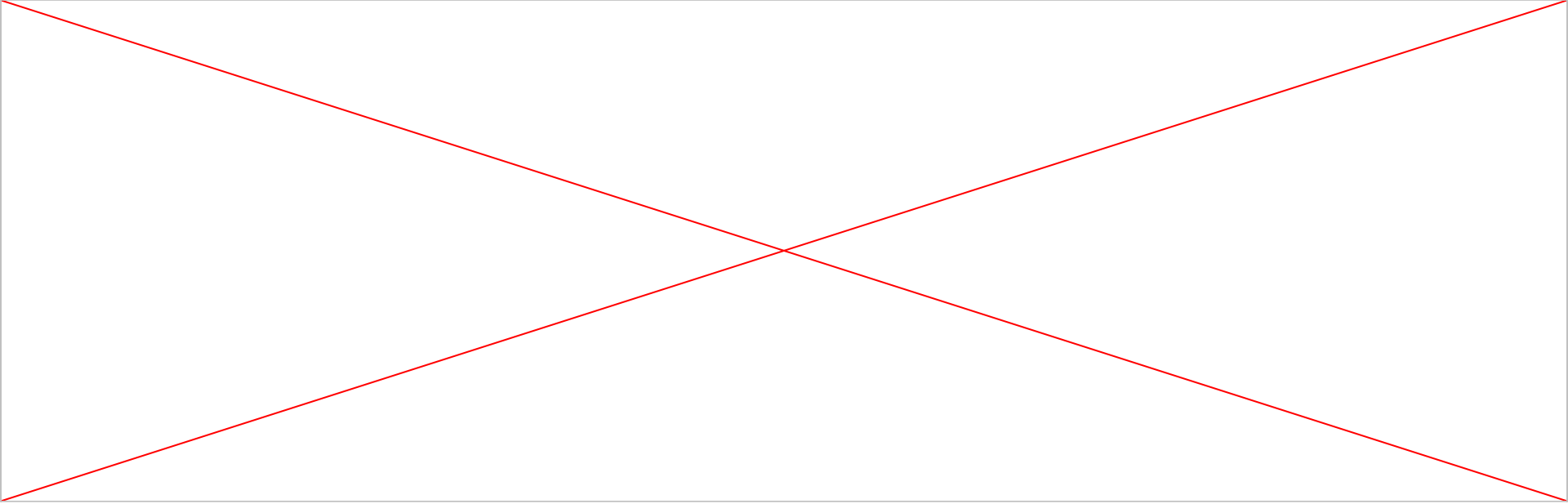
Halide

Industrial experience, design retrospective, and future directions

User-Schedulable Languages Workshop 2025
Rotterdam, Netherlands

Alex Reinking

Email: areinking@adobe.com
GitHub: @alexreinking
Website: alexreinking.com



```
input_16(x, y) = cast<uint16_t>(
  repeat_edge(input)(x, y));

blur_x(x, y) = (input_16(x-1, y) +
                input_16(x, y) +
                input_16(x+1, y))/3;

result(x, y) = cast<uint8_t>(
  (blur_x(x, y-1) +
   blur_x(x, y) +
   blur_x(x, y+1))/3);
```

```
result.compute_root()
      .tile(x, y, xi, yi, 128, 24)
      .parallel(y)
      .vectorize(xi, 32);

blur_x.compute_at(result, yi)
      .store_at(result, x)
      .vectorize(x, 32);

input_16.compute_at(result, x)
        .vectorize(x, 32);
```

Halide in industry

- **Google**
  - Pixel phone cameras (2015+)
- **Qualcomm**
  - Ships in the official Hexagon SDK with some proprietary improvements to HVX codegen.
- **Adobe**
  - Implements 2500+ of Photoshop's performance critical kernels.
  - WebAssembly backend facilitated porting Photoshop to the web.
- **Apple**
  - Contributed support for Apple Silicon, and there are Halide symbols in the system partition

Halide is both supported and constrained by industrial adoption.

What made Halide successful?

Technical reasons

- **Good ideas and design**
  - The scheduling language is expressive and right for the domain
  - Bounds inference keeps scheduling expressive
  - Schedules promote portability, even to future hardware
- **Practically good results**
  - Demonstrated large performance gains in non-trivial early applications
  - Halide code is more maintainable than intrinsics-heavy C++ code.

What made Halide successful?

Social reasons

- **Skunk-works style adoption**
  - Get hired at a company and deploy it there!
- **Well-balanced growth**
  - We never got caught in a hype cycle.
- **Symbiotic relationships with products**
  - As an engineer, you're rewarded for making products better.
  - As a researcher, your work needs to be aligned with that incentive.
- **No direct competitors**
  - Halide wasn't trying to be Yet Another Shader Language.
- **MIT Licensing**
  - Allows contributors to move between companies. Facilitates university collaboration.

What is holding Halide back?

- **Dependencies are a blessing and a curse**
  - We spend a lot of cycles tracking rapid changes to LLVM's API.
  - I have personally spent a lot of time fixing our dependencies' build systems.
- **Downstream inertia**
  - Even if we find better schedules, projects with enough resources will often rewrite their code to follow them rather than adopt a new tool.
- **Resources and headcount**
  - Manual maintenance burden: infrastructure, community support, onboarding
  - No formal managing entity: limits funding, avoids entanglement
  - Need more hands to implement new hardware backends.
- **Steep learning curve**
  - Halide's programming model is still very unfamiliar, not taught in colleges.
- **Missing hardware specifications**

Design & implementation: what worked?

- **Cross-compiling**
  - Generating object files rather than source code eliminates toolchain differences.
- **Term-rewriting for simplification/solving**
  - SMT solvers are powerful but slow. Need fast generated code *and* fast compile times.
- **Fully replaceable runtime**
  - Users can bring their own thread pools, GPU memory managers, command buffers, etc.
- **Embedding in C++/Python**
  - We get a lot of tooling “for free” (e.g. code completion, syntax highlighting, formatting)
  - Easy meta programming from the host language
- **Leaning into standard tooling**
  - Halide builds with CMake and we provide a CMake package for downstreams.
  - Distribution on PyPI: **`pip install halide`**
  - Works for C++, too, thanks to manylinux!

Design & implementation: what didn’t?

- **Algorithms and schedules aren’t perfectly separate**
  - In practice, the performance engineer has to adjust the algorithm to allow better scheduling
- **Reductions are somewhat ad-hoc**
  - Update stages and reduction domains are confusing and sometimes require manual bounds computation from the input shapes.
- **Debugging is hard at all levels**
  - Error messages are limited because “line numbers” don’t really exist.
  - *Open problem:* correlating compiler outputs to their inputs.
- **Complicated build process**
  - **Build** the generator executable *(host toolchain)*
  - **(Cross-)compile** a pipeline to produce an object file (on the host)
  - **Link** the resulting object file to the target application *(target toolchain)*

Testing & Reliability — Engineering Practices

- **Unit tests are helpful during development, but rarely fail later**
- **Don’t just test valid programs — test failure modes too**
- **Fuzz testing is essential:**
  - Uncovers unexpected uses by definition
  - Biased toward small programs means good reproducers
  - Broad coverage reduces need for hand-written tests (but turn fuzz failures into fixed tests)
- **Make fuzz failures reproducible across platforms**
  - Prefer deterministic RNGs (e.g., mt19937) with logged seeds
- **Bad test infrastructure kills morale:**
  - Too many tests, slow tests, flaky tests = unhappy contributors

Testing & Reliability — Theoretical and Formal Challenges

- **Formal verification can eliminate entire classes of bugs**
  - But only if the spec is right
- **Verified components still benefit from fuzzing**
  - Fuzz finds issues outside the formal model
- **Real-world users write "golden" tests**
  - Often opaque, fragile, and based on legacy behavior
  - May compare output by checksum or float-equality
- **Floating point fragility is unavoidable**
  - Thresholding doesn’t always work
  - Deterministic mode helps, but isn't realistic

Future directions for Halide

- **ML inference driving future work**
  - Especially important on edge devices.
  - Torch Inductor has an experimental Halide backend
- **New language features:**
  - Caching (e.g. for KV caches)
  - Tiled storage
  - User-scheduled approximation
  - Code outlining / deduplication
  - Offline computation
- **New hardware backends:**
  - Tensor Cores
  - Custom accelerators