Blend2D

High-Performance 2D Vector Graphics

Petr Kobalíček

About Me

- **20+ years of experience in writing C++ & assembly**
  - Self-employed; not representing any company
- **Focus on writing highly optimized software:**
  - SIMD optimized algorithms (AVX2, AVX-512)
  - Distributed solutions, multi-threading, and JIT
- **Recent projects:**
  - AsmJit & Blend2D
  - Sneller – AVX-512 optimized big-data query engine

Agenda

1. Background

- Software-Based 2D
- Existing Solutions

2. Blend2D

- Introduction
- API & Features

3. Performance

- Optimizations Implemented
- Performance Comparison

4. Epilogue

- Future Plans
- Discussion / QA

What is Software-Based 2D?

- **The rendering pipeline is implemented in software**
  - Rendering into pixel buffers
- **CPU is used for everything:**
  - Clipping
  - Stroking
  - Transformation
  - Rasterization
  - Style fetches
  - Composition

The Use of Software-Based 2D Rendering

- **Software-based 2D is used extensively in existing software**
  - It doesn't require additional hardware (GPU) to run
  - Drawing custom UI into 32-bit ARGB pixel buffers is common
  - **AGG** library is one of the most deployed libraries in this case
  - **Qt** Widgets are rendered by QPainter, which is software-based (Qt4+)

- **Today's research focuses mostly on hardware accelerated 2D**
  - Causes a stagnation of improving software-based 2D rendering
  - Transition to HW rendering of large code-bases in many cases non-trivial
  - Presents a unique opportunity for the Blend2D project

Software-based 2D rendering is not a bonus

It’s a safe path

Overview of Existing Open-Source 2D Libraries

- **The most used open-source libraries providing SW-based rendering:**

  AGG (C++) – Not actively developed
  Cairo (C) – Not actively developed
  Qt (C++) – QPainter's performance not improving
  Skia (C++) – Actively developed by Google for their products

- **None of them has been designed from scratch to fully take advantage of today's CPU capabilities**

What about a new 2D rendering engine which offers high-performance SW acceleration?

Blend2D – 2D Vector Graphics Engine

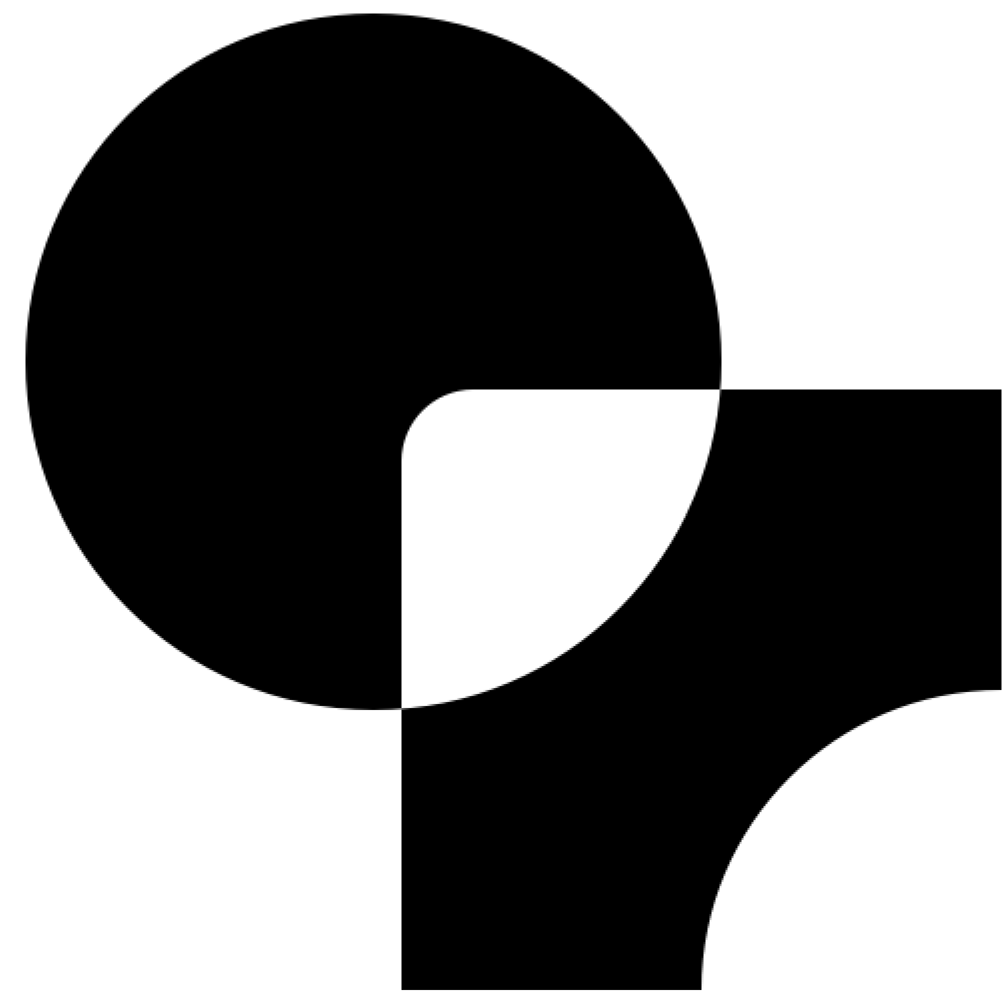
- **Designed from scratch for high performance**
- **Started as an experiment:**
  - Using AsmJit library to generate optimized 2D pipelines
  - Initially just for fun
  - Developed independently
- **Evolved into a 2D rendering engine:**
  - 2D rendering context that renders to a pixel buffer
  - Released under the Zlib license

Design Goals

- **Written in C++, but exports only C API**
  - C API can be used from most programming languages
  - C++ API consists of lightweight inline wrappers around C API
  - No exceptions & no dependency on C++ standard library
- **One optional third-party dependency – AsmJit**
- **Easy to build & integrate (CMake)**

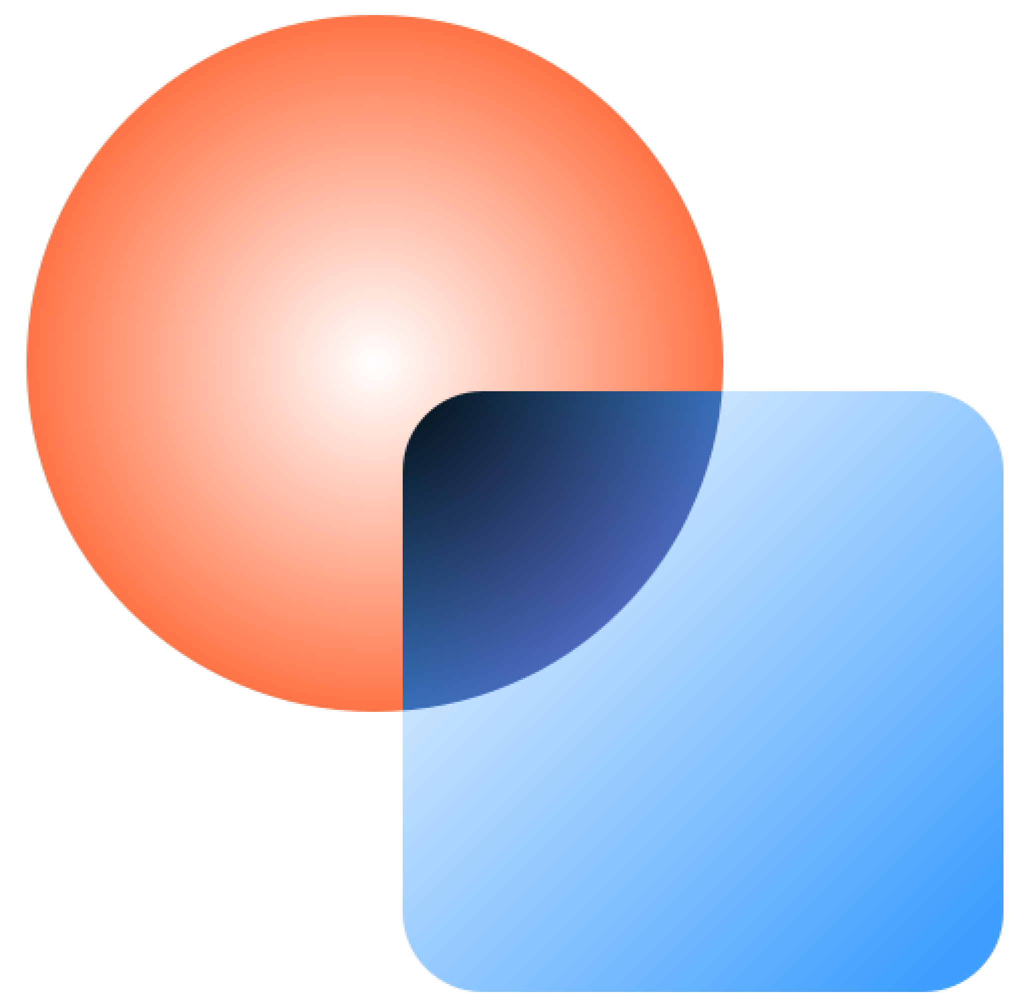
C++ API Example #1

```cpp
int main() {
  BLImage img(512, 512, BL_FORMAT_PRGB32);
  BLContext ctx(img);

  BLPath p;
  p.moveTo(494, 194);
    p.lineTo(494, 344);
    p.arcQuadrantTo(344, 344, 344, 494);
    p.lineTo(194, 494);
    p.lineTo(194, 230);
    p.arcQuadrantTo(194, 194, 230, 194);
  p.close();
  p.addCircle(BLCircle(180, 180, 174), BL_GEOMETRY_DIRECTION_CCW);

  ctx.fillAll(BLRgba32(0xFFFFFFFF));
  ctx.fillPath(p, BLRgba32(0xFF000000));
  ctx.end();

  img.writeToFile("example1.png");
  return 0;
}
```



C++ API Example #2

```cpp
int main() {
  BLImage img(512, 512, BL_FORMAT_PRGB32);
  BLContext ctx(img);

  BLGradient radial(BLRadialGradientValues(180, 180, 180, 180, 180));
  radial.addStop(0.0, BLRgba32(0xFFFFFFFF));
  radial.addStop(1.0, BLRgba32(0xFFFF6F3F));

  BLGradient linear(BLLinearGradientValues(194, 194, 470, 470));
  linear.addStop(0.0, BLRgba32(0xFFFFFFFF));
  linear.addStop(1.0, BLRgba32(0xFF3F9FFF));

  ctx.clearAll();
  ctx.fillCircle(BLCircle(180, 180, 174), radial);
  ctx.setCompOp(BL_COMP_OP_DIFFERENCE);
  ctx.fillRoundRect(BLRoundRect(194, 194, 300, 300, 40), linear);
  ctx.end();

  img.writeToFile("example2.png");
  return 0;
}
```



More Examples & Docs

- **Documentation**
  - blend2d.com

- **Interactive**
  - fiddle.blend2d.com

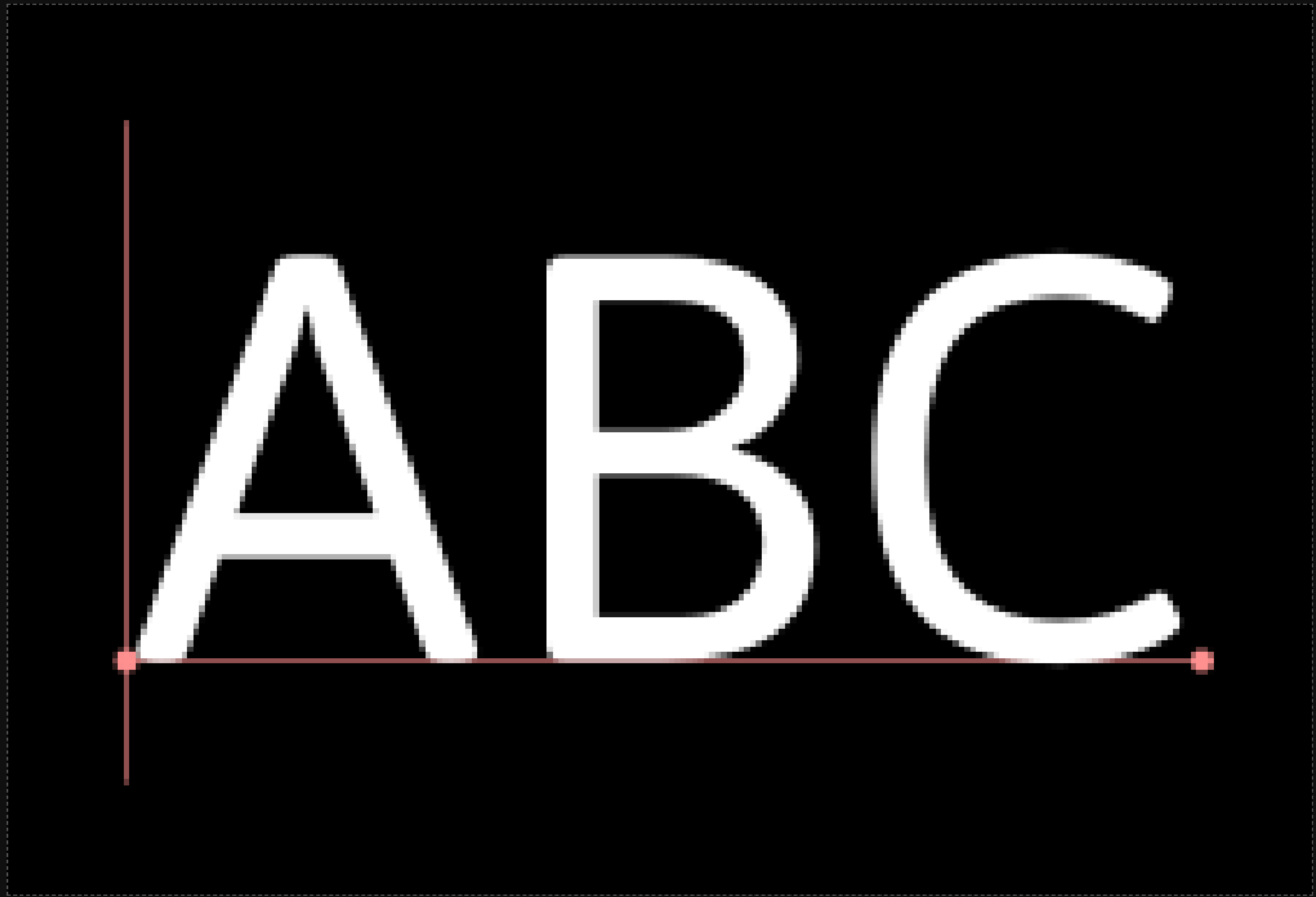
    (please don’t overload the server :})

Blend2D BETA
2D Vector Graphics Engine

Fiddle: <User-Fiddle> Profile: Release Tools: <No Tool> Run

```
BLImage render(const BLContextCreateInfo& cci) {
  BLFontFace face;
  face.createFromFile("fonts/Asap-Regular.ttf");

  BLFont font;
  font.createFromFace(face, 100);

  const char* text = "ABC";

  BLFontMetrics fm = font.metrics();
  BLTextMetrics tm;
  BLGlyphBuffer gb;

  gb.setUtf8Text(text);
  font.shape(gb);
  font.getTextMetrics(gb, tm);

  BLImage img(int(tm.advance.x + 40), int(fm.ascent + fm.descent + 40), BL_FORMAT_PRGB32);
  BLContext ctx(img, cci);

  BLPoint pos(20.5, floor(fm.ascent) + 20.5);

  ctx.clearAll();
  ctx.setStrokeStyle(BLRgba32(0xFF905050));
  ctx.strokeLine(pos.x, pos.y - fm.ascent, pos.x, pos.y + fm.descent);
  ctx.strokeLine(pos.x, pos.y, pos.x + tm.advance.x, pos.y);

  ctx.setFillStyle(BLRgba32(0xFFFFFFFF));
  ctx.fillUtf8Text(pos, font, text);

  ctx.setFillStyle(BLRgba32(0xFFFF9090));
  ctx.fillCircle(pos.x, pos.y, 2);
  ctx.fillCircle(pos.x + tm.advance.x, pos.y, 2);

  return img;
}
```

+ - Zoom=3x Size={225x154}



Compilation time: 95.6ms | Execution time: 9.7ms


Bounces Sample [1090x620] [209 objects] [AvgTime=0.87ms FPS=125.0]
Renderer: Blend2D
Limit FPS
Style: Linear Gradient
Sprites Sample [1090x628] [Count=169] [AvgTime=0.60ms FPS=125.0]
Renderer: Blend2D
Comp Op: SrcOver
Limit FPS
Count:
Tiger Sample [1090x628] [AvgTime=2.20ms FPS=125.1]
Renderer: Blend2D
Caching: None
Limit FPS
Zoom:
Stroke Sample
Stroke Caps: Butt
Random
A
B
C
D
E
F
Stroke Join: Miter-Clip
Dump
X
Y
Z
Width:
Miter Limit:

Blend2D Optimizations Overview

- **JIT pipeline generation**
- **Analytic rasterization improvements**
- **Multi-threaded rendering**

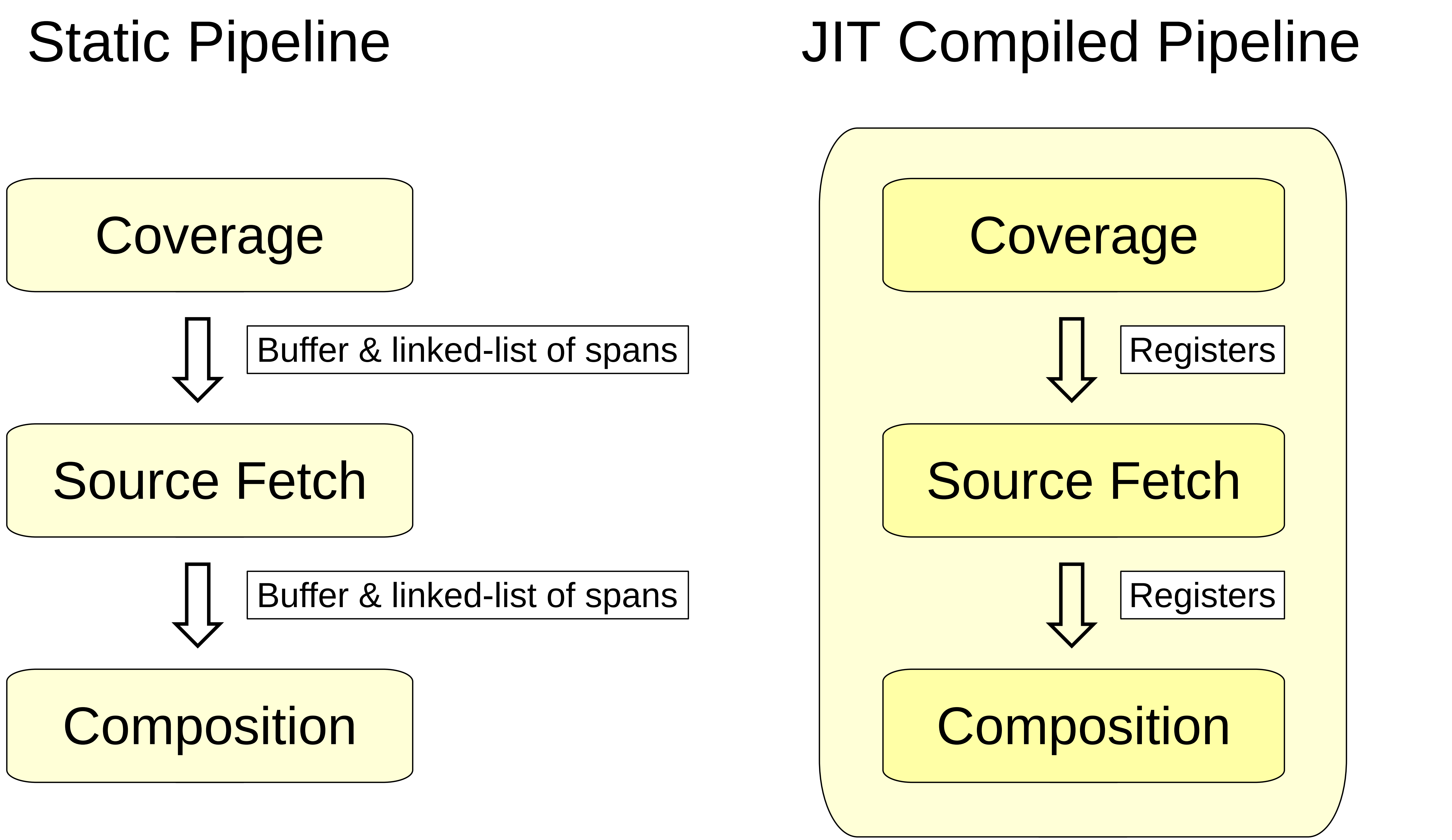
JIT Pipeline Generation

- **Compilation of 2D pipelines at runtime**
  - Optimized for the host CPU
  - Inlined 3 stages of a pipeline:
    - **Coverage** stage
    - **Style Fetch** stage
    - **Composition** stage
  - Data between stages is transferred via CPU registers
  - Processing pixels at bands => Multiple scanlines per call

JIT Pipeline Generation

- **Requirements**
  - The generated pipeline should be faster than an equivalent written in C++
  - A single pipeline must be generated in a sub-millisecond time
- **Solution**
  - Join multiple pre-implemented parts written in assembly
  - Use AsmJit to perform register allocation and machine code generation
- **Take advantage of host CPU extensions**
  - Use general purpose extensions (BMI, BMI2, ...) when available
  - Take advantage of SIMD levels – SSE2+, AVX2+FMA, AVX512+
  - What about “machine learning” extensions such as AVX512_VNNI?

Comparison of Static and JIT Pipelines



Analytic Rasterization

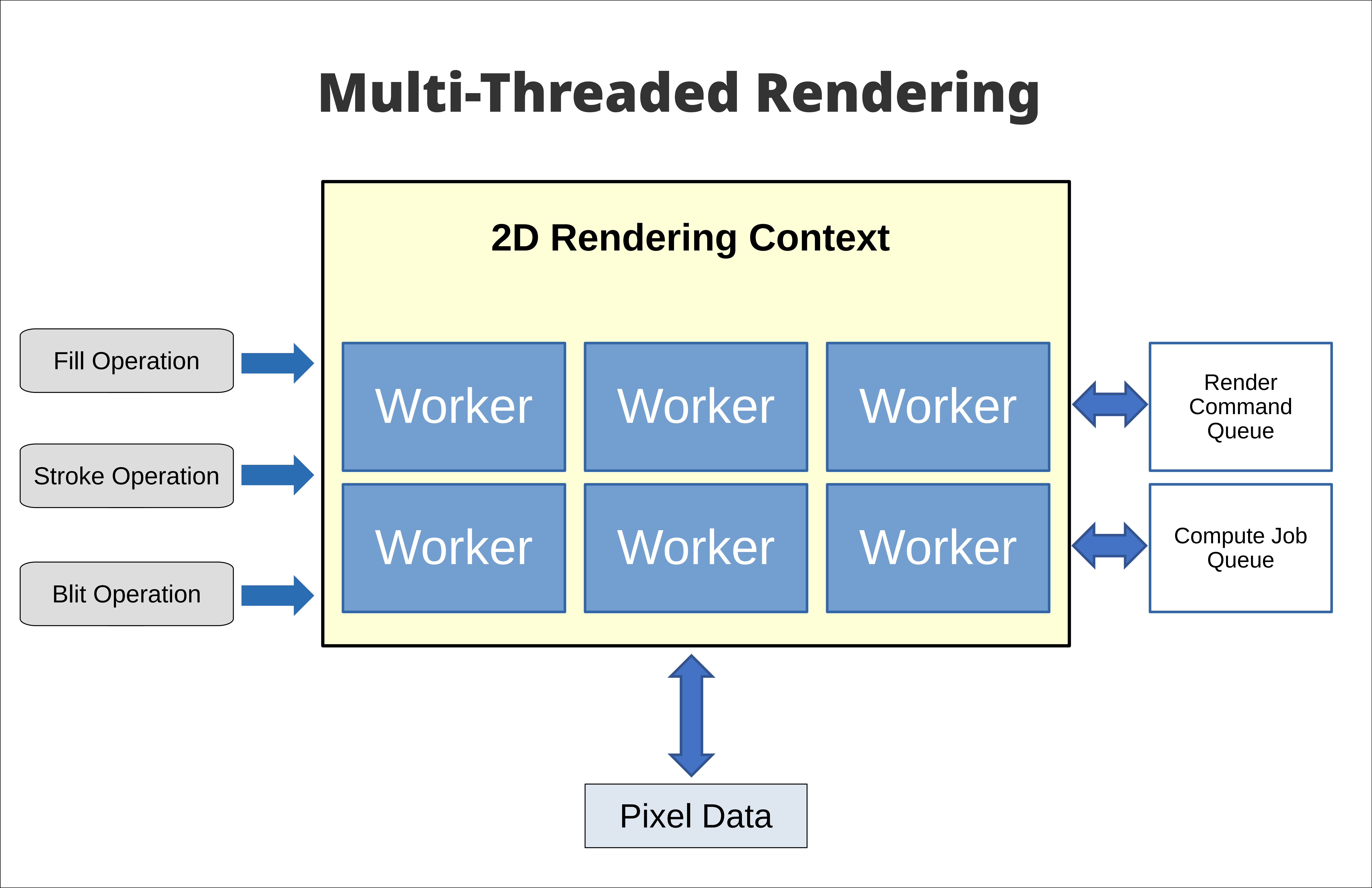
Analytic Rasterization Basics

- **Rasterization**
  - The rasterizer iterates over edges and accumulates area and cover for each pixel it intersects
  - Area and cover form a cell; each associated with a pixel at [x, y]
  - Calculating the final pixel coverage:
    - Cover values are accumulated, area values are used independently
    - Alpha = calculate_coverage(sum(cover[0:x]) - area[x])
- **Association with cells and pixels is important:**
  - Traditionally, linked-list (Qt) or a vector of cells (AGG) is used
  - Not really efficient when a lot of edges cross a single scanline

Analytic Rasterization Improvements

- **Dense cell-buffer:**
  - Each pixel has a pre-allocated cell
  - Cells are consecutive => no need to store cover and area values separately
  - By only using a single value per cell, the storage has been reduced by 50%

- **Shadow bit-buffer**
  - In order to maintain which cells are non-zero, a shadow bit-buffer has been introduced
  - Each bit describes 4 cells, thus a single 64-bit value represents 256 cells / pixels
  - To quickly find the coordinate of non-zero cells, use trailing/leading bit-count


Multi-Threaded Rendering
2D Rendering Context
Fill Operation
Stroke Operation
Blit Operation
Worker
Worker
Worker
Worker
Worker
Worker
Render Command Queue
Compute Job Queue
Pixel Data

Multi-Threaded Rendering

- **Motivation**
  - Screens are getting wider => more pixels to render
  - Increasing the CPU frequency is not practical anymore
  - Modern CPUs have 8+ cores – a lot of computational power
- **Premise**
  - Offer a multi-threaded rendering context for complex workloads
  - It must use the same API as single-threaded renderer
  - It should not regress the performance even in border cases

Work Distribution

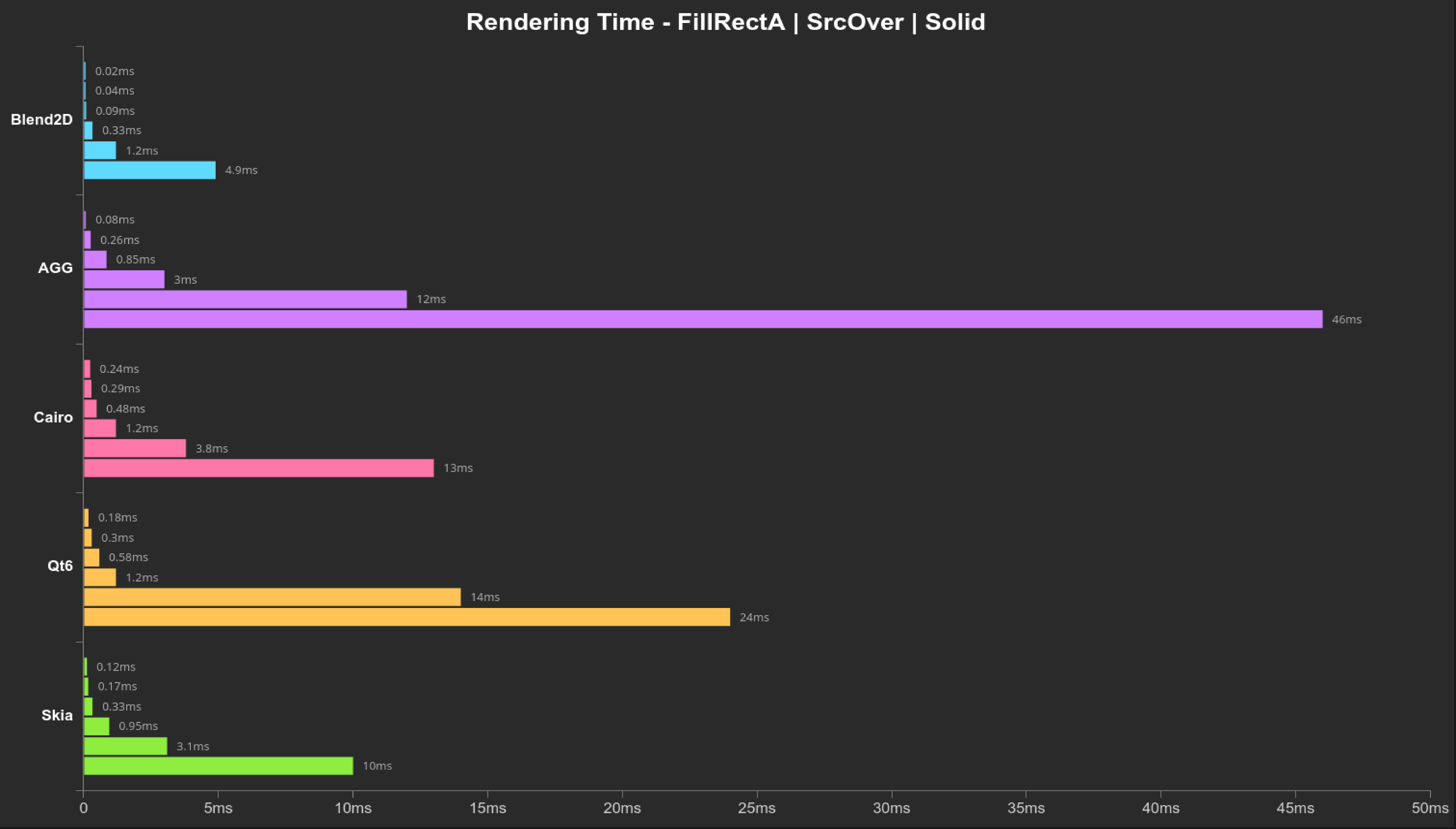
- **Work Separation**
  - Each worker thread has pre-assigned parts of the image it operates on
  - There are two types of work each thread does
    - **Render command** – changes pixels of the target image – invokes 2D pipelines
    - **Compute job** – work to do before a render command can be processed (edge building, stroking, font outline extraction, or a combination)

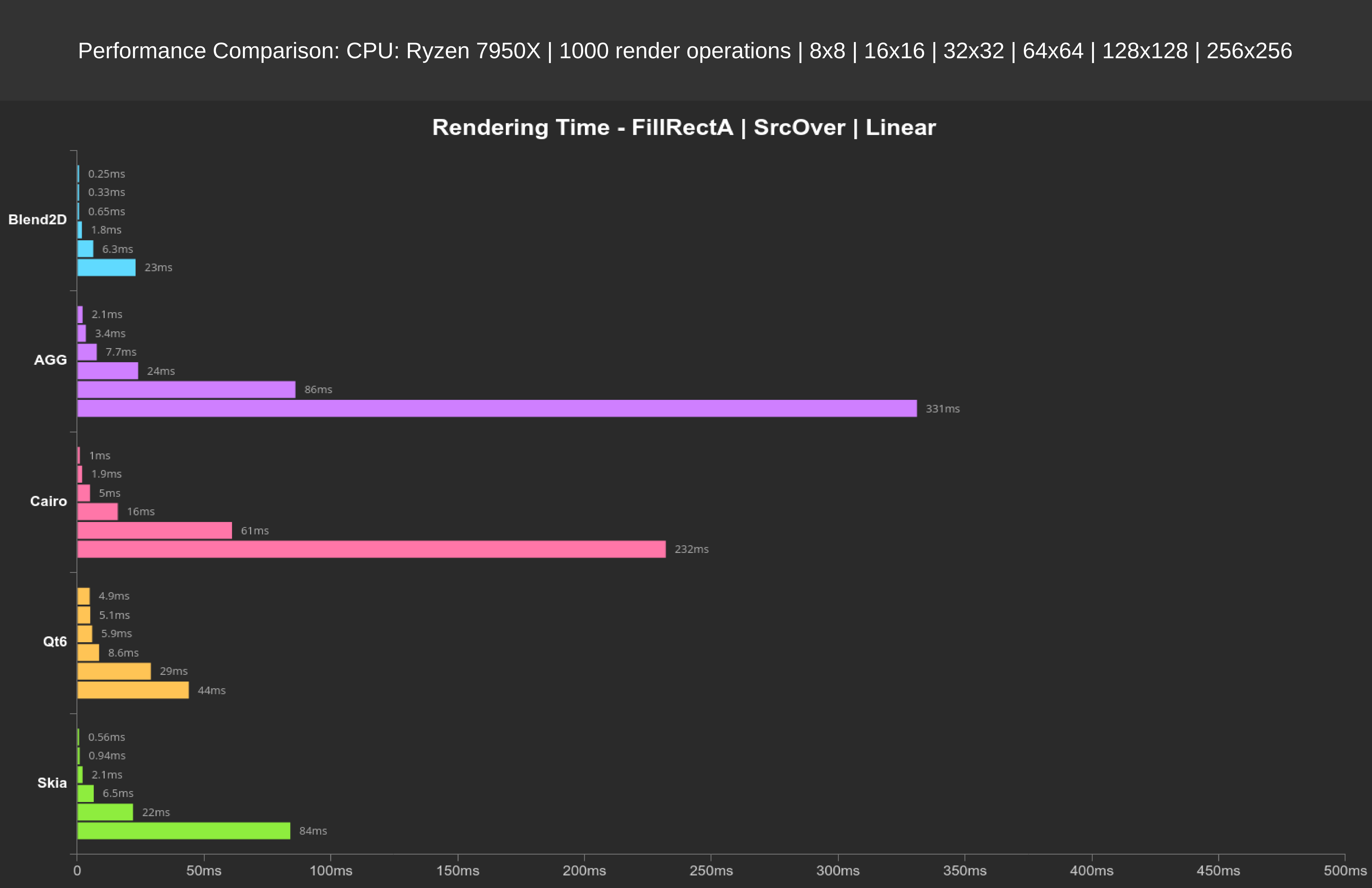
- **Work Serialization**
  - In MT rendering mode the frontend has to serialize instead of dispatch:
    - Ensure that shared states match the current rendering context state
    - If a render request needs a compute job, create it and add it to a job queue
    - Add a render command to the command queue

Performance Comparison

Explore at blend2d.com/performance.html

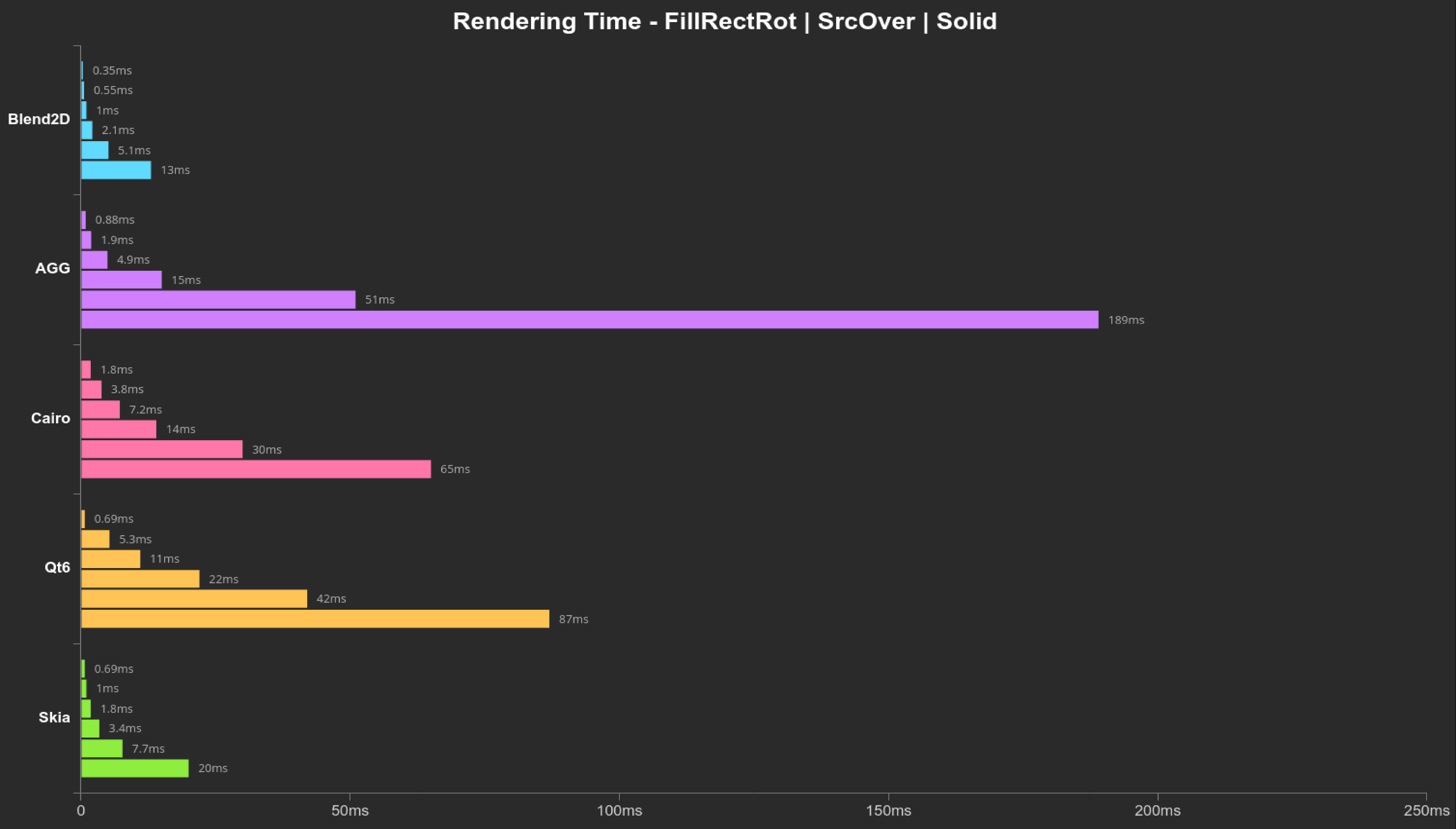
Performance Comparison: CPU: Ryzen 7950X | 1000 render operations | 8x8 | 16x16 | 32x32 | 64x64 | 128x128 | 256x256



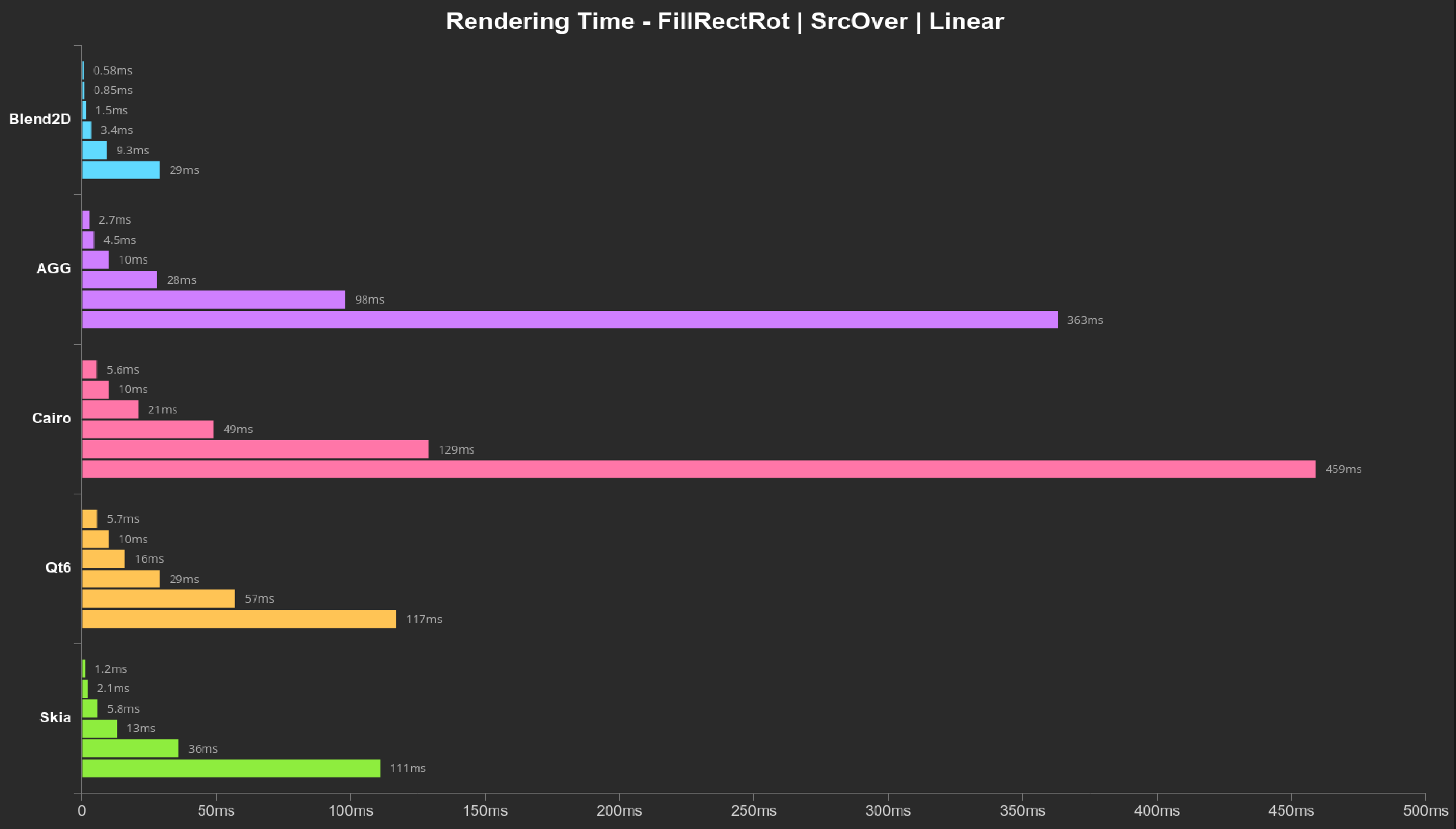


Performance Comparison: CPU: Ryzen 7950X | 1000 render operations | 8x8 | 16x16 | 32x32 | 64x64 | 128x128 | 256x256
Rendering Time - FillRectA | SrcOver | Linear
Blend2D
0.25ms
0.33ms
0.65ms
1.8ms
6.3ms
23ms
AGG
2.1ms
3.4ms
7.7ms
24ms
86ms
331ms
Cairo
1ms
1.9ms
5ms
16ms
61ms
232ms
Qt6
4.9ms
5.1ms
5.9ms
8.6ms
29ms
44ms
Skia
0.56ms
0.94ms
2.1ms
6.5ms
22ms
84ms
0
50ms
100ms
150ms
200ms
250ms
300ms
350ms
400ms
450ms
500ms

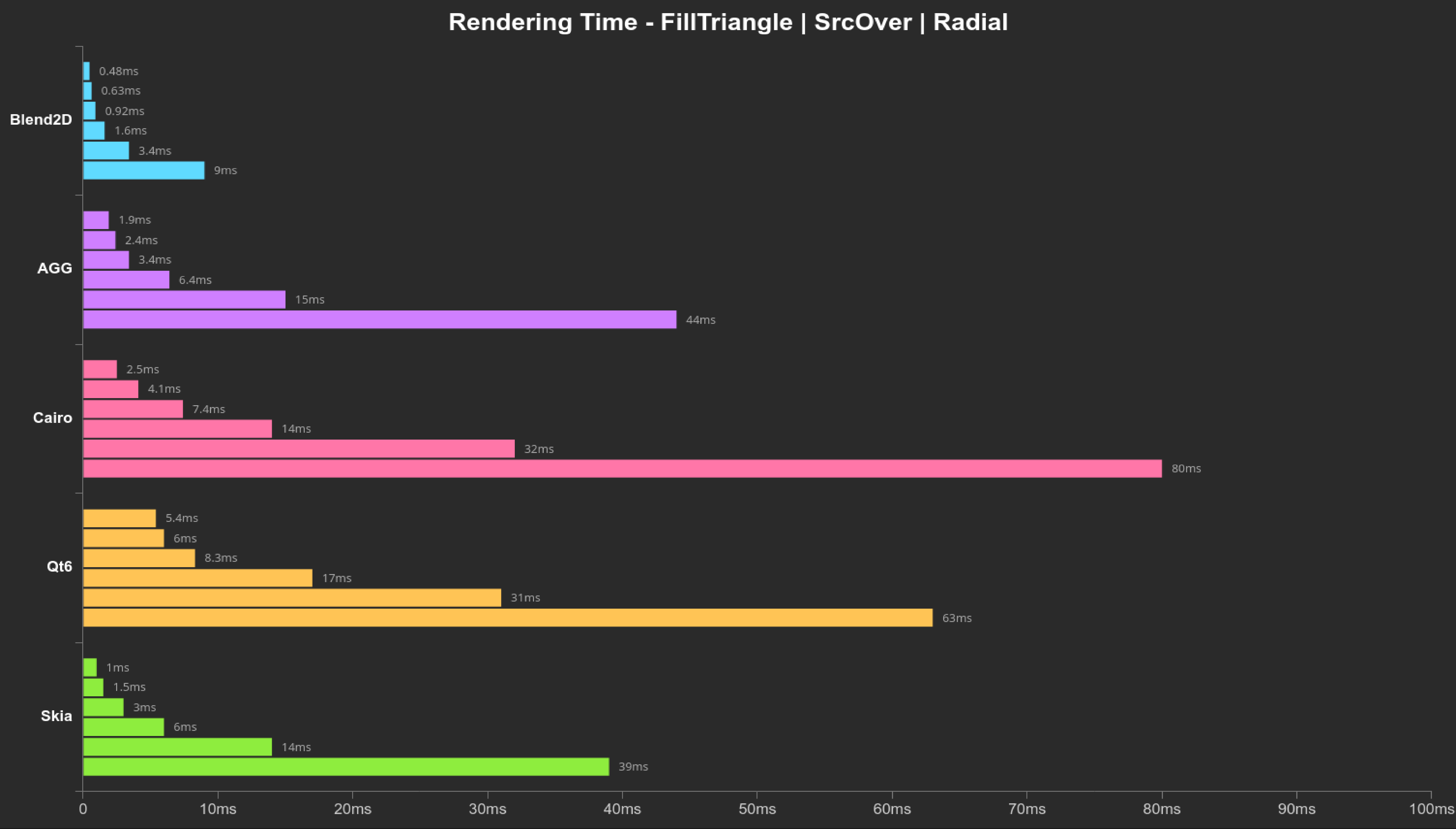
Performance Comparison: CPU: Ryzen 7950X | 1000 render operations | 8x8 | 16x16 | 32x32 | 64x64 | 128x128 | 256x256



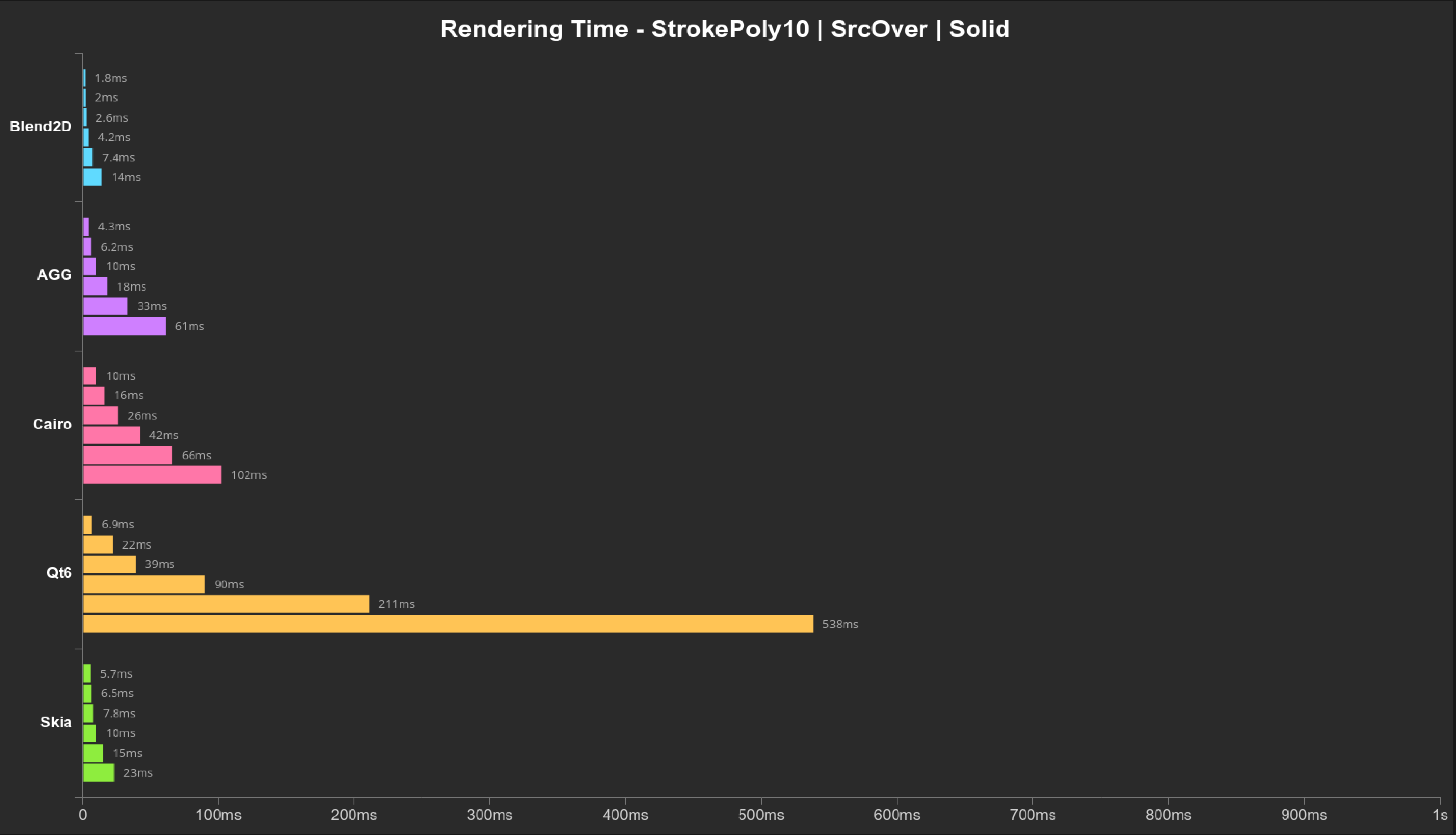
Performance Comparison: CPU: Ryzen 7950X | 1000 render operations | 8x8 | 16x16 | 32x32 | 64x64 | 128x128 | 256x256



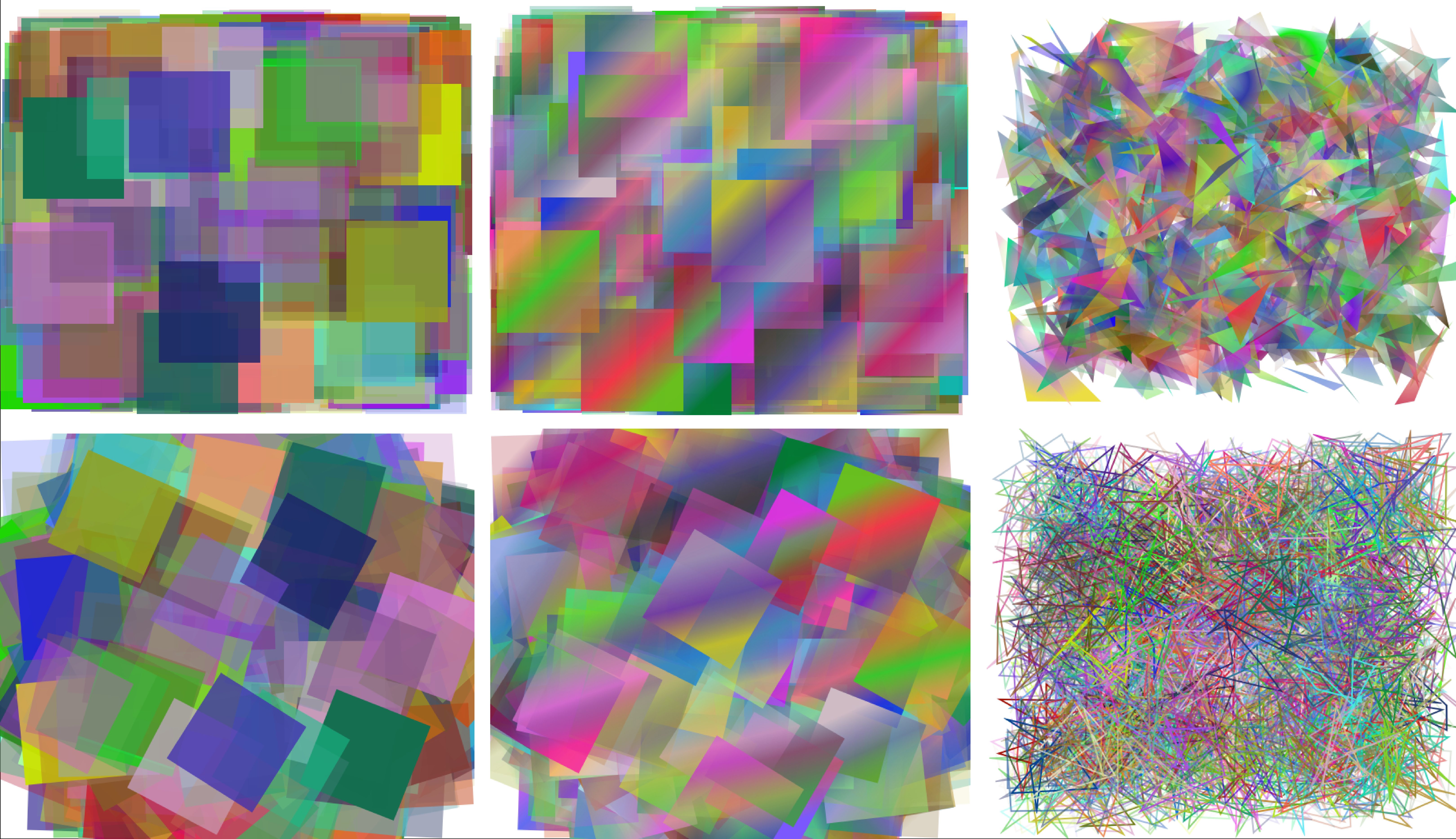
Performance Comparison: CPU: Ryzen 7950X | 1000 render operations | 8x8 | 16x16 | 32x32 | 64x64 | 128x128 | 256x256



Performance Comparison: CPU: Ryzen 7950X | 1000 render operations | Line Width = 3



Benchmarking Tool Output



Blend2D – Future Plans

- **2D Effects**
  - Geometry effects, pixel effects, convolution
  - Shading language – being discussed by the community
- **Optimizations**
  - Rasterizer can still be optimized (mostly scalar code)
  - Text rendering doesn't use caching at the moment
- **Functionality**
  - Non-rectangular clipping – a rasterized path or user-provided mask
  - Better text rendering – text shaping, full OpenType GSUB/GPOS support
- **GPU Acceleration**
  - Blend2D started as a library to offer software-based rendering first
  - But the rendering context is abstract and can offer GPU acceleration in the future

Thank You!

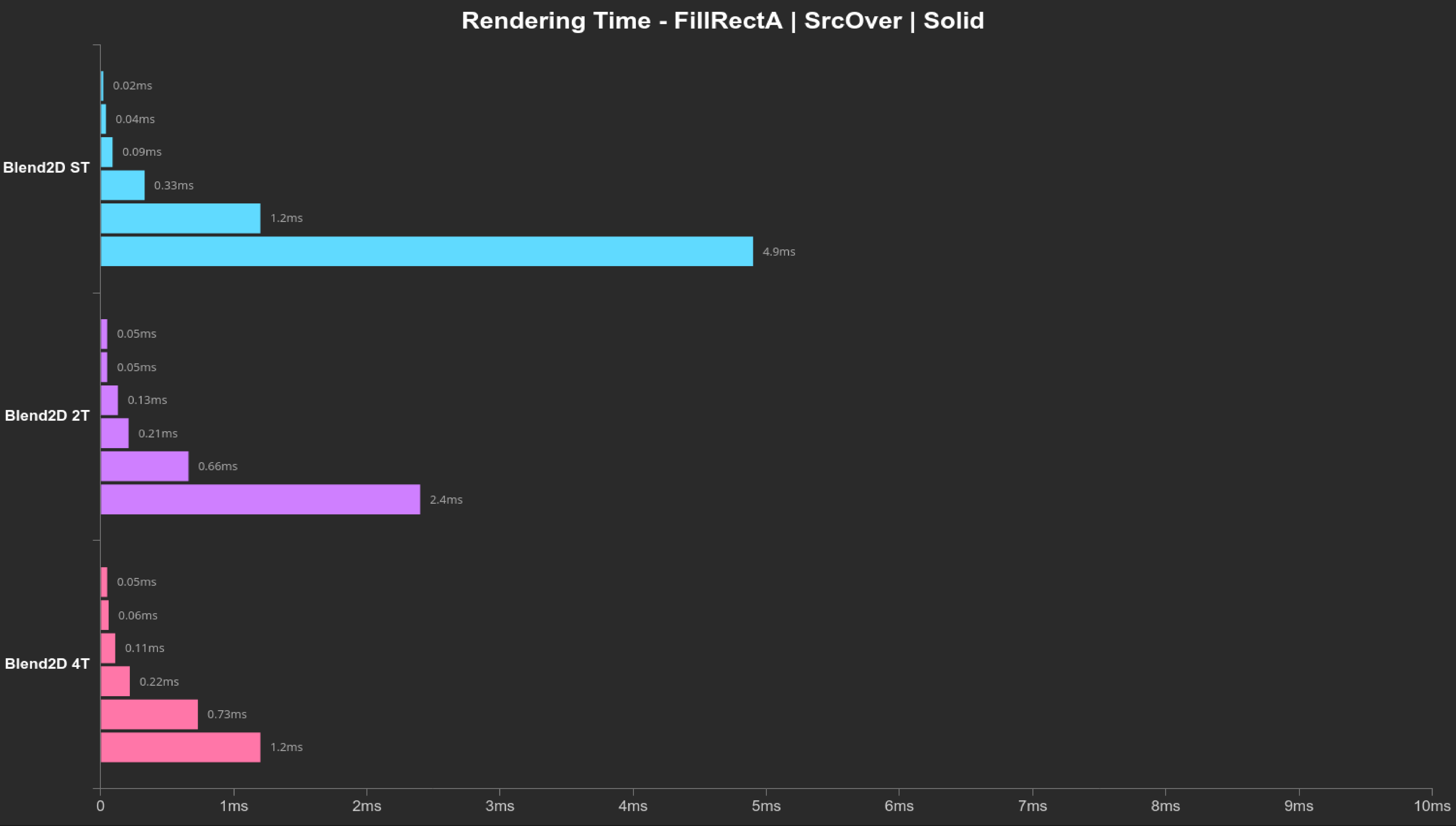
Time for Discussion & QA

Check out blend2d.com for more information

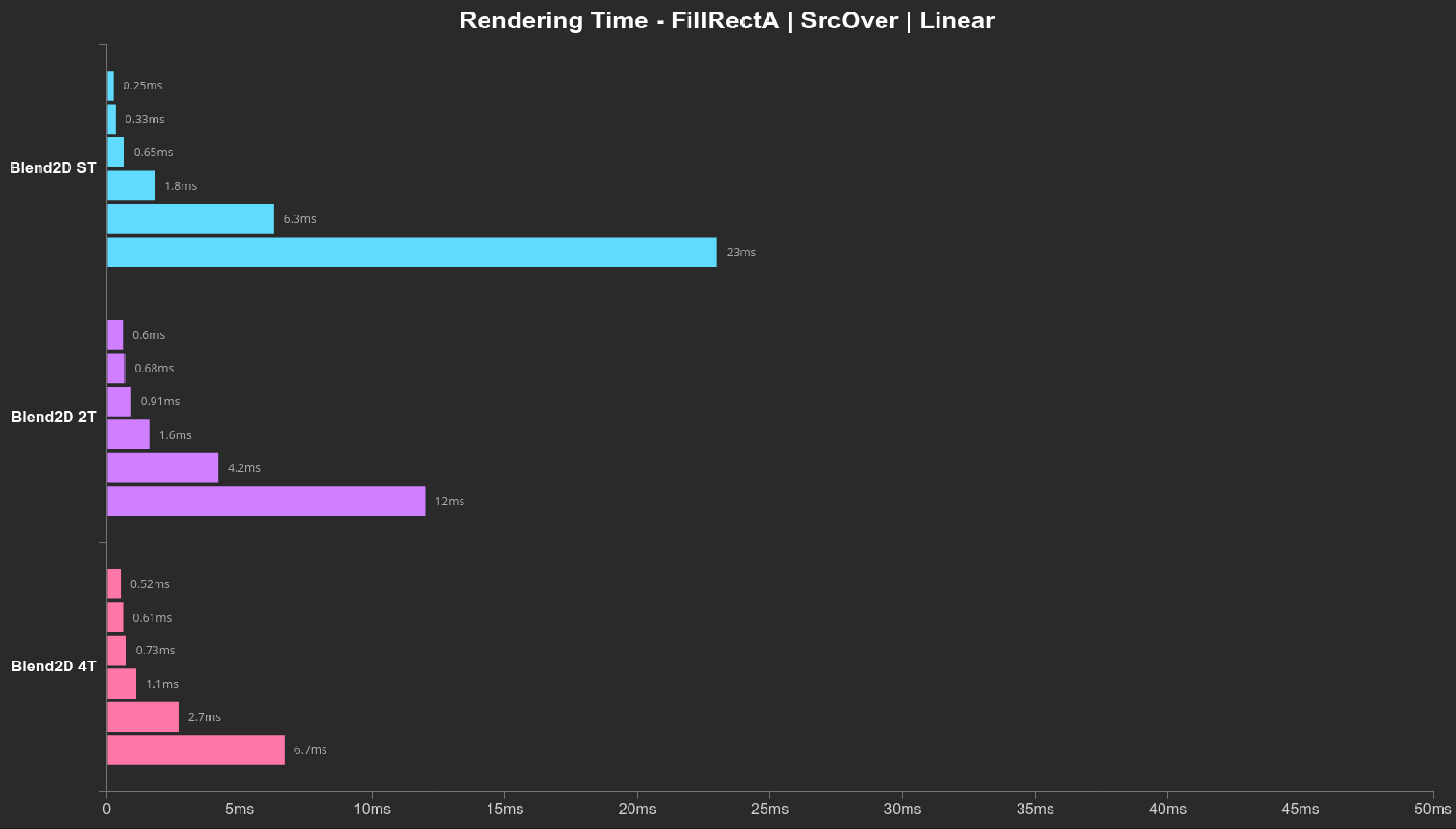
Appendix – Solid SrcOver Composition (SIMD)

```
SSE2: 8 pixels                | AVX2: 16 pixels                 | AVX512: 32 pixels
                              |                                 |
movaps xmm1, [rax]            | vpmovzxbw ymm3, [rax]           | vpmovzxbw zmm1, [rax]
movaps xmm2, [rax+16]         | vpmovzxbw ymm1, [rax+16]        | vpmovzxbw zmm0, [rax+32]
movaps xmm3, xmm1             | vpmovzxbw ymm2, [rax+32]        | vpmovzxbw zmm2, [rax+64]
punpckhbw xmm3, xmm7          | vpmovzxbw ymm0, [rax+48]        | vpmovzxbw zmm3, [rax+96]
punpcklbw xmm1, xmm7          | vpmullw ymm3, ymm3, ymm4        | vpmullw zmm1, zmm1, zmm4
movaps xmm0, xmm2             | vpmullw ymm1, ymm1, ymm4        | vpmullw zmm0, zmm0, zmm4
punpckhbw xmm0, xmm7          | vpmullw ymm2, ymm2, ymm4        | vpmullw zmm2, zmm2, zmm4
punpcklbw xmm2, xmm7          | vpmullw ymm0, ymm0, ymm4        | vpmullw zmm3, zmm3, zmm4
pmullw xmm1, xmm4             | vpaddw ymm3, ymm3, ymm5         | vpaddw zmm1, zmm1, zmm5
pmullw xmm3, xmm4             | vpaddw ymm1, ymm1, ymm5         | vpaddw zmm0, zmm0, zmm5
pmullw xmm2, xmm4             | vpaddw ymm2, ymm2, ymm5         | vpaddw zmm2, zmm2, zmm5
pmullw xmm0, xmm4             | vpaddw ymm0, ymm0, ymm5         | vpaddw zmm3, zmm3, zmm5
paddw xmm1, xmm5              | vpmulhuw ymm3, ymm3, ymm6       | vpmulhuw zmm1, zmm1, zmm6
paddw xmm3, xmm5              | vpmulhuw ymm1, ymm1, ymm6       | vpmulhuw zmm0, zmm0, zmm6
paddw xmm2, xmm5              | vpmulhuw ymm2, ymm2, ymm6       | vpmulhuw zmm2, zmm2, zmm6
paddw xmm0, xmm5              | vpmulhuw ymm0, ymm0, ymm6       | vpmulhuw zmm3, zmm3, zmm6
pmulhuw xmm1, xmm6            | vpackuswb ymm1, ymm3, ymm1      | vpmovwb [rax], zmm1
pmulhuw xmm3, xmm6            | vpackuswb ymm0, ymm2, ymm0      | vpmovwb [rax+32], zmm0
pmulhuw xmm2, xmm6            | vpermq ymm1, ymm1, 0xD8         | vpmovwb [rax+64], zmm2
pmulhuw xmm0, xmm6            | vpermq ymm0, ymm0, 0xD8         | vpmovwb [rax+96], zmm3
packuswb xmm1, xmm3           | vmovdqu [rax], ymm1             |
packuswb xmm2, xmm0           | vmovdqu [rax+32], ymm0          |
movaps [rax], xmm1            |                                 |
movaps [rax+16], xmm2         |                                 |
```

Appendix – Comparison of ST and MT Rendering (1000 commands)



Appendix – Comparison of ST and MT Rendering (1000 commands)



Appendix – Comparison of ST and MT Rendering (1000 commands)