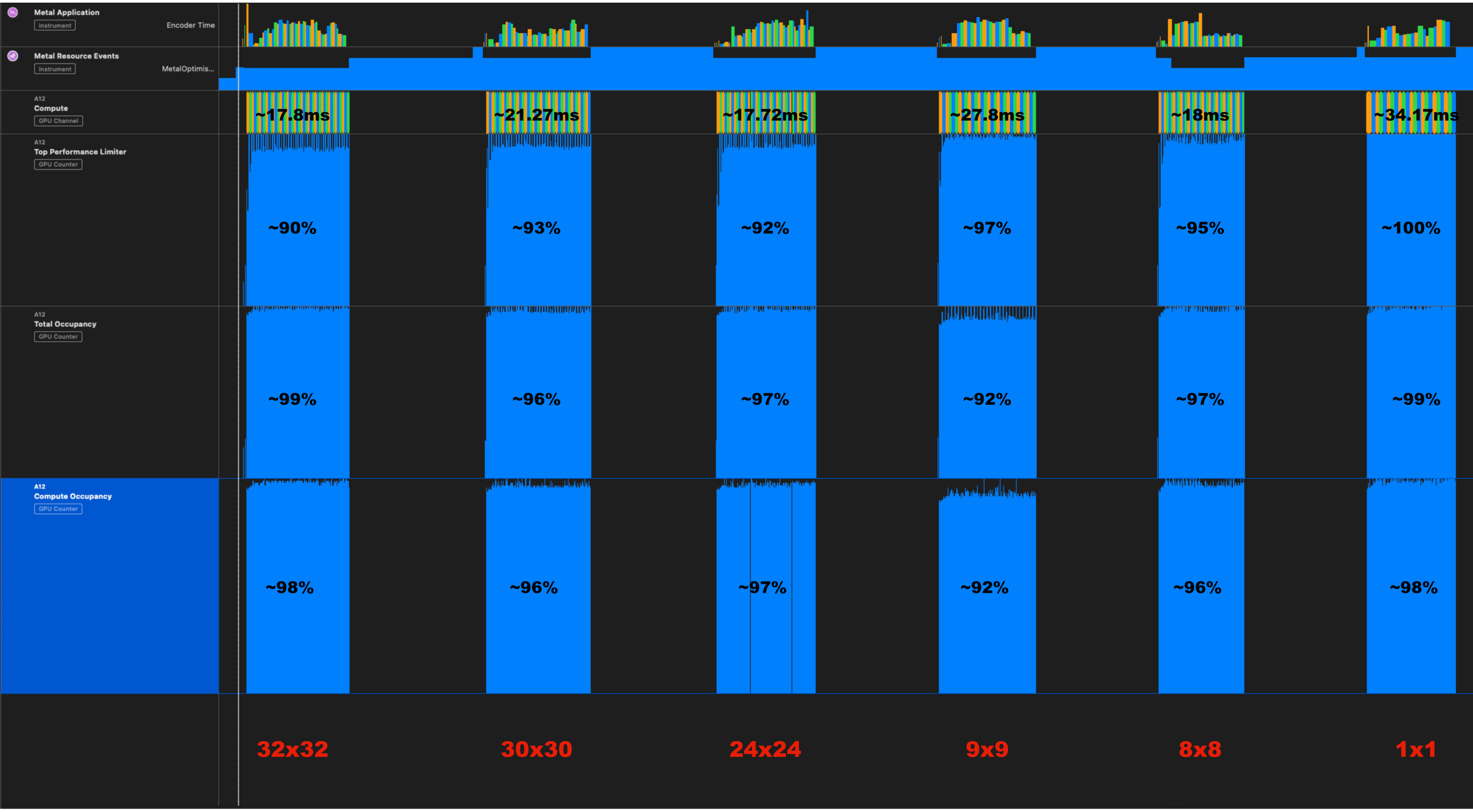
Size of thread group

NO🙅🏼



Size of thread group

NO🙅


Metal Application
Instrument
Encoder Time
Metal Resource Events
Instrument
MetalOptimis...
A12
Compute
GPU Channel
~17.8ms
~21.27ms
~17.72ms
~27.8ms
~18ms
~34.17ms
A12
Top Performance Limiter
GPU Counter
~90%
~93%
~92%
~97%
~95%
~100%
A12
Total Occupancy
GPU Counter
~99%
~96%
~97%
~92%
~97%
~99%
A12
Compute Occupancy
GPU Counter
~98%
~96%
~97%
~92%
~96%
~98%
32x32
30x30
24x24
9x9
8x8
1x1

A11 and TBDR

A11 and TBDR

GPU Family 4

- Asynchronous Vertex and Fragment passes
- Faster access to tile’s memory
- Storing memoryless buffers in tile’s memory

WWDC + Example

Tips & Tricks

Cache

everything

Swift vs C

Swift vs C

- SIMD
- pointers
- determinacy

Structures

padding

```
typedef struct {
   simd_float4 rect;
   bool flag;
   short index;
   simd_float3 size;
} StructureBadPadding;

//...

sizeof(StructureBadPaddin
```

```
typedef struct {
   simd_float4 rect;
   bool flag;
   short index;
   simd_float3 size;
   float f;
} StructureBadPadding;

//...

sizeof(StructureBadPadding); // 64
```

Structures

padding

```
typedef struct {
   simd_float4 rect;
   bool flag;
// +1
   short index;
// +12
   simd_float3 size;
   float f;
// +12
} StructureBadPadding;
```

```
StructureBadPadding: 64
   rect  : 0  (16)
   flag  : 16 ( 1)
   index : 18 ( 2)
   size  : 32 (16)
   f     : 48 ( 4)
```

Structures

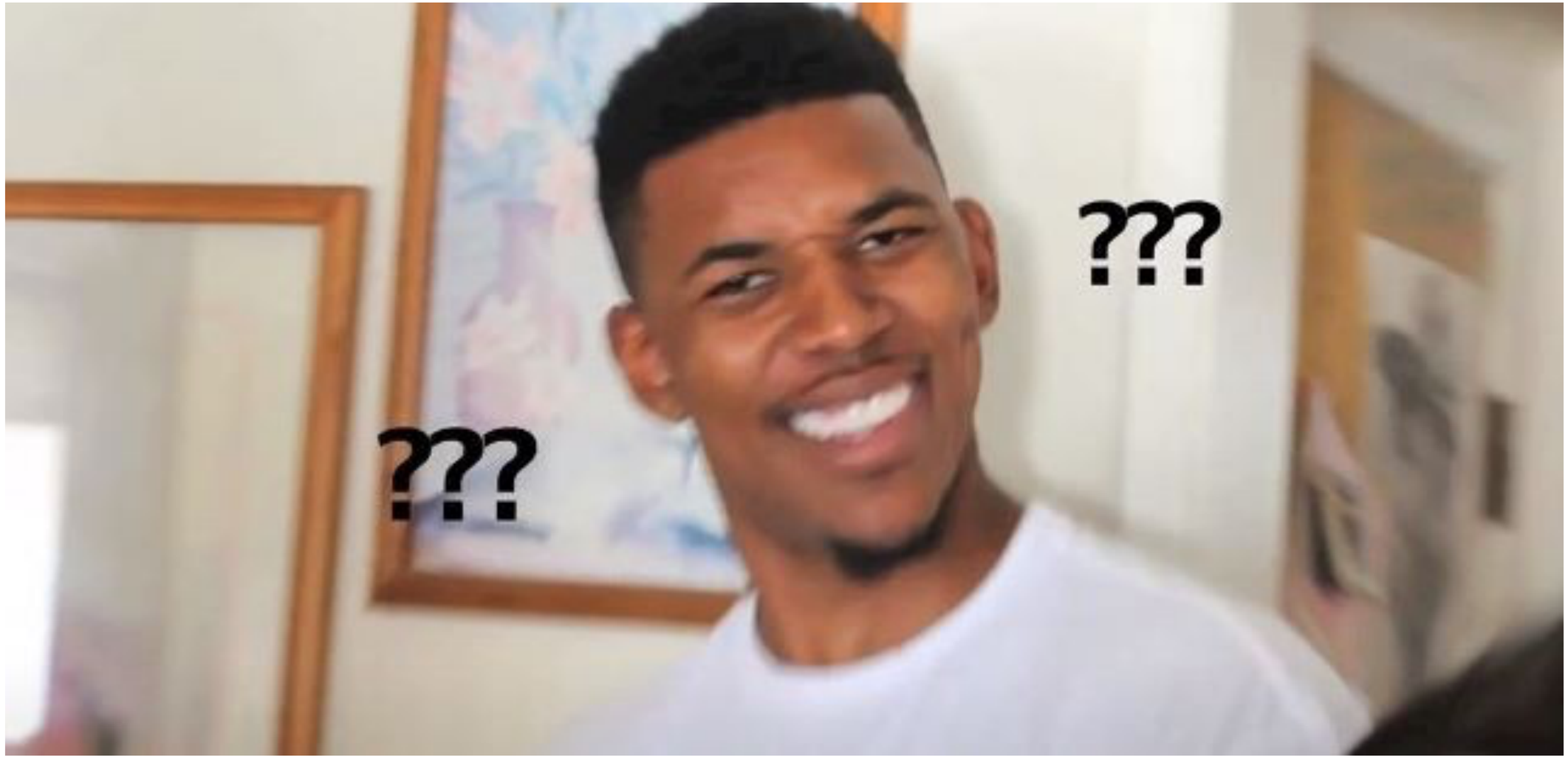
padding

```
typedef struct {
   simd_float4 rect;
   simd_float3 size;
   short index;
   bool flag;
} StructureGoodPadding;

//...

sizeof(StructureGoodPaddi
```

```
typedef struct {
   simd_float4 rect;
   simd_float3 size;
   float f;
   short index;
   bool flag;
} StructureGoodPadding;

//...

sizeof(StructureGoodPadding); // 48
```

Structures

padding



Structures

padding

```
typedef struct {
  simd_float4 rect;
  simd_float3 size;
  float f;
  short index;
  bool flag;
// +9
} StructureGoodPadding;
```

```
StructureGoodPadding: 48
   rect  : 0  (16)
   size  : 16 (16)
   f     : 32 ( 4)
   index : 36 ( 2)
   flag  : 38 ( 1)
```

Constants

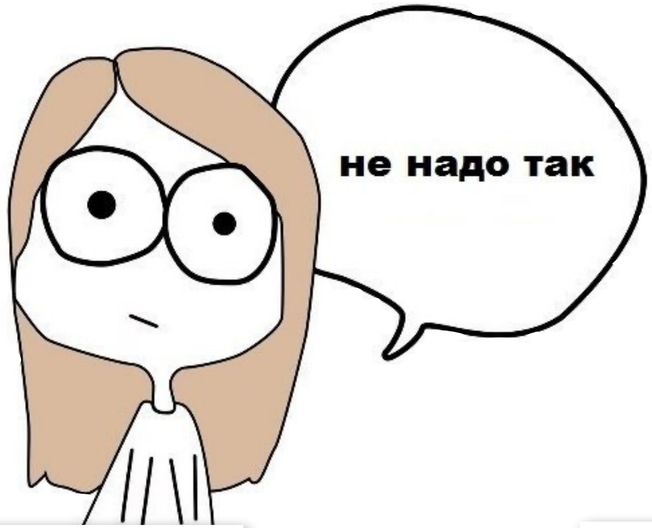
```
1      // int
1.0    // double
1.0f   // float
```

Constants

```
float foo(float a, float b) {
   return 10.0f * a * a + 5.0f * b * b;
}

foo(float, float):
        push    {r4, r5, r11, lr}
        mov     r4, r1
        mov     r1, r0
        bl      __aeabi_fmul
        mov     r1, #18874368
        orr     r1, r1, #1073741824
        bl      __aeabi_fmul
        mov     r5, r0
        mov     r0, r4
        mov     r1, r4
        bl      __aeabi_fmul
        mov     r1, #10485760
        orr     r1, r1, #1073741824
        bl      __aeabi_fmul
        mov     r1, r5
        bl      __aeabi_fadd
        pop     {r4, r5, r11, lr}
        bx      lr
```

```
float foo(float a, float b) {
   return 10.0 * a * a + 5.0 * b * b;
}

foo(float, float):
        push    {r4, r5, r6, lr}
        mov     r4, r1
        bl      __aeabi_f2d
        mov     r2, r0
        mov     r3, r1
        bl      __aeabi_dmul
        mov     r3, #2359296
        mov     r2, #0
        orr     r3, r3, #1073741824
        bl      __aeabi_dmul
        mov     r5, r0
        mov     r0, r4
        mov     r6, r1
        bl      __aeabi_f2d
        mov     r2, r0
        mov     r3, r1
        bl      __aeabi_dmul
        mov     r3, #1310720
        mov     r2, #0
        orr     r3, r3, #1073741824
        bl      __aeabi_dmul
        mov     r2, r5
        mov     r3, r6
        bl      __aeabi_dadd
        bl      __aeabi_d2f
        pop     {r4, r5, r6, lr}
        bx      lr
```

Constants




```
float foo(float a, float b) {
   return 10.0 * a * a + 5.0 * b * b;
}
```

```
float foo(float a, float b) {
   return 10.0f * a * a + 5.0f * b * b;
}
```

Links

- **Optimizing Performance with the Shader Profiler**
- **Metal System Trace**
- Metal Best Practices
- Tile-Based Deferred Rendering (TBDR)
- **WWDC Videos**

- Demo app

Contacts

wdfeffects@gmail.com

@GOstrobrod

@GOstrobrod

https://wdfteam.gitlab.io