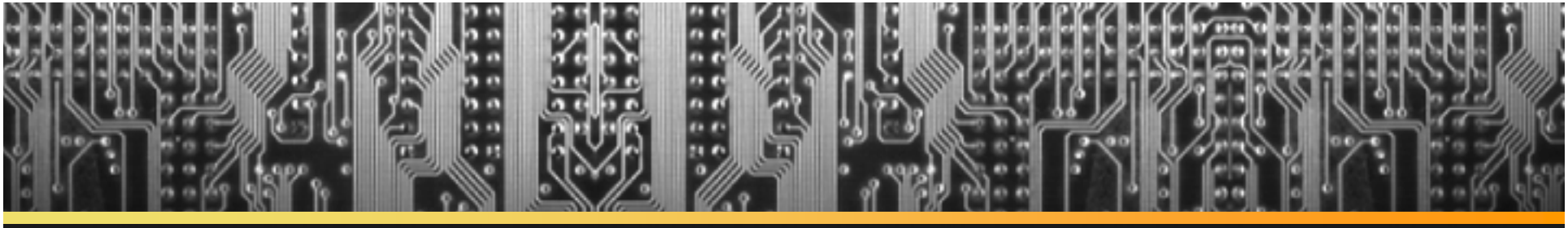


Tracy: A Debugger and System Analyzer for Cross-Platform Graphics Development

Sami Kyöstilä (Nokia)
Kari J. Kangas (Nokia)
Kari Pulli (Nokia Research Center)

Motivation

Mobile graphics development environment

- Cross-platform
- Cross-company
- Immature and evolving

Common tasks

- Error isolation
- Optimization
- Performance estimation

Overview

Tracy architecture

Related work

Workflow, common use-cases

Data-driven API configuration

- Code generation, state tracking

Trace compression

Trace analysis and transformation

Conclusion

Tracy architecture

Components

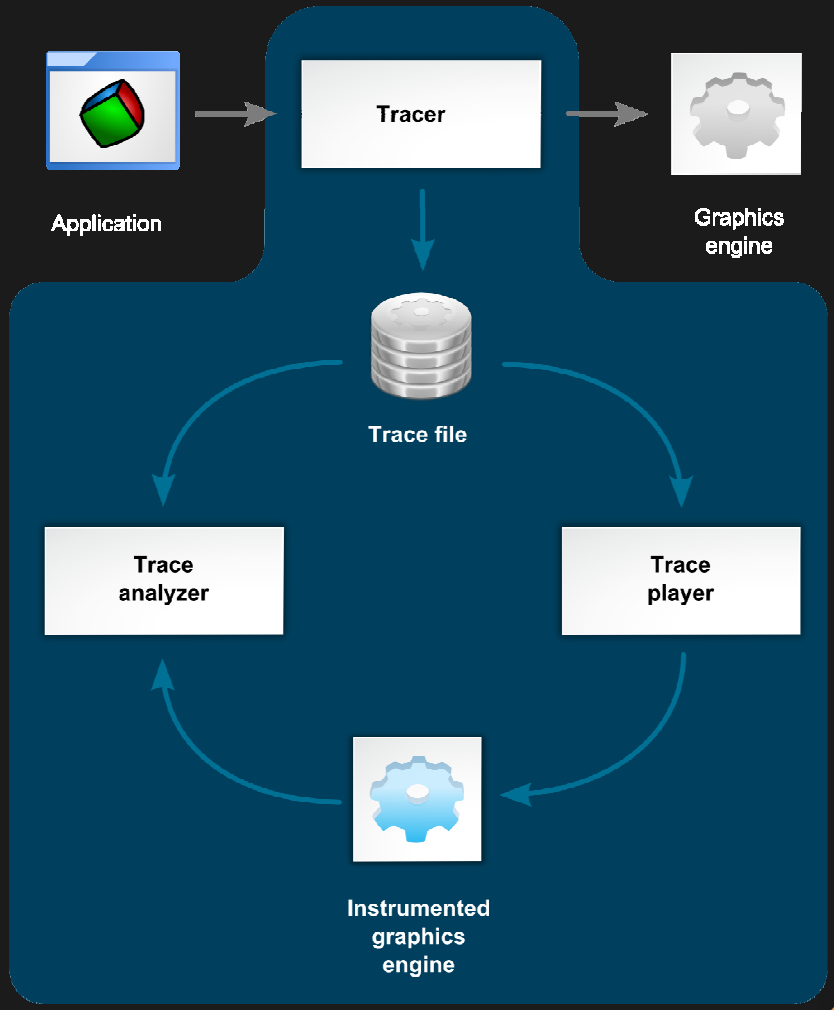
- Tracer
- Trace player
- Trace analyzer

Cross-platform

Optimized for mobiles

Data-driven design

- OpenGL ES, OpenVG, EGL



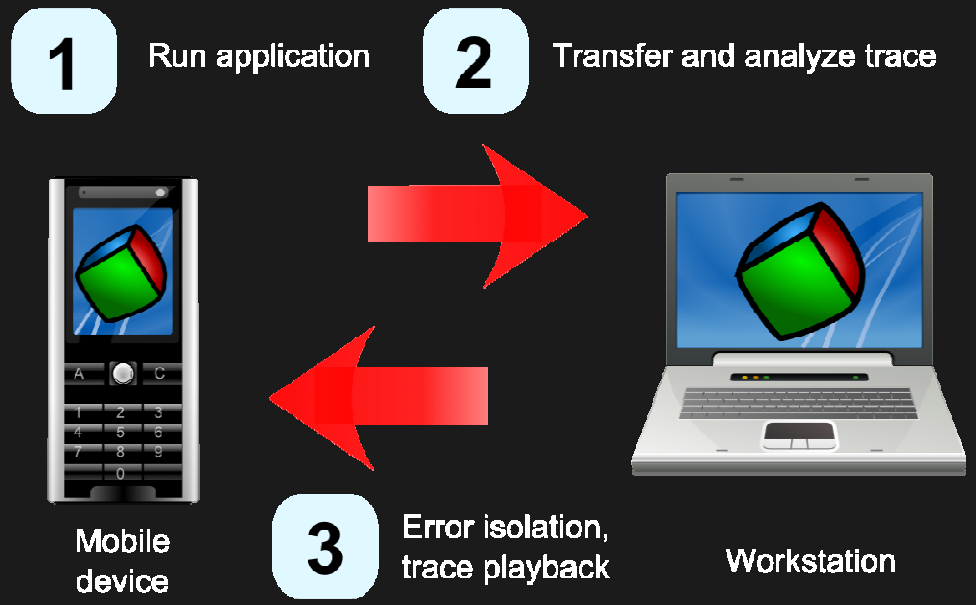
Related work

Tracing: Tracing interactive 3D graphics programs (Dunwoody & Linton, 1990), Chromium (Humphreys et al., 2002)

State tracking: Tracking graphics state for networked rendering (Buck et al., 2000)

Graphical debugging: PerfHUD (NVIDIA), gDEBugger (graphicREMEDY), PIX (Microsoft)

Workflow



Main use-cases

1. Debug visual errors and performance problems
2. Analyze application quality
3. Benchmark graphics engine

Use case 1: Graphics debugging

Original application

Trace on reference engine

Rendering error in application

Play trace on reference engine

Reference engine output OK ➔ bug in the engine

Isolate test frame and debug

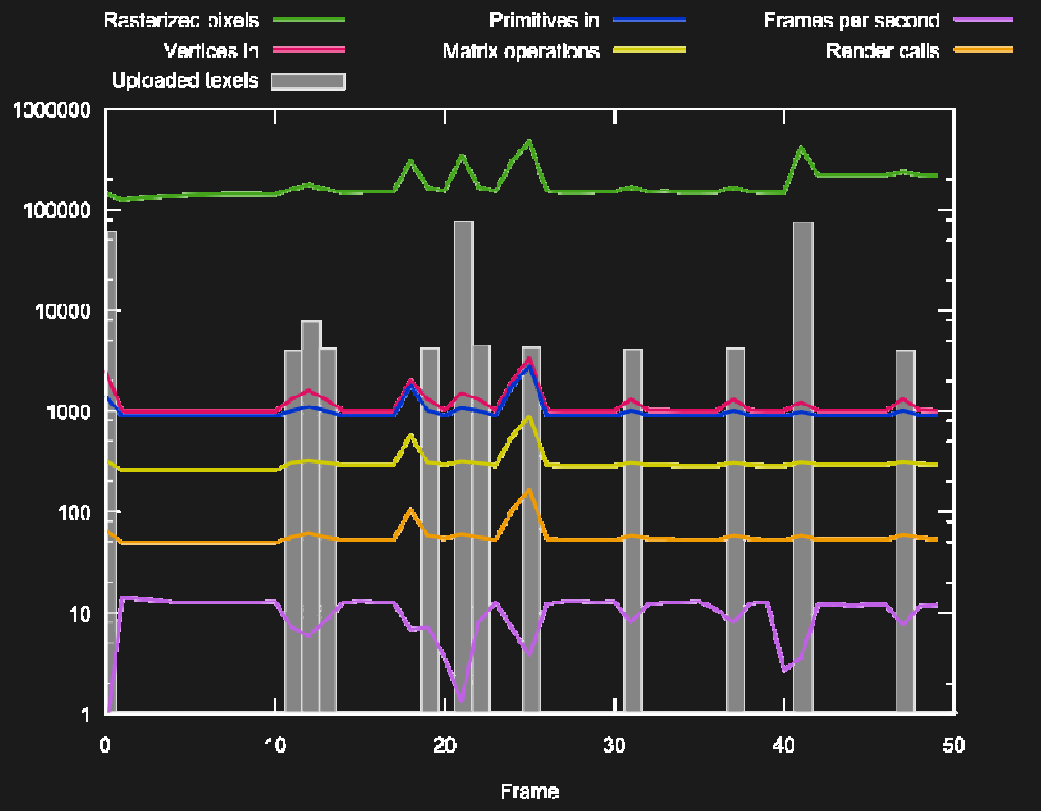
Use case 2: Analyze application quality

Offline trace analysis

Graphics expert system

Quality problems

- High resource utilization
- Suboptimal API usage



Use case 3: Benchmark graphics engine

Benchmarking with traces

Two approaches

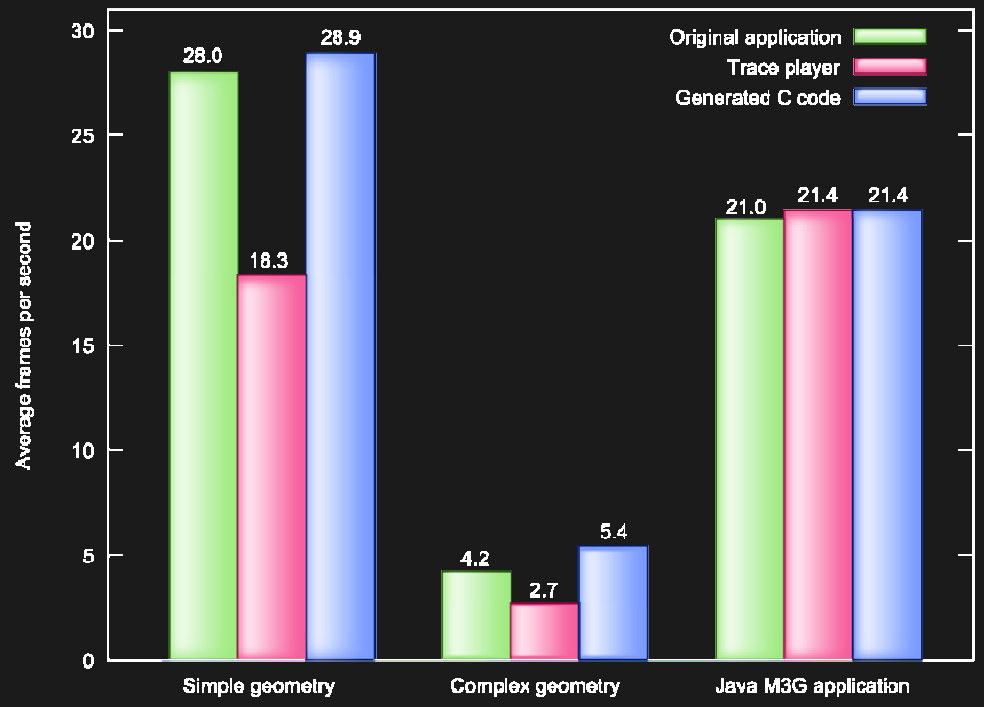
- Trace player: overhead
- Native code

Edit traces

- Custom benchmarks

Single frame

- Steady-state benchmarks



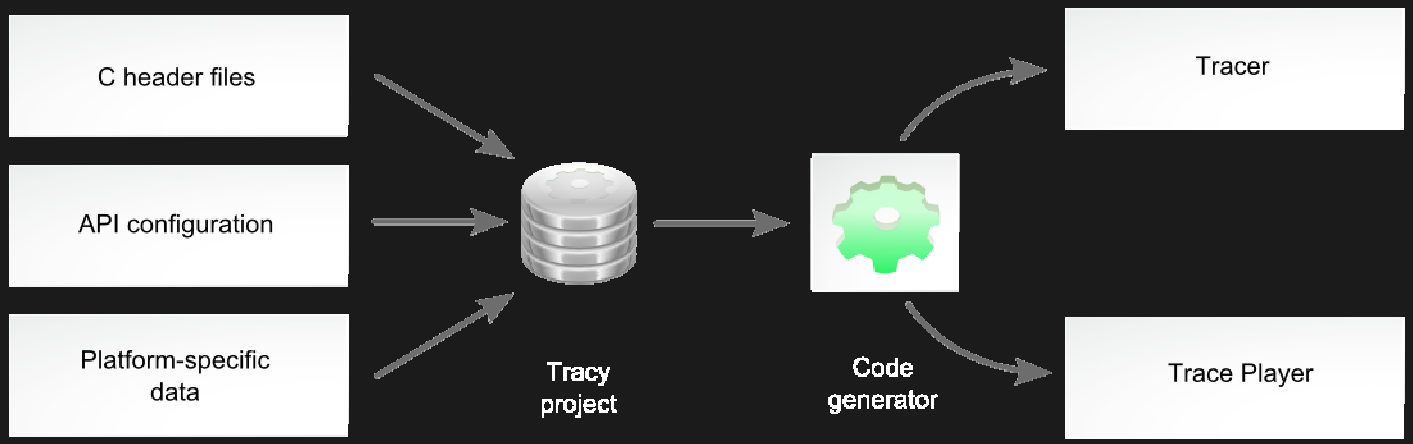
ANSI C code generation

Trace ➔ platform-independent ANSI C source code

- Highly portable
- Very low performance overhead ➔ benchmarking and profiling

- Challenges
  - Compiler limitations
  - Workaround: data arrays in assembly language

Data-driven API configuration



C header files

- API functions, objects and constants

API configuration

- Special functions
- State structure
- Serialization rules

Platform-specific data

API configuration example

glLightfv(GLenum **light**, GLenum **pname**, const GLfloat ***params**)

```
glLightfv
{
    light:              "ctx.light"
    pname:              "ctx.light.parameter"
    params
    {
        state:          "ctx.light.parameter.value"
        metatype(class = "array", size = 4)
        [
            size(condition = "pname", value = "GL_SPOT_DIRECTION"): 3
            size(condition = "pname", value = "GL_SPOT_EXPONENT"): 1
            size(condition = "pname", value = "GL_SPOT_CUTOFF"): 1
            size(condition = "pname", value = "GL_CONSTANT_ATTENUATION"): 1
            size(condition = "pname", value = "GL_LINEAR_ATTENUATION"): 1
            size(condition = "pname", value = "GL_QUADRATIC_ATTENUATION"): 1
        ]
    }
}
```

params array: 4 components by default

If *pname* equals *GL_SPOT_DIRECTION:* 3 components

If *pname* equals *GL_SPOT_EXPONENT:* 1 component

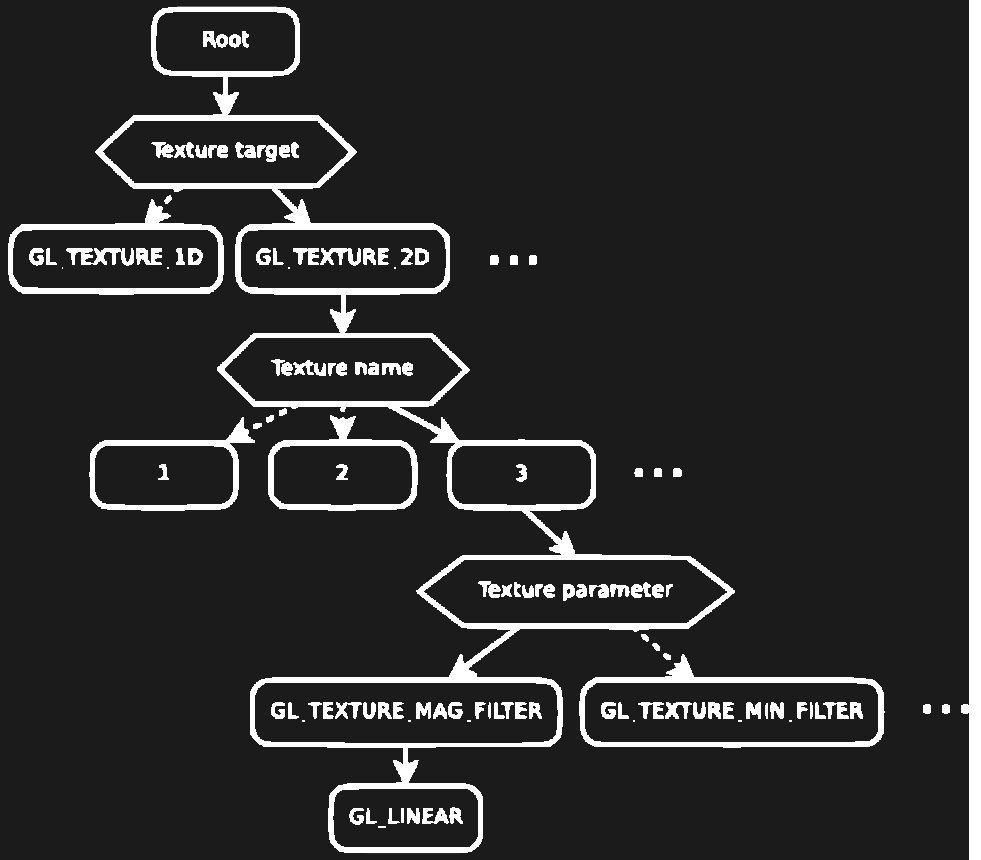
State tracking

State tree: graphics API state

Branches: function call parameters ➔ map API calls to state changes

Use cases

- Implicitly defined parameters
- API state computation
- Trace optimization



State tracking example

OpenGL ES Vertex buffer objects (VBO)

glBindBuffer() – Set active VBO

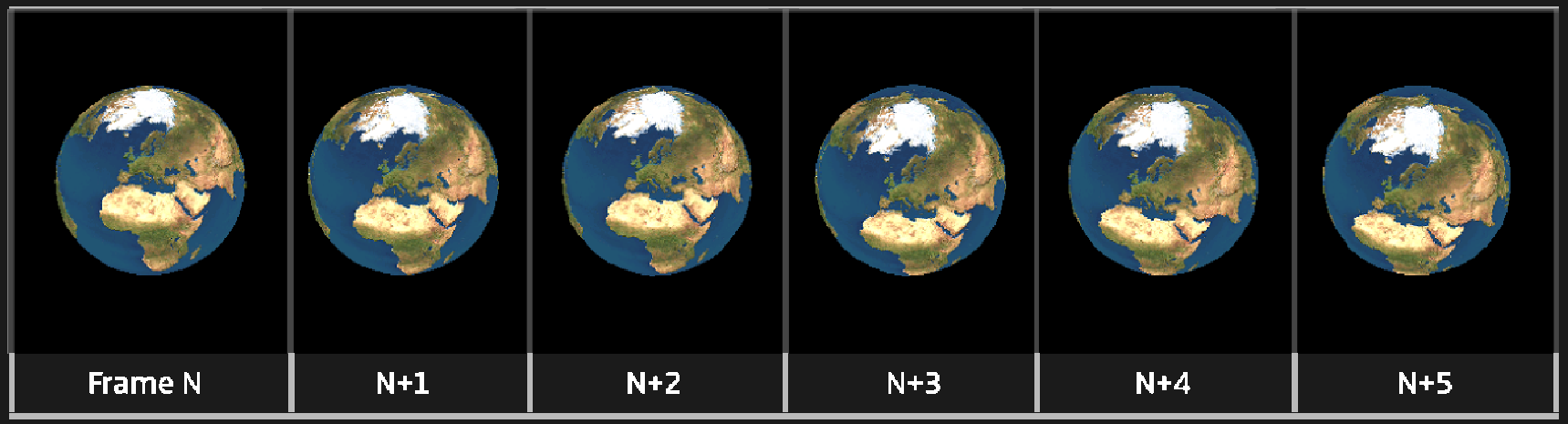
glBufferData() – Set active VBO data

➔ glBufferData depends on glBindBuffer

Dependency encoded in state tree paths:

- glBindBuffer() ➔ **root.vbo.handle**
- glBufferData() ➔ **root.vbo.handle.data**
- glBindBuffer() path is a prefix for glBufferData() path

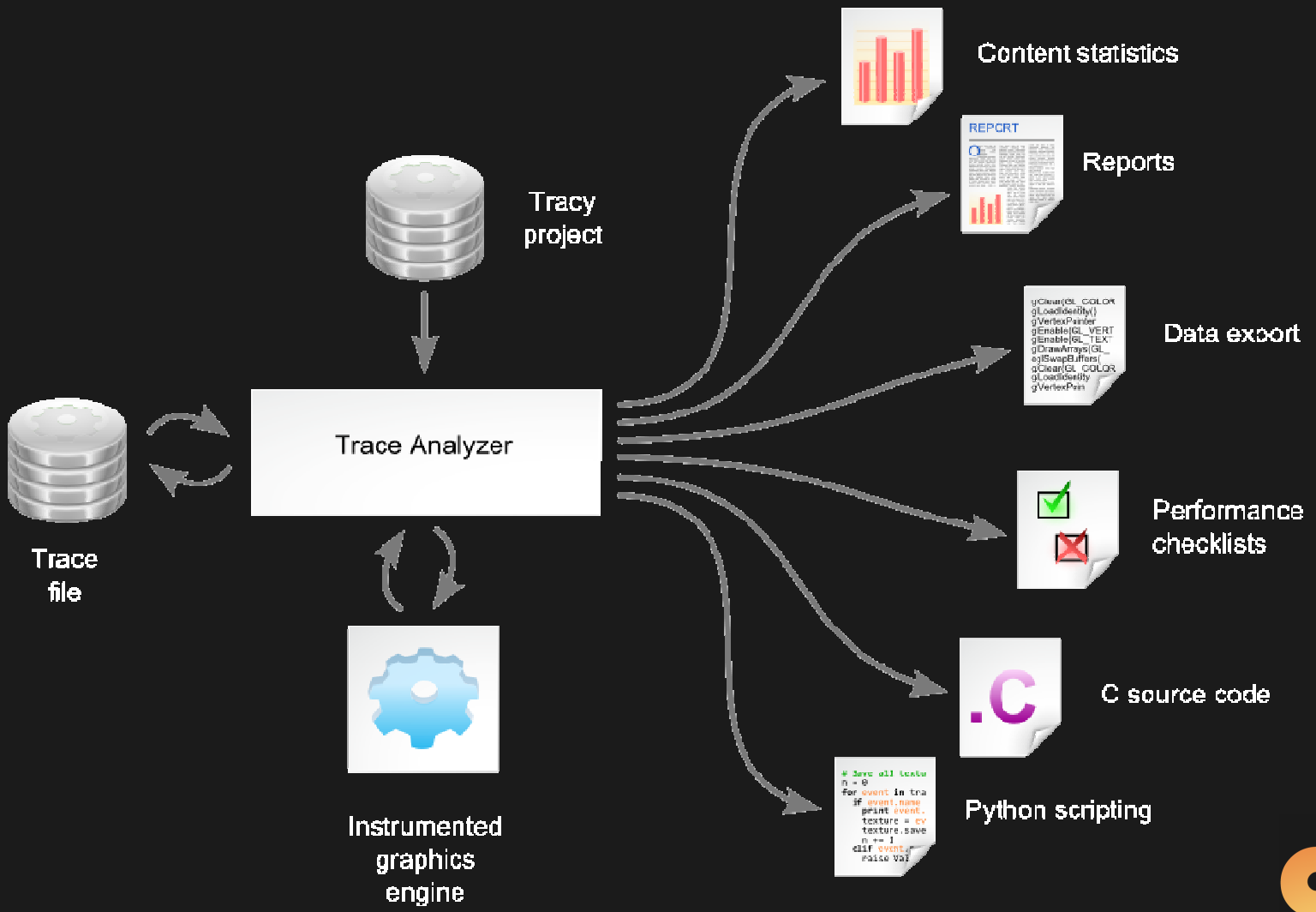
Trace compression



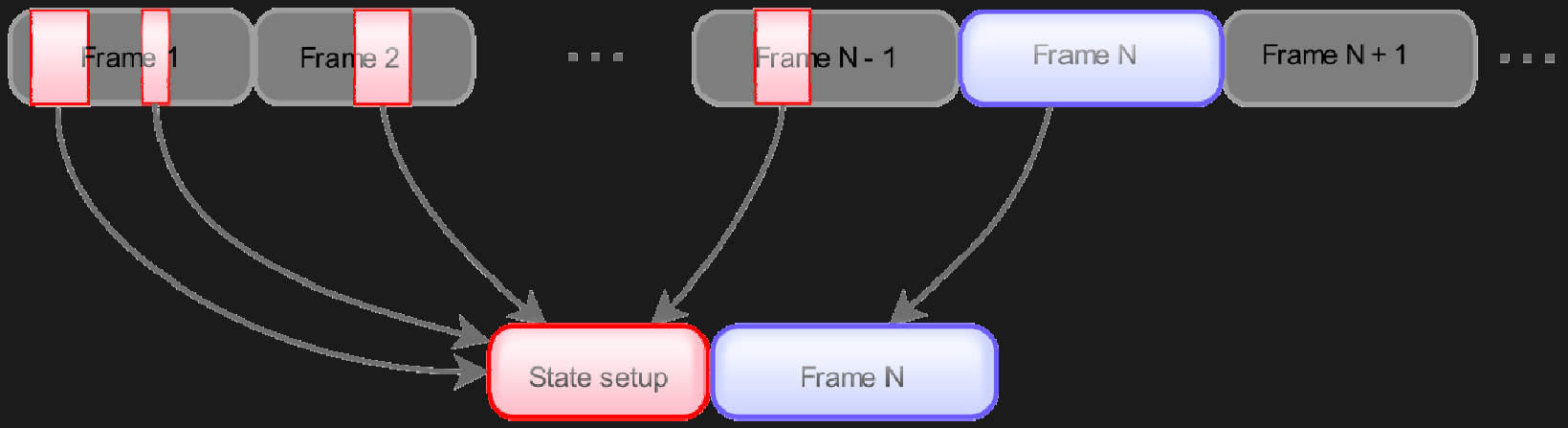
Runtime trace compression

- Internal copies of reused arrays
- Benefits: ~10-100x reduction in size, 1 FPS ➔ 10 FPS improvement in tracing performance

Trace analysis and transformation



Trace analysis: Frame extraction



Cull redundant commands, create state setup sequence

Use cases

- Error isolation
- Test design
- Benchmark design

Conclusion

Offline graphics debugging

Optimized for mobile graphics

Flexible data-driven design

Reliable profiling and benchmarking

Future work

- OpenGL ES 2.0
- Content clustering for benchmarking

Thank you – Questions?

Sami Kyöstilä <sami.kyostila@nokia.com>
Kari J. Kangas <kari.j.kangas@nokia.com>
Kari Pulli <kari.pulli@nokia.com>