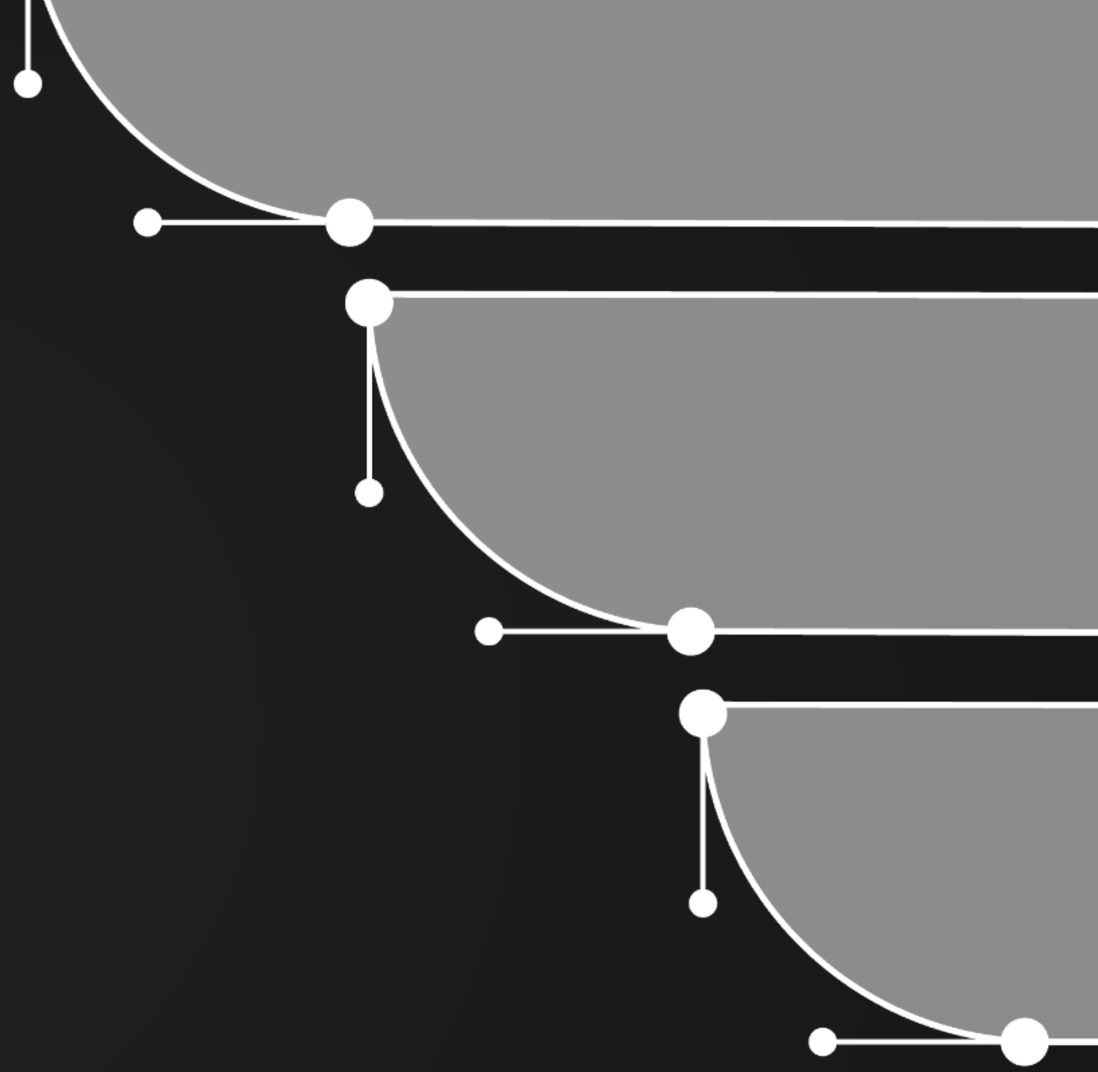
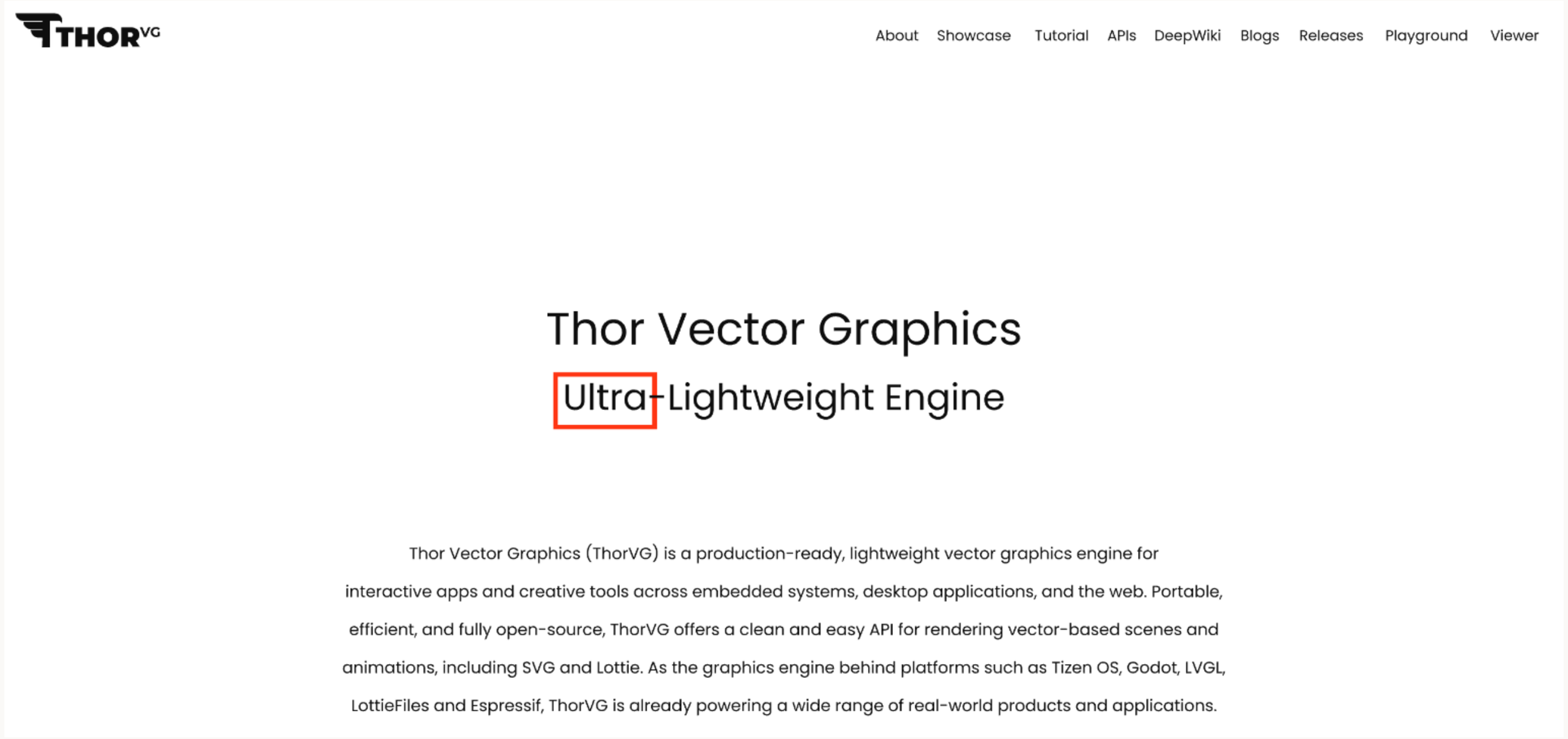
OSSCA2026/
ThorVG

ThorVG C++

SoonGeon Noh



ThorVG — Ultra-Lightweight Engine



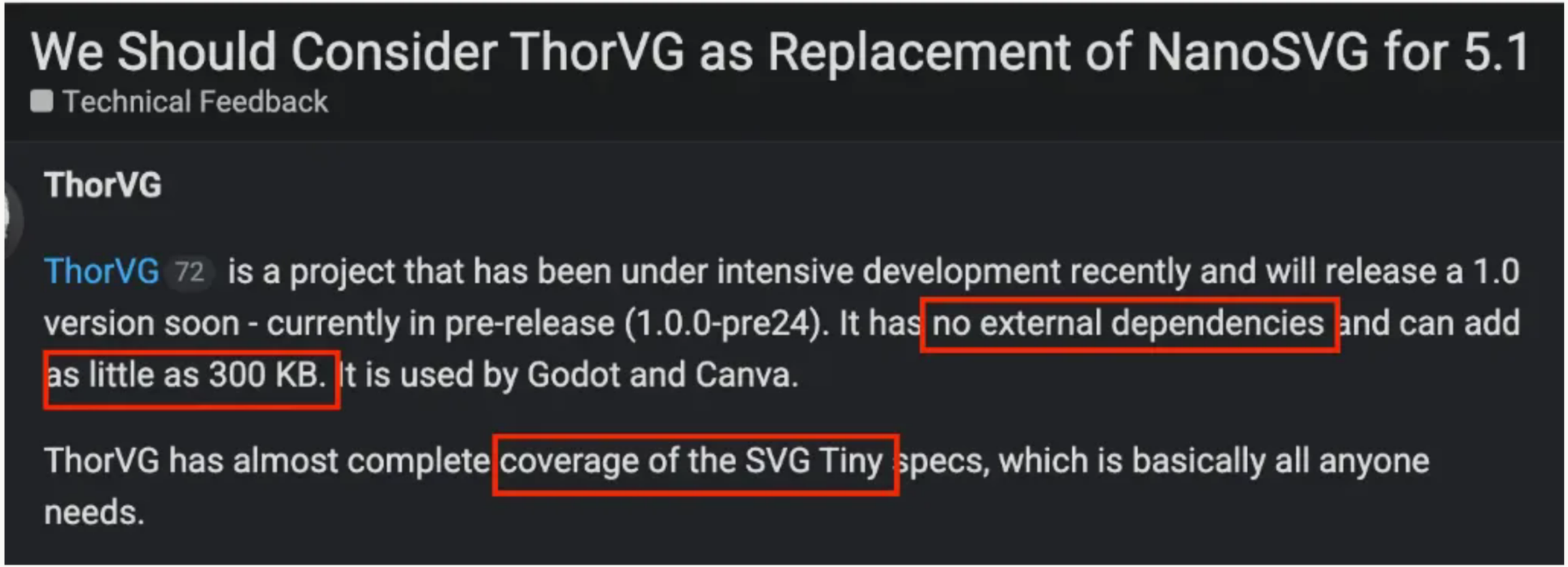
ThorVG — Ultra-Lightweight Engine

Broad Portability

ThorVG is based on the C++ standard and provides consistent functionality across various platforms through an abstraction layer that minimizes dependence on specific operating systems or hardware.

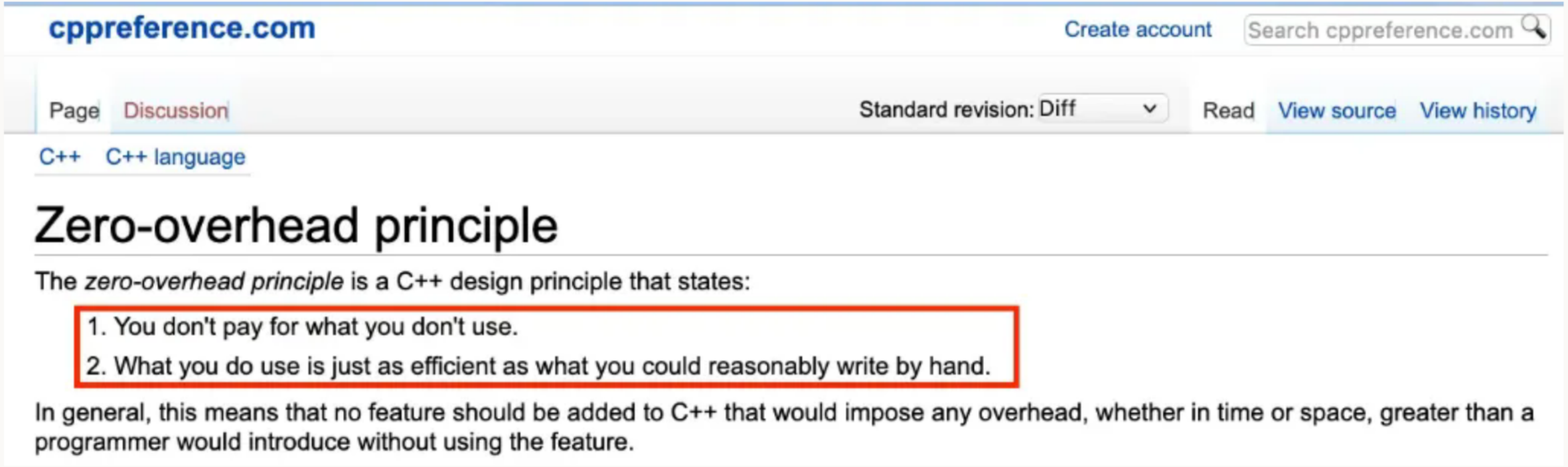
- **Extensive Platform Support**: ThorVG supports web platforms, desktop operating systems such as Windows, macOS, and Linux, mobile platforms including Android and iOS, as well as embedded systems like Tizen and RTOS-based environments.
- **Microcontroller Support**: ThorVG has been shown to run on microcontrollers like the ESP32, demonstrating its efficiency even within environments with highly limited memory and storage.
- **Headless Rendering Support**: ThorVG can perform rendering without a display server, enabling use cases such as server-side graphics processing or offline rendering tools.

ThorVG — Ultra-Lightweight Engine



Blender Community

C++ — Zero-overhead principle



cppreference.com

Create account

Search cppreference.com

Page Discussion

Standard revision: Diff

Read View source View history

C++ C++ language

Zero-overhead principle

The *zero-overhead principle* is a C++ design principle that states:

1. You don't pay for what you don't use.
2. What you do use is just as efficient as what you could reasonably write by hand.

In general, this means that no feature should be added to C++ that would impose any overhead, whether in time or space, greater than a programmer would introduce without using the feature.

Contents

01

빌드에서 실행까지

Meson 옵션은 어떻게 실행 파일까지 이어지는가?

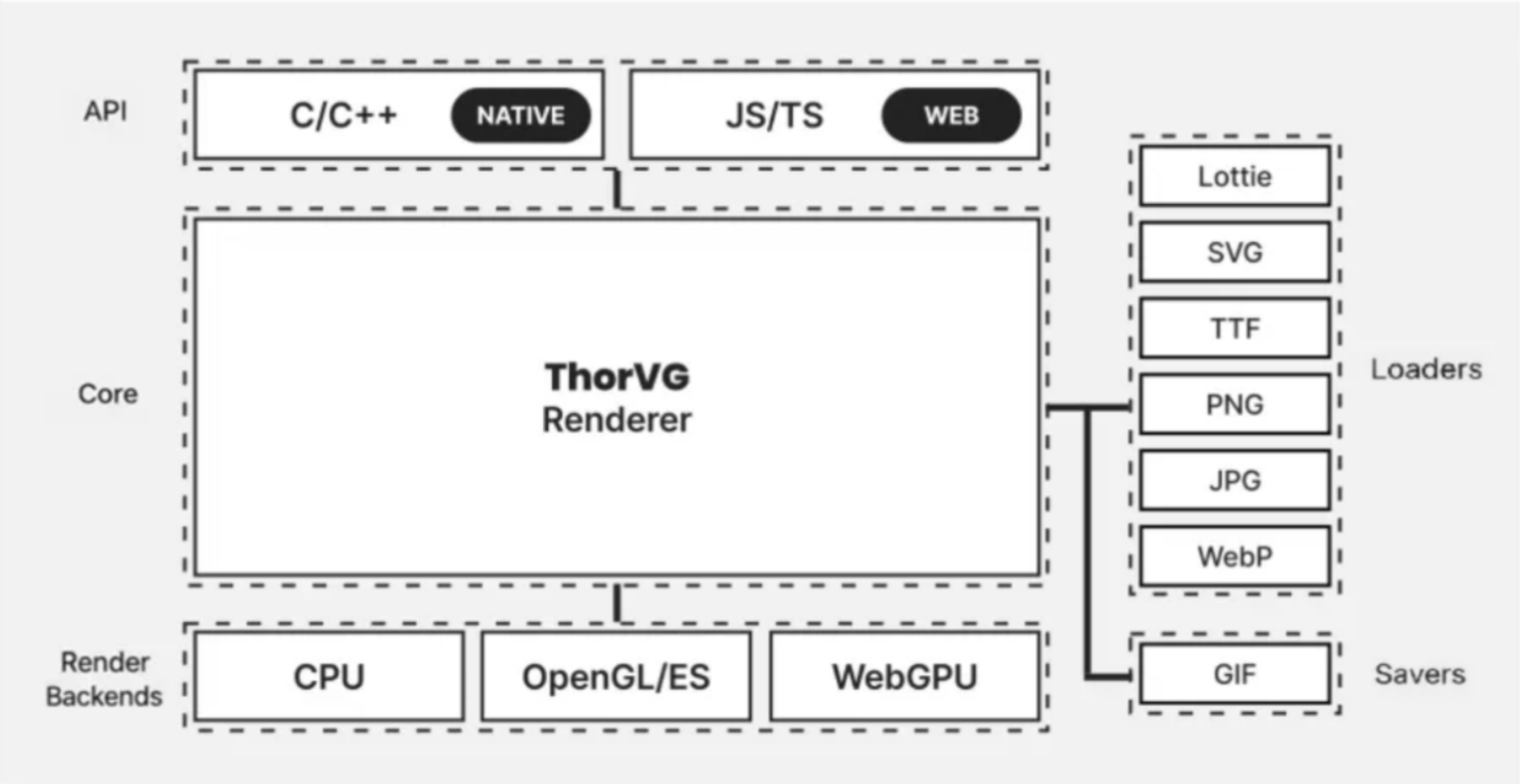
1-1

ThorVG 모듈과 빌드

C/C++ 라이브러리가 빌드되기 까지의 과정

ThorVG의 모듈

ThorVG is designed for a wide range of programs, offering adaptability for integration and use in various applications and systems. It achieves this through a single binary with selectively buildable, modular components in a building block style. This ensures both optimal size and easy maintanence.



The core library of ThorVG maintains a binary size of approximately 170KB. This is significantly smaller compared to graphics engines designed primarily for desktop environments and offers the following

Meson 옵션으로 보는 ThorVG의 모듈과 경량성

```
# source/thorvg/meson_options.txt:12-16
option('loaders',
   type: 'array',
   choices: ['', 'svg', 'png', 'jpg',
     'lottie', 'ttf', 'otf', 'webp', 'all'],
   value: ['svg', 'lottie', 'ttf'])

# src/loaders/meson.build:15-23
if png_loader
    if get_option('static')
        subdir('png')           # 내장
    else
        subdir('external_png') # 시스템
```

Loader

```
# source/thorvg/meson_options.txt:1-5
option('engines',
   type: 'array',
   choices: ['cpu', 'gl', 'wg', 'all'],
   value: ['cpu'], ...)

# src/renderer/meson.build:3-9
if cpu_engine
    subdir('cpu_engine')
endif
if gl_engine or wg_engine
    subdir('gpu_engine')
endif
```

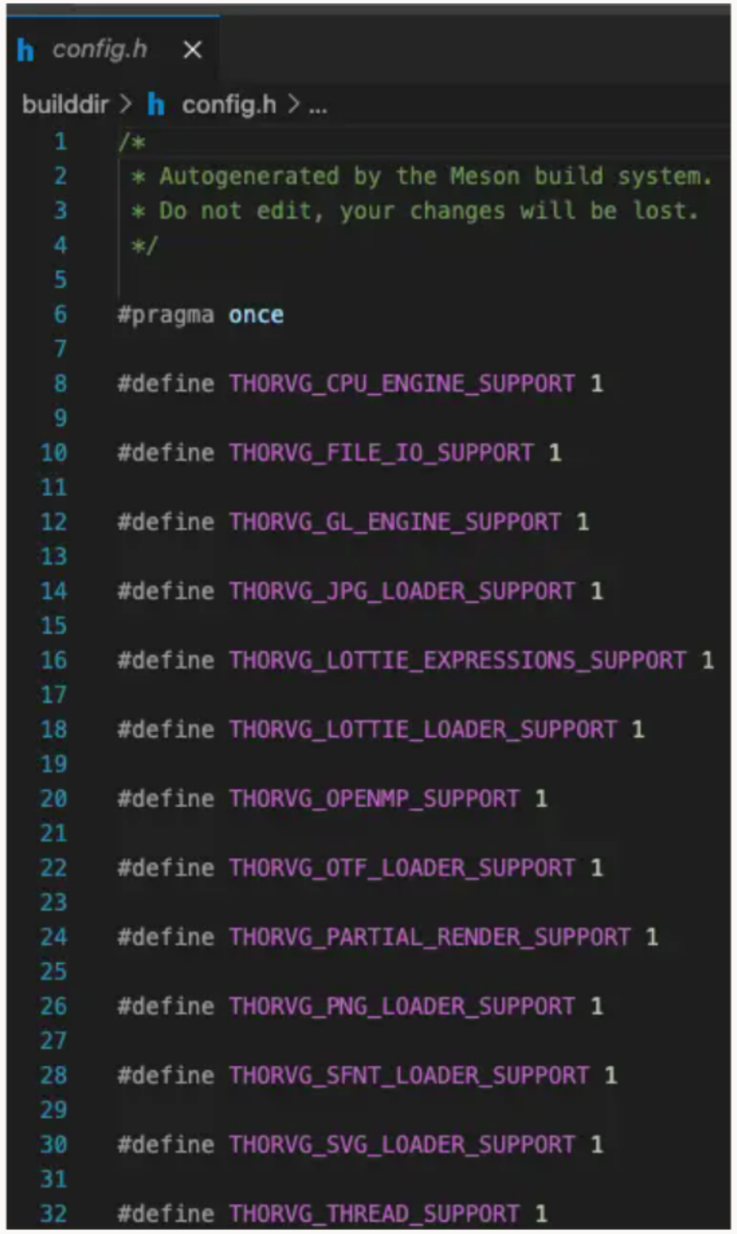
Engine

옵션과 config.h: Preprocessor

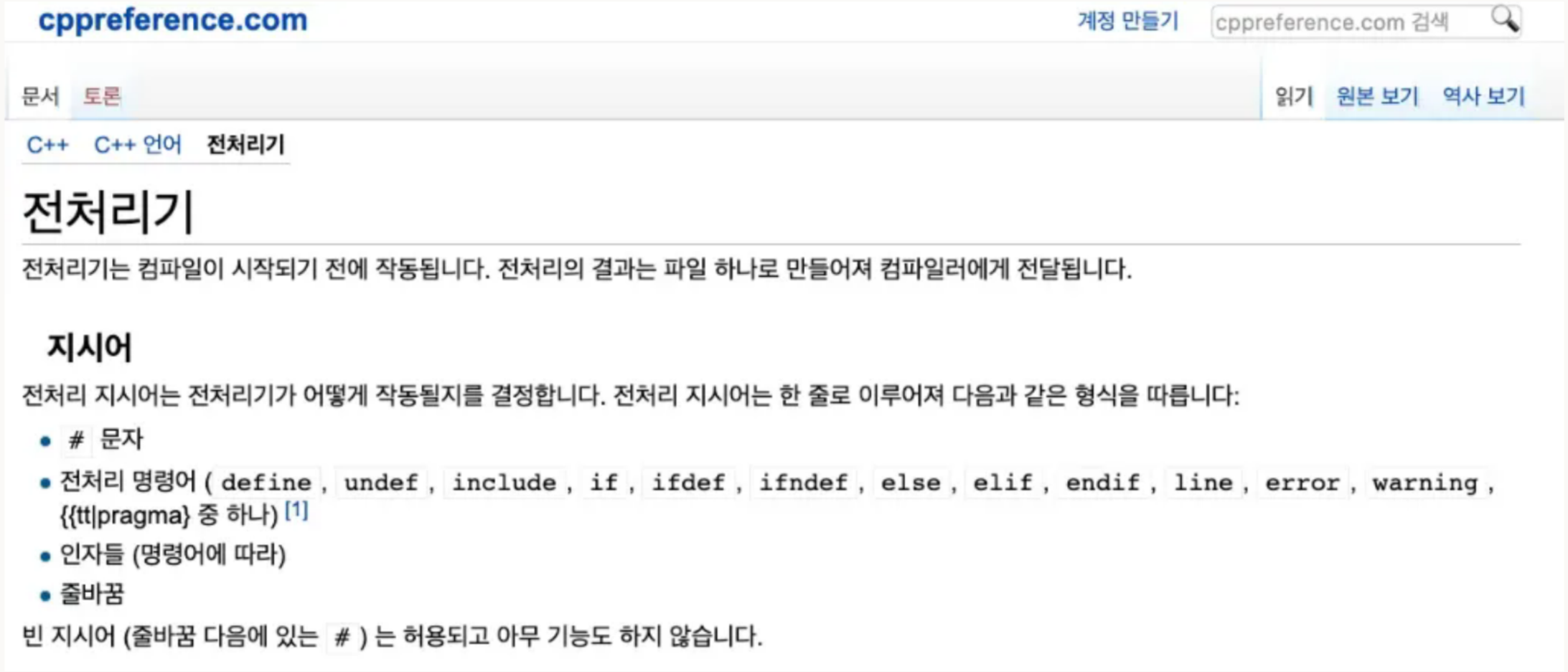
```
# source/thorvg/meson.build:20-85
if all_engines or 'cpu' in engines
    config_h.set10(
        'THORVG_SW_RASTER_SUPPORT', true)
endif

// 소스 쪽 — config.h를 읽는다
#ifdef THORVG_SW_RASTER_SUPPORT
    ... raster 코드가 존재 ...
#endif
```

옵션 → config.h 매크로 생성

#ifdef 조건부 컴파일 컴파일에서 제외됨



옵션과 config.h: Preprocessor



cppreference.com 계정 만들기 cppreference.com 검색

문서 토론 읽기 원본 보기 역사 보기

C++ C++ 언어 전처리기

전처리기

전처리기는 컴파일이 시작되기 전에 작동됩니다. 전처리의 결과는 파일 하나로 만들어져 컴파일러에게 전달됩니다.

지시어

전처리 지시어는 전처리기가 어떻게 작동될지를 결정합니다. 전처리 지시어는 한 줄로 이루어져 다음과 같은 형식을 따릅니다:

- `#` 문자
- 전처리 명령어 (`define` , `undef` , `include` , `if` , `ifdef` , `ifndef` , `else` , `elif` , `endif` , `line` , `error` , `warning` , {{tt|pragma} 중 하나) [1]
- 인자들 (명령어에 따라)
- 줄바꿈

빈 지시어 (줄바꿈 다음에 있는 `#`) 는 허용되고 아무 기능도 하지 않습니다.

cppreference.com

옵션과 config.h: Preprocessor

```
141  SwCanvas* SwCanvas::gen(EngineOption op) noexcept
142  {
143  #ifdef THORVG_CPU_ENGINE_SUPPORT
144      if (engineInit > 0) {
145          auto renderer = new SwRenderer(TaskScheduler::threads(), op);
146          renderer->ref();
147          auto ret = new SwCanvas;
148          ret->pImpl->renderer = renderer;
149          return ret;
150      }
151  #endif
152      TVGLOG("RENDERER", "SwCanvas is not supported");
153      return nullptr;      Hermet Park, 10 months ago • renderer: unify sw/gl
154  }
```

THORVG_CPU_ENGINE_SUPPORT 가 꺼지면

해당 부분은 컴파일러에게 전달되지 않는다

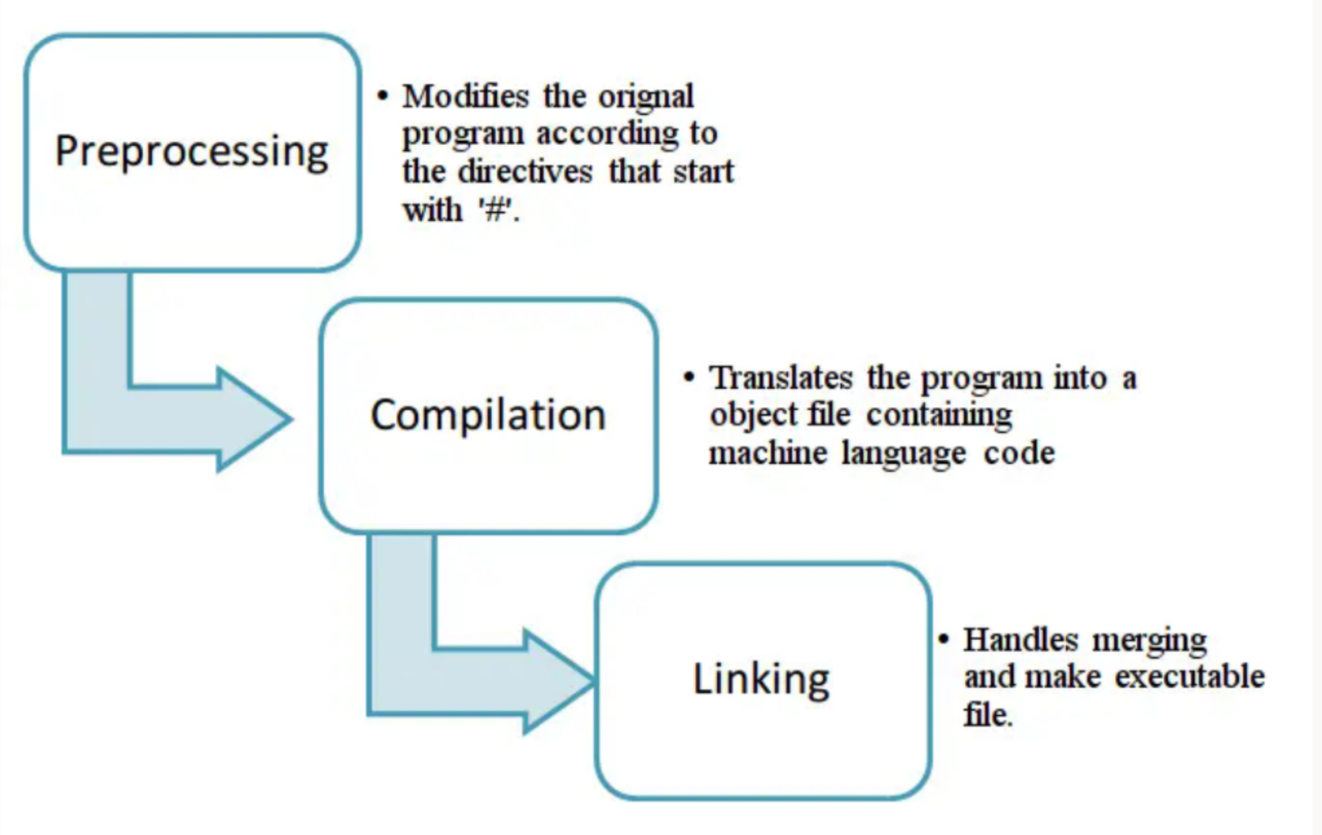
Preprocessor -> Compiler -> Linker

Meson 에서 설정된 옵션과 config_h 를 통해

Preprocessor 에서 코드를 전처리하고

사용자가 필요한 기능만 담긴 최소한의 엔진이 빌드

zero-overhead



isocpp.org/blog/2013/11/compiling-and-linking

1-2

Shared? Static?

C/C++ 라이브러리의 빌드 산출물

Shared는 실행 시, Static은 링크 시 결합

Shared (.dylib · .so · .dll)

실행 시 dynamic loader가 심볼 해석

여러 프로세스가 코드를 공유

라이브러리만 교체해 업데이트 가능

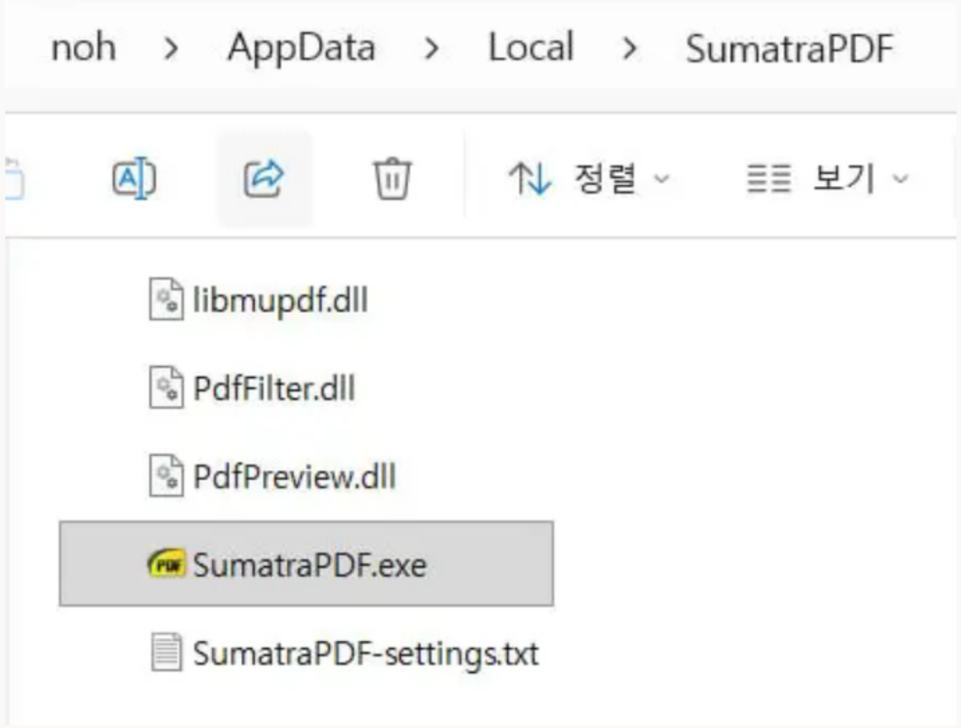
Static (.a · .lib)

링크 시 실행 파일 안으로 복사됨

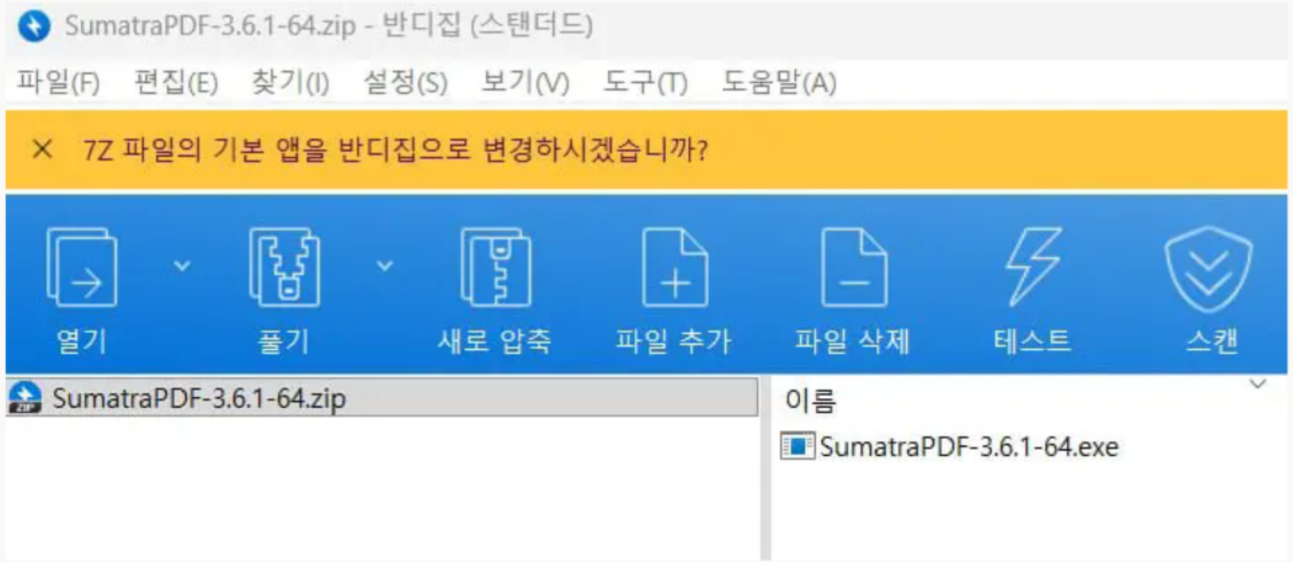
시작 빠름 · 배포가 파일 하나

실행 파일 크기는 커질 수 있음

Shared vs Static 예시



Shared



Static, Portable

Shared: FFMPEG, License Compliance

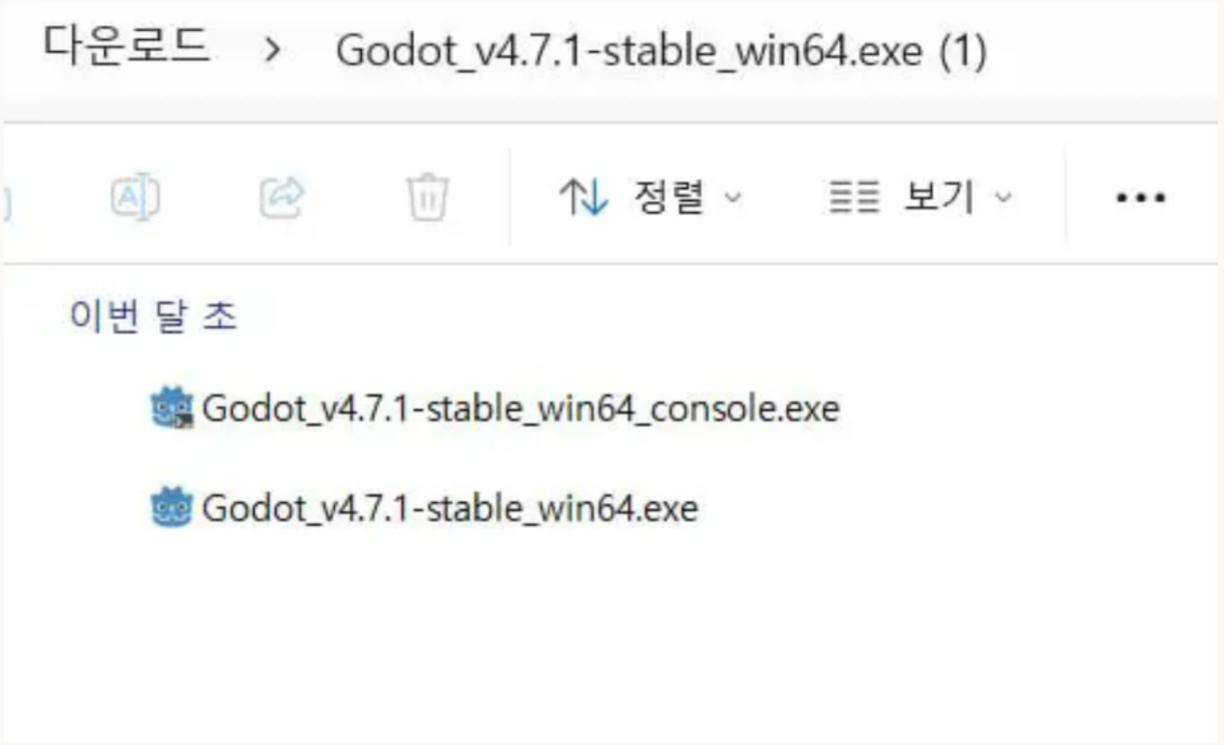
License Compliance Checklist

The following is a checklist for LGPL compliance when linking against the FFmpeg libraries. It is not the only way to comply with the license, but we think it is the easiest. There are also a few items that are not really related to LGPL compliance but are good ideas anyway.

1. Compile FFmpeg **without** "--enable-gpl" and **without** "--enable-nonfree".
2. Use dynamic linking (on windows, this means linking to dlls) for linking with FFmpeg libraries.
3. Distribute the source code of FFmpeg, no matter if you modified it or not.
4. Make sure the source code corresponds exactly to the library binaries you are distributing.
5. Run the command "git diff > changes.diff" in the root directory of the FFmpeg source code to create a file with only the changes.
6. Explain how you compiled FFmpeg, for example the configure line, in a text file added to the root directory of the source code.
7. Use tarball or a zip file for distributing the source code.
8. Host the FFmpeg source code on the same webserver as the binary you are distributing.
9. Add "This software uses code of FFmpeg licensed under the LGPLv2.1 and its source can be downloaded here" to every page in your website where there is a download link to your application.
10. Mention "This software uses libraries from the FFmpeg project under the LGPLv2.1" in your program "about box".
11. Mention in your EULA that your program uses FFmpeg under the LGPLv2.1.
12. If your EULA claims ownership over the code, you have to **explicitly** mention that you do not own FFmpeg, and where the relevant owners can be found.
13. Remove any prohibition of reverse engineering from your EULA.
14. Apply the same changes to all translations of your EULA.
15. Do not misspell FFmpeg (two capitals F and lowercase "mpeg").
16. Do not rename FFmpeg dlls to some obfuscated name, but adding a suffix or prefix is fine (renaming "avcodec.dll" to "MyProgDec.dll" is not fine, but to "avcodec-MyProg.dll" is).
17. Go through all the items again for any LGPL external library you compiled into FFmpeg (for example LAME).
18. Make sure your program is not using any GPL libraries (notably libx264).

FFMPEG, 라이센스 권고 사항

Static: Godot



Instructions

- Extract and run. Godot is self-contained and does not require installation.
- If you run into an issue, check the Troubleshooting page for common issues and their solutions.

Godot is code-signed and notarized for macOS by *Prehensile Tales B.V.*.

하나의 exe 파일, 쉬운 배포

default_library -> Shared, Static

```
lib_type = get_option('default_library')

if lib_type == 'shared'
    compiler_flags += ['-DTVG_EXPORT', '-DTVG_BUILD']
else
    compiler_flags += ['-DTVG_STATIC']
endif
```

Meson 옵션에서 default_library 옵션을
사용해서 Shared, Static 빌드 결정 가능

Example의 의존성: thorvg-1

meson install

library·헤더와 함께 thorvg-1.pc를 설치

Example은 pkg-config 경로만 알면

dependency()로 ThorVG를 발견

```
# 생산자 — ThorVG 설치
meson install -C builddir
#    $PREFIX/lib/pkgconfig/thorvg-1.pc

# 연결 고리 — 설치 결과를 찾게 하기
PKG_CONFIG_PATH="$PREFIX/lib/pkgconfig" \
  meson setup build-demo

# 소비자 — thorvg.example/src/meson.build:1
thorvg_dep = dependency('thorvg-1')
```

Source · source/thorvg.example/src/meson.build:1-3 · $PREFIX/lib/pkgconfig/thorvg-1.pc

Example 실행

이슈 작업 등에서

ThorVG 를 Shared 로 빌드하고 Install 하면

Example 다시 빌드하지 않고,

새로운 ThorVG 으로 테스트할 수 있다.

1-3

다양한 언어로 바인딩되는 ThorVG

C++의 맹글링과 C API, 언어 호환성: Kotlin · Swift · Dart · C# · Rust...

API와 ABI

Application binary interface

23 languages

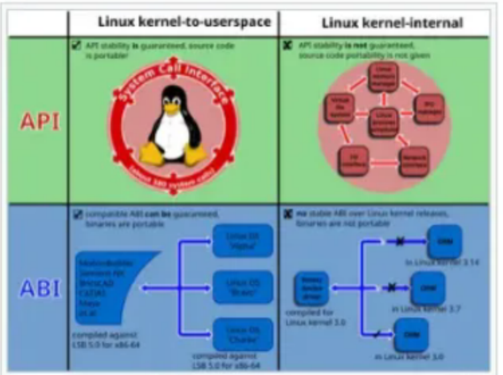
Article Talk Read Edit View history

From Wikipedia, the free encyclopedia

An **application binary interface** (**ABI**) is an interface exposed by software that is defined for in-process machine code access. Often, the exposing software is a library, and the consumer is a program.

An ABI is at a relatively low level of abstraction. Interface compatibility depends on the target hardware and the software build toolchain. In contrast, an application programming interface (API) defines access in source code, which is a relatively high-level, hardware-independent, and human-readable format. An API defines an interface at the source code level, before compilation, whereas an ABI defines an interface to compiled code.

API compatibility is generally the concern for system design and of the toolchain. However, a programmer may have to deal with an ABI directly when writing a program in multiple languages or when using multiple compilers for the same language.



A high-level comparison of in-kernel and kernel-to-userspace APIs and ABIs

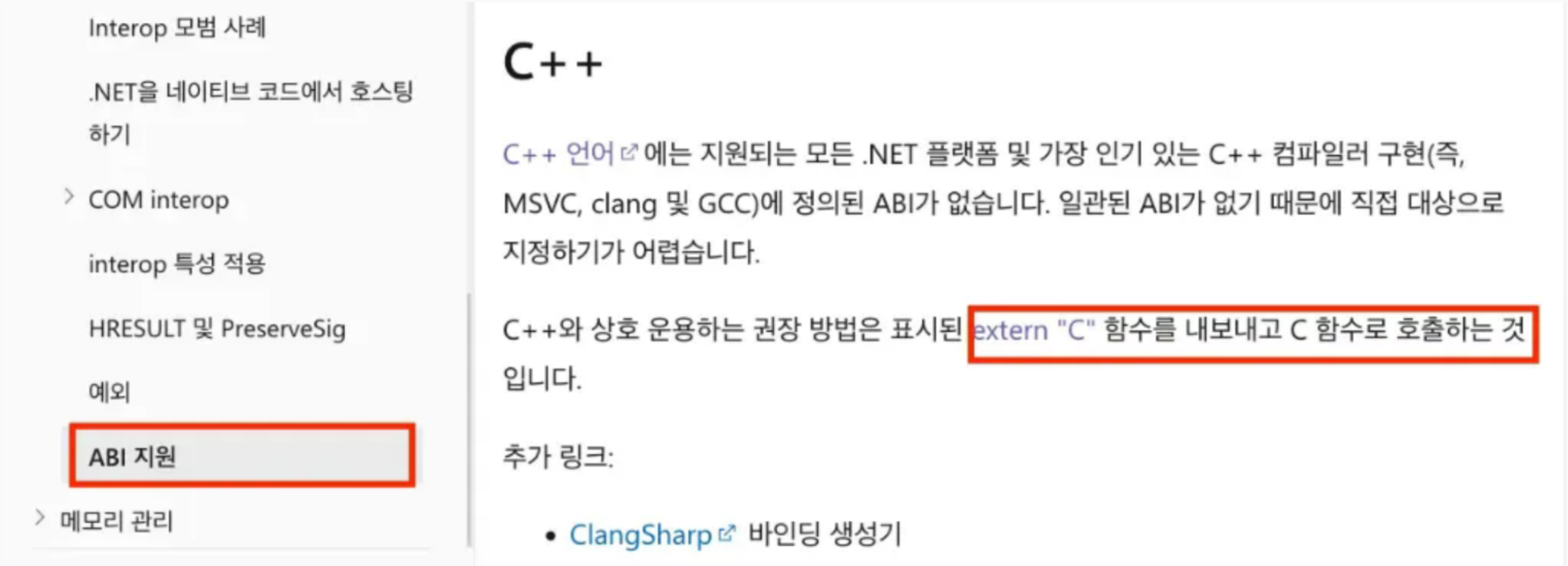
Wikipedia

API — 헤더에 적힌 함수 서명, 소스를 컴파일할 때의 약속, 단일 언어

ABI — 호출 규약 · 심볼 이름 ·타입 레이아웃, 링크·실행의 약속

API는 서로 언어가 다르면 호환되지 않지만 ABI가 같다면 언어에 관계 없이 사용

ThorVG가 지원하는 C API와 ABI



learn.microsoft

C++ 심볼은 맹글링 — extern "C"가 언어 사이의 경계를 지킴

C++의 오버로딩·네임스페이스

- 이름에 타입을 인코딩 (mangling)
- 컴파일러마다 다름

extern "C"

- 맹글링 OFF
- 함수 이름과 타입에 C 언어 링크 규약을 지정
- 런타임에 이름으로 직접 찾기 쉬움

```
$ nm libthorvg.so | grep gen
_ZN3tvg5Shape3genEv    # 링커가 보는 이름
$ c++filt _ZN3tvg5Shape3genEv
tvg::Shape::gen()      # 사람이 보는 이름

// bindings/capi/tvgCapi.cpp:35,43
extern "C" {
TVG_API Tvg_Result
tvg_engine_init(unsigned threads) {
    return (Tvg_Result)
        Initializer::init(threads);
```

여러 바인딩 방법

① C API(thorvg_capi)

C# · Unity — `thorvg/thorvg.unity` — **DllImport(P/Invoke)로 tvg_* 직접 선언** · `Runtime/Sys/TvgLib.cs`

Swift · Apple — `thorvg/thorvg.swift` — **C interop — tvg_engine_init() 직접 호출** · `swift/Models/Engine.swift`

Rust · LottieFiles — `LottieFiles/dotlottie-rs` — `bindgen` — `tvg_engine_init()` · `src/renderer/thorvg.rs`

② extern "C"·JNI glue

Dart · Flutter — `thorvg/thorvg.flutter` — `dart:ffi` · `tvgFlutterLottieAnimation.cpp`

Kotlin · Android — `thorvg/thorvg.android` — **external fun(JNI) + 자체 C++ glue** · `thorvg-core/src/main/cpp/`

③ Web, C++를 embind

TS · Web — `thorvg/thorvg.web` — `C++ API` · `tvgWasmWebCanvas.cpp`

02

ThorVG로 보는 C++ 개념들

2-1

Struct, Class, 메모리 레이아웃

struct와 class의 차이는 기본값뿐

기본 접근 — public vs private

기본 상속 — public vs private

ThorVG 공개 API 내부

- struct가 기본
- private, protected가 필요한 곳만 명시

```
// 차이는 기본값뿐
struct S { int x; };       // 기본 public
class  C { int x; };       // 기본 private

struct D : S {};           // 기본 public 상속
class  E : S {};           // 기본 private 상속

// inc/thorvg.h — ThorVG는 struct가 기본
struct TVG_API Paint
{
protected:                 // 필요한 곳만 명시
    virtual ~Paint();
};
```

컴파일러가 만들어 주는 여섯 개의 특수 멤버 함수

컴파일러가 자동으로 만들어 주는 여섯 함수

- 생성자, 소멸자, = 연산자
- 복사(const T&) = 둘 다 유효한 사본
- 이동(T&&) = 소유권 이전

관련 키워드

- Rule of Five/Three/Zero

```
struct T
{
    T();                          // 생성자
    ~T();                         // 소멸자
    T(const T& o);                // 복사 생성자
    T& operator=(const T& o);     // 복사 대입
    T(T&& o);                     // 이동 생성자
    T& operator=(T&& o);          // 이동 대입
};

// 소멸자·복사·이동 중 하나라도 직접 다루면
// 나머지도 함께 살펴야 함 — Rule of Five
```

ThorVG API 구조체 선언에서 볼 수 있는 매크로

매크로 자동화

- 복사·대입 delete, 생성자 protected
- 생성과 소멸을 특정 함수에 책임을 맡김(gen, rel)
  - 사용자의 API 사용 실수를 막음

Impl, pImpl 이디엄

- 복잡한 구현을 감추고 외부 사용자에게 안전하고 쉬운 API 제공
- 사용자의 오용을 방지하기 위함
- 주의) 공개 API 에서만 사용하고 있는 패턴

  (내부에서는 지양하는 패턴, 복잡도와 바이너리 비용 존재)

```
// inc/thorvg.h:49 — 선언 끝의 매크로 3형제
#define _TVG_DECLARE_PRIVATE(A) \
protected: \
    A(const A&) = delete; \
    const A& operator=(const A&) = delete; \
    A()

#define _TVG_DECLARE_PRIVATE_BASE(A) \
    _TVG_DECLARE_PRIVATE(A); \
public: \
    struct Impl; Impl* pImpl  // pImpl 자동 선언

// _DERIVE = _PRIVATE + protected 소멸자
```

ThorVG 의 코드 컨벤션

```
class MyClass : public Base
{
    static constexpr float NAME_MAX = 100;
    enum Type : uint8_t { Default = 0, Type1, Type2, Type3 };

    Type type = Type::Default;   //assign an initial value when essential
    float length = 1.0f;
    char* name = nullptr;        //use nullptr, not NULL or 0
    float size;                  //omit initial value only when not logically needed

    //pack booleans together for optimal memory alignment (3 bytes here)
    bool initialized = false;
    bool valid = false;
    uint8_t refCount = 0;
```

주의점은

- 선언과 함께 초깃값 설정은 필요할때만

- memory alignment 를 지킬것

메모리 최적화) ThorVG 에서 볼 수 있는 union 활용

union 멤버 중 가장 사이즈가 큰 멤버만큼 메모리 차지

- 타입마다 다르게 해석해야하는 멤버들을
  하나로 묶어 메모리 낭비를 줄임

타입?

- channelSize/FillType 태그 -> buf8 or buf32
- Gradient 타입에 따라 다른 멤버 사용

```
// renderer/tvgRender.h:62-68 — 뷰 union
struct RenderSurface
{
    union {
        pixel_t*  data;    // 시스템 쪽 제네릭 포인터
        uint32_t* buf32;   // 픽셀당 4B — RGBA 한 번에
        uint8_t*  buf8;    // 픽셀당 1B — Grayscale8 마스크
    };
    uint32_t stride, w, h;
    ColorSpace cs;         // 어느 뷰가 유효한지 결정
    uint8_t channelSize;   // 4 또는 1
};
// SwFill
    union {
        SwLinear linear;
        SwRadial radial;
    };
```

메모리 최적화) 멤버 선언 순서가 구조체 크기를 바꿈

이유는 정렬과 패딩

멤버 각각이 struct, class 메모리에
어떻게 들어가야할지에 대한것

-Wpadded -> 패딩이 삽입될 때 경고

```
// 개념 예시 — 같은 멤버, 다른 크기
struct A {
    char c1; double d; char c2;
};  // sizeof(A) == 24

struct B {
    double d; char c1; char c2;
};  // sizeof(B) == 16
```

자세한건: https://en.wikipedia.org/wiki/Data_structure_alignment

GPU 메모리 레이아웃 — std140과 WebGPU 규약 (CPU -> GPU)

메모리 레이아웃중요한 경우

- shader uniform buffer(std140)·
- WebGPU 256B 정렬

규약을 지키지 않으면 GPU 오류 발생

- alignas와 명시적 패딩 배열
- static_assert(sizeof==256)로

컴파일 타임에 검증

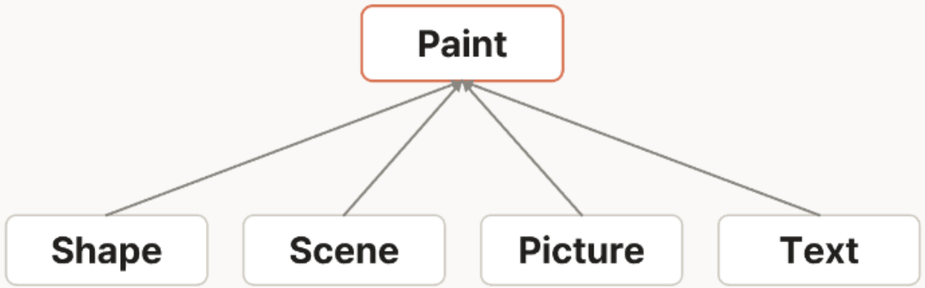
```
// gpu_engine/gl/tvgGlCommon.h:157-161
alignas(16) float nStops[4] = {};
alignas(16) float startPos[2] = {};
alignas(8)  float stopPos[2] = {};

// gpu_engine/wg/tvgWgShaderTypes.h:48
uint8_t _padding[
    256 - sizeof(WgShaderTypeMat4x4f)]{};
```

2-2

상속, Virtual, Casting

plmpl, static_cast, reinterpret_cast

Paint 상속 계층




```
// inc/thorvg.h — 공개 타입은 상속 계층
struct TVG_API Paint
{
    virtual Type type() const noexcept = 0;
protected:
    virtual ~Paint();    // 직접 delete도 봉쇄
};

struct TVG_API Shape   : Paint { /* ... */ };
struct TVG_API Scene   : Paint { /* ... */ };
struct TVG_API Picture : Paint { /* ... */ };
struct TVG_API Text    : Paint { /* ... */ };
```

Virtual? — Paint에서 virtual은 type()과 소멸자뿐

virtual 함수

- 오버라이딩 + 동적 바인딩 = 다형성

- base 클래스 포인터/참조로 호출하면 동적 바인딩을 통해 실제 객체 타입인 derived 클래스의 함수가 실행됨

```
// inc/thorvg.h — Paint 공개 메서드 (발췌)
struct TVG_API Paint
{
    Result translate(float x, float y) noexcept;
    Result rotate(float degree) noexcept;
    Result scale(float factor) noexcept;
    Result opacity(uint8_t o) noexcept;
    Result clip(Shape* clipper) noexcept;
    Result blend(BlendMethod method) noexcept;
    virtual Type type() const noexcept = 0; // 유일
protected:
    virtual ~Paint();   // …와 소멸자
};
```

PAINT_METHOD — type 스위치가 Impl 함수를 호출함

Paint 다형성 메서드는 virtual 없이 Impl로

type()으로 갈라서

to<XxxImpl>(paint)->함수 호출

(bounds·duplicate·render·update...)

장점?

- 내부에서 사용하는 함수들 사용자에게 노출 x
- 관련 함수 virtual table 접근 X <-- 단, 이점은 없을 것 (type 에서 한번 접근)

```
// src/renderer/tvgPaint.cpp:34 — 타입별 분기
#define PAINT_METHOD(ret, METHOD) \
  switch (paint->type()) { \
    case Type::Shape: \
      ret = to<ShapeImpl>(paint)->METHOD; break; \
    case Type::Scene: \
      ret = to<SceneImpl>(paint)->METHOD; break; \
    case Type::Picture: /* Text도 동일 */ \
    default: ret = {}; \
  }

// 사용 — Paint::Impl 안에서
PAINT_METHOD(ret, render(renderer, flag));
```

Impl 서브클래스로 구현되어 있는 객체들

PAINT_METHOD

- ShapeImpl이 Shape를 상속

- gen()이 준 Shape*는 사실 ShapeImpl*

```
// src/renderer/tvgShape.h — ShapeImpl 구조 (선언만)
struct ShapeImpl : Shape
{
    Paint::Impl impl;    // 뿌리 Impl 합성
    RenderShape rs;      // 지오메트리 실데이터
    uint8_t opacity;

    bool render(RenderMethod* renderer, ...);
    bool update(RenderMethod* renderer, ...);
    RenderRegion bounds();
    Result fill(Fill* f);  void resetPath();
    // …스트로크·트림 등 20여 개 — 전부 비가상
};
```

```
30  Shape* Shape::gen() noexcept
31  {
32      return new ShapeImpl;
33  }
34
```

추가) static_cast: 관계 있는 타입 사이의 변환

컴파일러가 관계를 아는 변환

int↔float, void*↔T*, Base↔Derived

동적 타입 검사는 없음 '원래 그 타입'이라는 전제 필요

to<T>()는 type() 태그를 먼저 확인

RTTI 없는 안전한 downcast

'변환'이지 '재해석'이 아님

```
// common/tvgAllocator.h:35
return static_cast<T*>(
    std::malloc(size));   // void*  T*

// renderer/tvgPaint.cpp — 태그 확인 후
case Type::Shape:
    ret = to<ShapeImpl>(paint)->METHOD;
```

추가) reinterpret_cast: 바이트의 재해석

변환이 아니라 재해석

타입읽는 방식을 교체

uint64_t -> 픽셀 2개를 한번에 읽기

```
// cpu_engine/tvgSwRle.cpp:816-817
cellsMax = reinterpret_cast<SwCell*>(
    (char*)rw.buffer + rw.bufferSize);

// cpu_engine/tvgSwRasterC.h:56·74-77
PIXEL_T* dst;    // 여기선 uint32_t* — 픽셀 버퍼
auto val64 = (uint64_t(val) << 32) | uint64_t(val);
*reinterpret_cast<uint64_t*>(dst)
    = val64;     // 32bit 픽셀 2개를 한 번에 씀
```

추가) virtual table

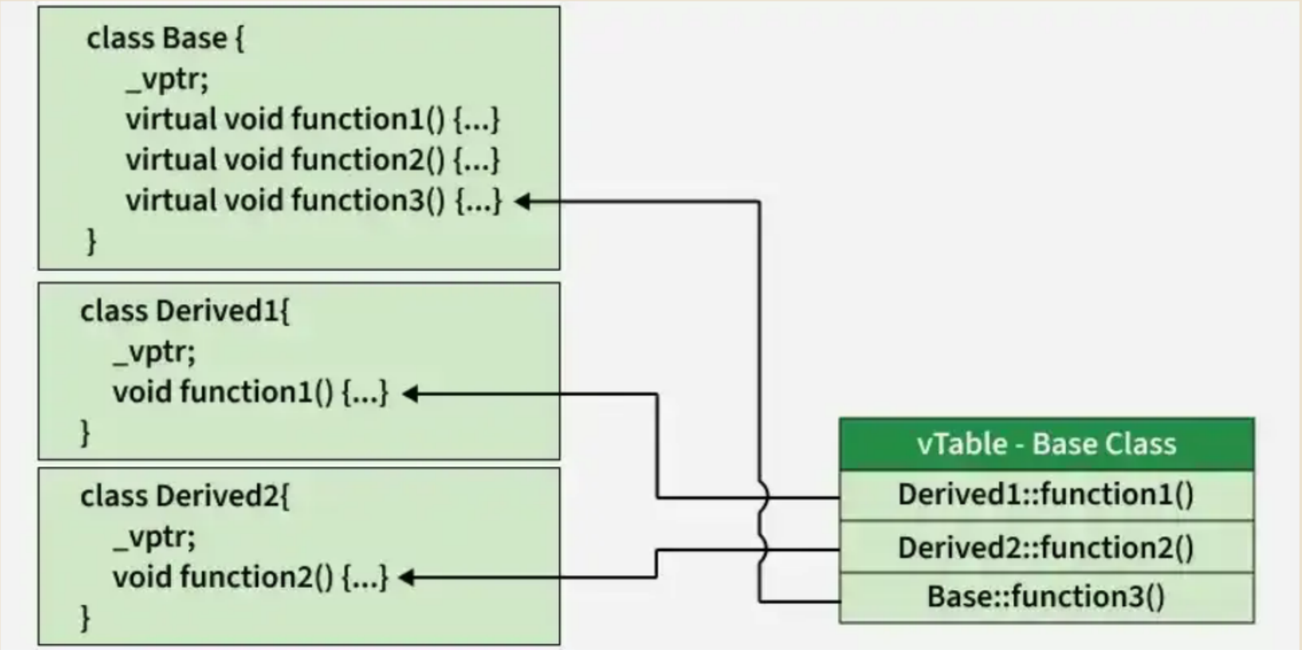
virtual 함수가 하나라도 있으면

클래스마다 vtable(함수 주소 표),

객체마다 숨은 vptr가 생김 (+8B)

호출 = vptr 로드 → 표에서 주소 로드

→ 간접 호출 — 인라인·분기 예측에 불리



추가) 가상 소멸자

Paint 에서는 Type 과 소멸자만

Virtual 존재

소멸자에 virtual 이 없을 경우

업캐스팅된 포인터에 delete 사용시

Base 의 소멸자만 호출된다

Virtual destructors

Although C++ provides a default destructor for your classes if you do not provide one yourself, it is sometimes the case that you will want to provide your own destructor (particularly if the class needs to deallocate memory). You should *always* make your destructors virtual if you're dealing with inheritance. Consider the following example:

```
#include <iostream>
class Base
{
public:
    ~Base() // note: not virtual
    {
        std::cout << "Calling ~Base()\n";
    }
};

class Derived: public Base
{
private:
    int* m_array {};

public:
    Derived(int length)
      : m_array{ new int[length] }
    {
    }

    ~Derived() // note: not virtual (your compiler may warn you about this)
    {
        std::cout << "Calling ~Derived()\n";
        delete[] m_array;
    }
};

int main()
{
    Derived* derived { new Derived(5) };
    Base* base { derived };

    delete base;

    return 0;
}
```

Note: If you compile the above example, your compiler may warn you about the non-virtual destructor (which is intentional for this example). You may need to

https://www.learncpp.com/cpp-tutorial/virtual-destructors-virtual-assignment-and-overriding-virtualization/

2-3

포인터와 소유권

raw pointer, unique_ptr, scope

init()과 term()은 쌍 — 카운터가 엔진 초기화 참조를 카운트

엔진 전역 수명은

- Initializer::init() / term() 쌍

중첩 호출은 카운터로 안전

- 마지막 term()에서 실제 종료

이때 만들어지고 정리되는 것

- LoaderMgr(포맷 로더 관리)
- TaskScheduler(스레드 풀)

= 전역 객체 수명 관리

```
// test/testInitializer.cpp:31-45
TEST_CASE("Basic initialization") {
  REQUIRE(Initializer::init() == Result::Success);
  REQUIRE(Initializer::term() == Result::Success);
}

TEST_CASE("Multiple initialization") {
  REQUIRE(Initializer::init() == Result::Success);
  REQUIRE(Initializer::init() == Result::Success);
  REQUIRE(Initializer::term() == Result::Success);
  // 카운터 — init 횟수만큼 term해야 종료
}
```

객체 생성 gen() 함수
- static factory method, Raw Pointer 반환

```
// test/testSwCanvas.cpp — "Pushing Paints" (축약)
REQUIRE(Initializer::init() == Result::Success);
{
  auto canvas = unique_ptr<SwCanvas>(SwCanvas::gen());
  uint32_t buffer[100*100] = {};
  canvas->target(buffer, 100, 100, 100,
                 ColorSpace::ARGB8888);
  canvas->add(Shape::gen());   // Paint도 gen()으로
  canvas->update();
  canvas->draw();
}  // scope 끝 — unique_ptr가 canvas 정리
REQUIRE(Initializer::term() == Result::Success);
```

ThorVG 의 주요 객체는 gen 이라는 일관성있는 함수로 생성됨
Canvas 는 사용자가 관리해야할 대상
Paint 는 Canvas 에서 관리하는 대상

Paint: gen()에서 delete까지 소유권 흐름

사용자: ref()는 반드시 unref()와 짝 — 계속 참조하려면 push 전에 ref()
ref --> Canvas/Scene 이 소멸되더라도 사용자가 후에 재사용 가능
(ex) 모든 Canvas 삭제 후 새로운 Canvas에 추가)

Canvas: gen(), 사용자가 수명 관리

```
141  SwCanvas* SwCanvas::gen(EngineOption op) noexcept
142  {
143  #ifdef THORVG_CPU_ENGINE_SUPPORT
144      if (engineInit > 0) {
145          auto renderer = new SwRenderer(TaskScheduler::threads(), op);
146          renderer->ref();
147          auto ret = new SwCanvas;
148          ret->pImpl->renderer = renderer;
149          return ret;
150      }
151  #endif
152      TVGLOG("RENDERER", "SwCanvas is not supported");
153      return nullptr;
154  }
```

```
// test/testSwCanvas.cpp — "Pushing Paints" (축약)
REQUIRE(Initializer::init() == Result::Success);
{
  auto canvas = unique_ptr<SwCanvas>(SwCanvas::gen());
  uint32_t buffer[100*100] = {};
  canvas->target(buffer, 100, 100, 100,
                 ColorSpace::ARGB8888);
  canvas->add(Shape::gen());   // Paint도 gen()으로
  canvas->update();
  canvas->draw();
}  // scope 끝 — unique_ptr가 canvas 정리
REQUIRE(Initializer::term() == Result::Success);
```

raw 포인터와 unique_ptr

raw 포인터

- delete 책임은 사용자
- Paint: 참조 관리를 통해 자동 소멸되는 Paint
- Canvas: Delete 책임은 사용자

unique_ptr

- 소유가 타입에 명시,
- scope 끝·예외에도 자동 해제
- 비용은 raw와 동일 (zero-overhead)
- 유닛 테스트 코드에서 Canvas 수명을 관리하기 위해 사용

```
SwCanvas* c = SwCanvas::gen();
// 중간에 return·throw가 있었다면?
delete c;                    // 놓치면 누수

auto c2 = unique_ptr<SwCanvas>(SwCanvas::gen());
// 어디로 벗어나든 scope 끝에 자동 delete
```

ThorVG도 처음엔 unique_ptr를 반환했음

v1.0 직전(2024-11)까지 gen()과 push()는 unique_ptr 사용

커밋 ed01ef71에서 전부 제거

—"메모리 관리는 사용자 책임으로"

```
# git show ed01ef71 (2024-11-07)
# "api: revise the spec"
# Remove the requirement for unique_ptr
# ... memory management will now be
# the responsibility of the user.

- static unique_ptr<Shape> gen();
+ static Shape* gen();

- Result push(unique_ptr<Paint> paint);
+ Result push(Paint* paint);
```

Source · source/thorvg git ed01ef71 "api: revise the spec" (2024-11-07)

추가) 팩터리의 반환형으로 unique_ptr은 관례

팩터리가 unique_ptr을 돌려주면
소유권 이전이 시그니처에 드러나고
호출자가 원하는 수명 정책으로 옮김
(그대로 쓰거나 shared_ptr로 변환)

C++ Core Guidelines도 같은 권고
"소유권 이전엔 unique_ptr"

> "A common use for std::unique_ptr is as a factory function return type for objects in a hierarchy."
>
> — Scott Meyers, Effective Modern C++

추가) scope

{ }가 만드는 scope

- 지역 객체는 진입 시 생성,
- 이탈 시 선언 역순으로 소멸

scope는 자원 수명을 묶는 단위가 됨

```
{
  scoped_lock lock(mtx);          // 잠금
  auto canvas =
      unique_ptr<SwCanvas>(SwCanvas::gen());
  // ... canvas 작업 ...
}  // 역순 소멸 — canvas 해제, lock 풀림
```

추가) RAII

RAII

- 생성시 자원 할당
- 소멸자 호출 -> 자원 해제
- Lock·파일·버퍼 관리에 사용됨
- Scope { } 를 최대한 활용하는 케이스
- unique_ptr, scoped_lock등 모던 C++ 패턴

ThorVG에서도 유효한 패턴

- ScopedLock

```
// 개념 예시 — 실제 관용구:
// tvgLottieExpressions.cpp:1568
{
    ScopedLock lock(_lockKey);
    // 생성자에서 잠금
    ...
}   // scope 끝 ⍁ 소멸자가 해제
```

추가) tvg::malloc

ThorVG 내부의 raw 할당은

tvg::malloc / calloc / realloc / free

표준 malloc 위의 얇은 래퍼

사용자가 커스텀하기 쉽게 하기 위함

(PR #3892)

```
// source/thorvg/src/common/tvgAllocator.h
//separate memory allocators
//for clean customization
template<typename T = void>
static inline T* malloc(size_t size)
{
    return static_cast<T*>(
        std::malloc(size));
}
```

03

검증 및 성능 개선과 이슈 처리를 위한 유용한 도구들

멀티 플랫폼 개발에서 사용되는 여러 도구들에 대해 소개

ThorVG.Pixel Inspector: Golden/GM

Two steps, one per ThorVG version. References are generated on demand

STEP 1 · Install thorvg@refA & Update reference PNG -> 1.0.0 minor

STEP 2 · Install thorvg@refB & Make Test set & Evaluate

UnitTest, Memory Check: valgrind, ASAN

unit_test
succeeded 2 weeks ago in 10m 49s

- Set up job
- Run actions/checkout@v4
- Install Packages
- Install wgpu_native
- Build
- Run actions/upload-artifact@v4
- Run memcheck Script(valgrind)
- Build & Run memcheck Script(ASAN)
- Post Run actions/checkout@v4
- Complete job

unit_test summary

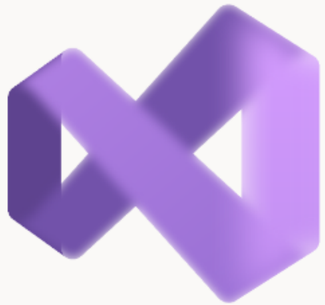
▼Valgrind output

MEMCHECK(VALGRIND) RESULT:

```
env LIBGL_ALWAYS_SOFTWARE=1 valgrind --
suppressions=/home/runner/work/thorvg/thorvg/.github/workflows/valgrind.supp --leak-check=yes ./tvgUnitTests
```

```
==4850== Memcheck, a memory error detector
==4850== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==4850== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==4850== Command: ./tvgUnitTests
==4850==
===============================================================================
All tests passed (23819 assertions in 99 test cases)

==4850==
==4850== HEAP SUMMARY:
==4850==     in use at exit: 238,645 bytes in 2,722 blocks
==4850==   total heap usage: 6,065,703 allocs, 6,062,981 frees, 4,376,128,941 bytes allocated
==4850==
==4850== 84 bytes in 1 blocks are possibly lost in loss record 2,575 of 2,593
==4850==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==4850==    by 0x4E76F87: hashbrown::raw::RawTableInner::fallible_with_capacity (in /usr/local/lib/libwgpu_nativ
==4850==    by 0x4E7790C: hashbrown::raw::RawTable<T,A>::reserve_rehash (in /usr/local/lib/libwgpu_native.so)
==4850==    by 0x4E6A7E9: <wgpu_hal::gles::egl::Instance as wgpu_hal::Instance>::init (in /usr/local/lib/libwgpu
```

Debugging: VSCode, CLion, Visual Studio, GDB, LLDB...



Profiling: VTUNE, Tracy, Instruments, CLion...

Profiling: VTUNE

Hotspots

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Flame Graph

Elapsed Time: 122.792s - 150.529s = -27.737s

CPU Time: 117.307s - 143.512s = -26.205s

Total Thread Count: Not changed, 3

Paused Time: Not changed, 0s

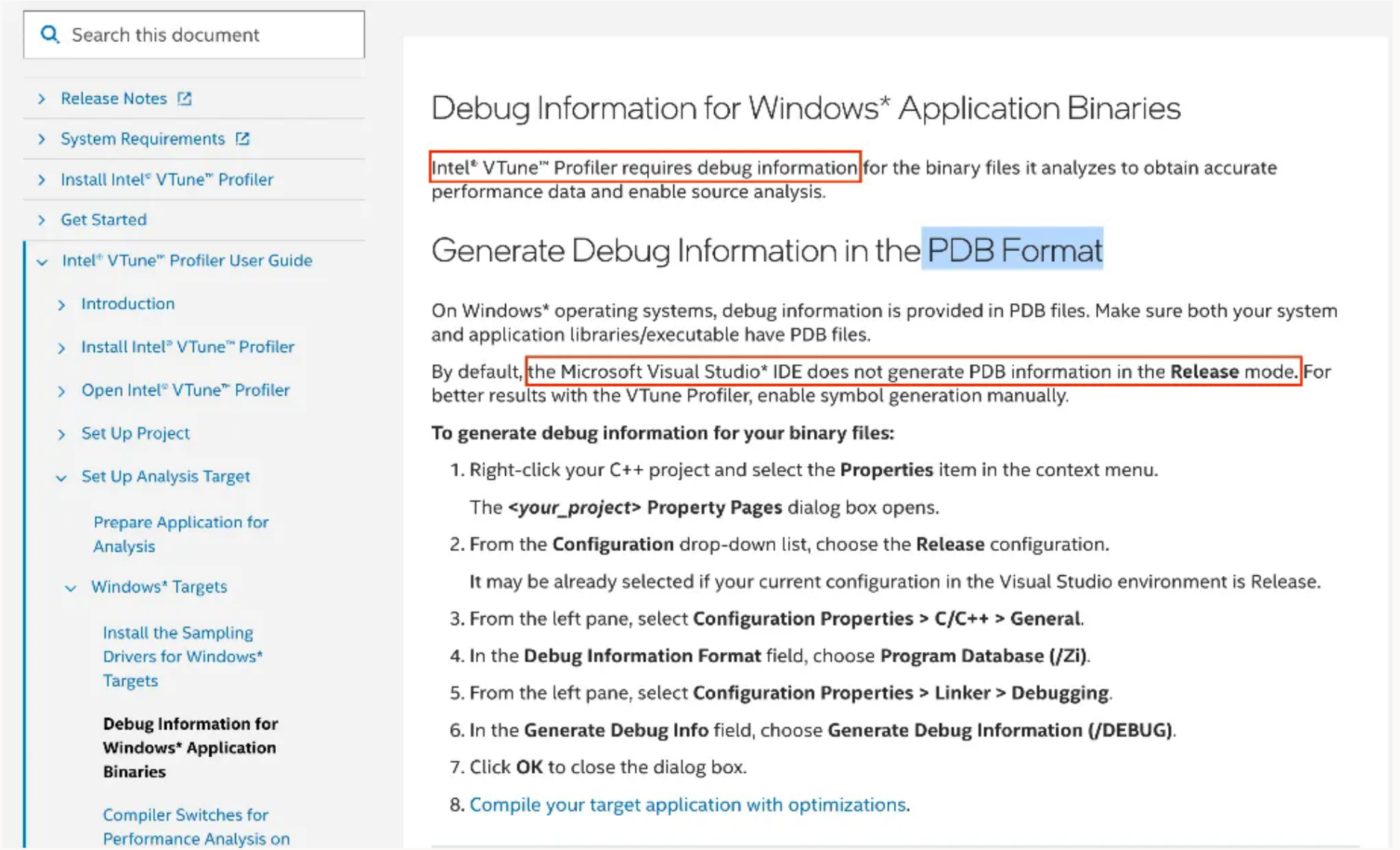
Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

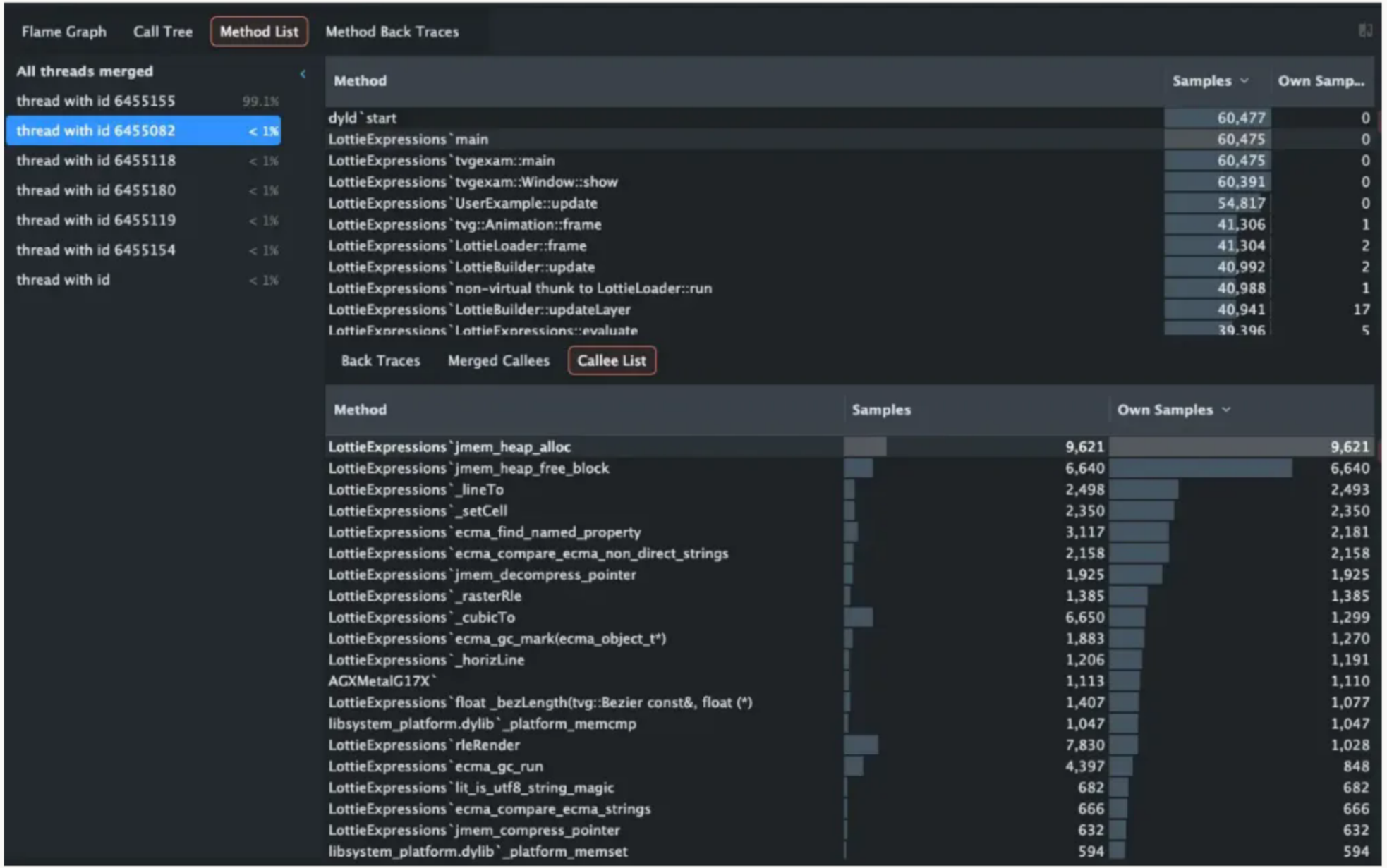
Function	Module	CPU Time	% of CPU Time
pthread_getspecific	libwinpthread-1.dll	18.523s - undefined = 18.523s	15.8% - undefined = 15.8%
pthread_spin_lock	libwinpthread-1.dll	10.902s - undefined = 10.902s	9.3% - undefined = 9.3%
jmem_heap_alloc	LottieExpressions.exe	7.703s - 17.088s = -9.385s	6.6% - 11.9% = -5.3%
RtlRestoreLastWin32Error	ntdll.dll	7.079s - undefined = 7.079s	6.0% - undefined = 6.0%
GetLastError	KERNEL32.DLL	5.412s - undefined = 5.412s	4.6% - undefined = 4.6%
[Others]	N/A*	67.687s - 126.423s = -58.736s	57.7% - 88.1% = -30.4%

*N/A is applied to non-summable metrics.

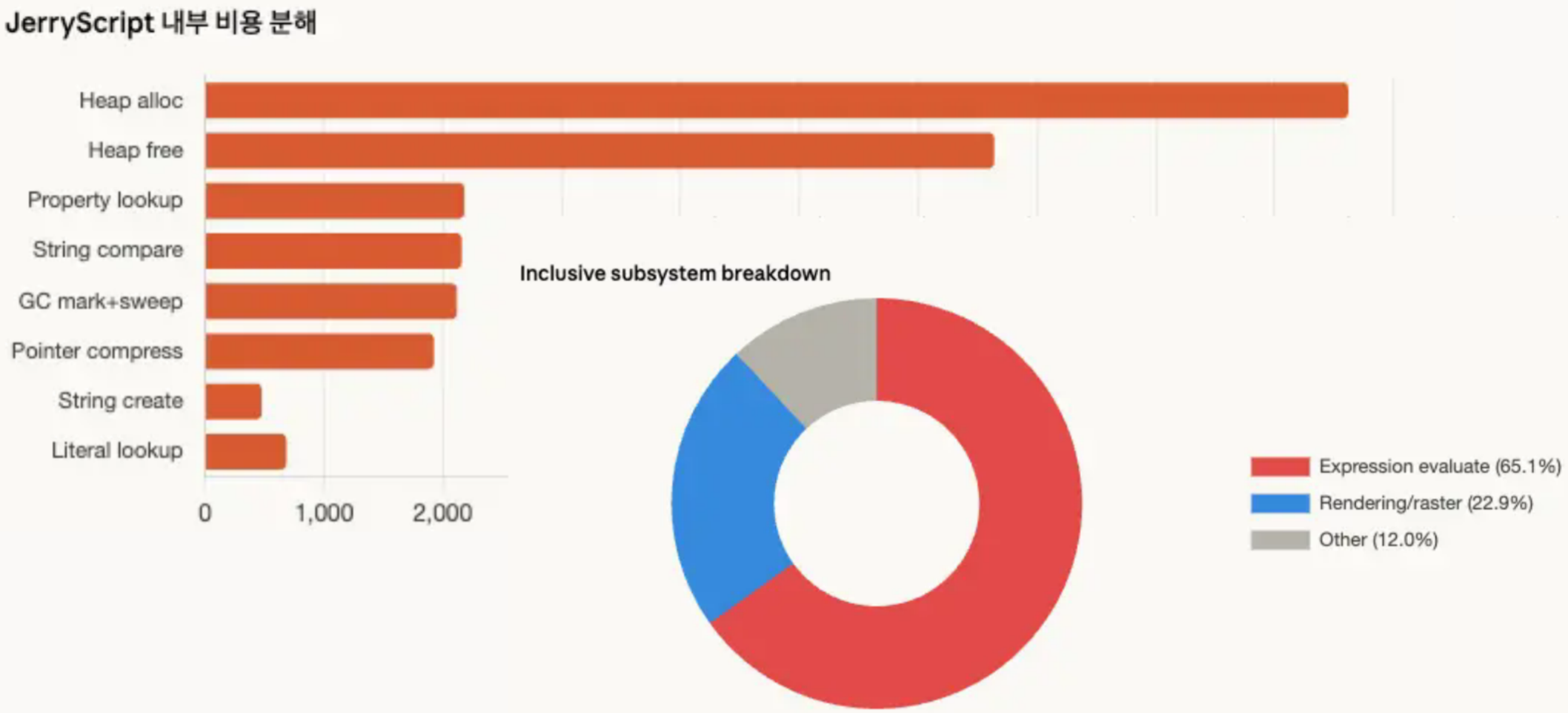
Profiling: VTUNE and PDB



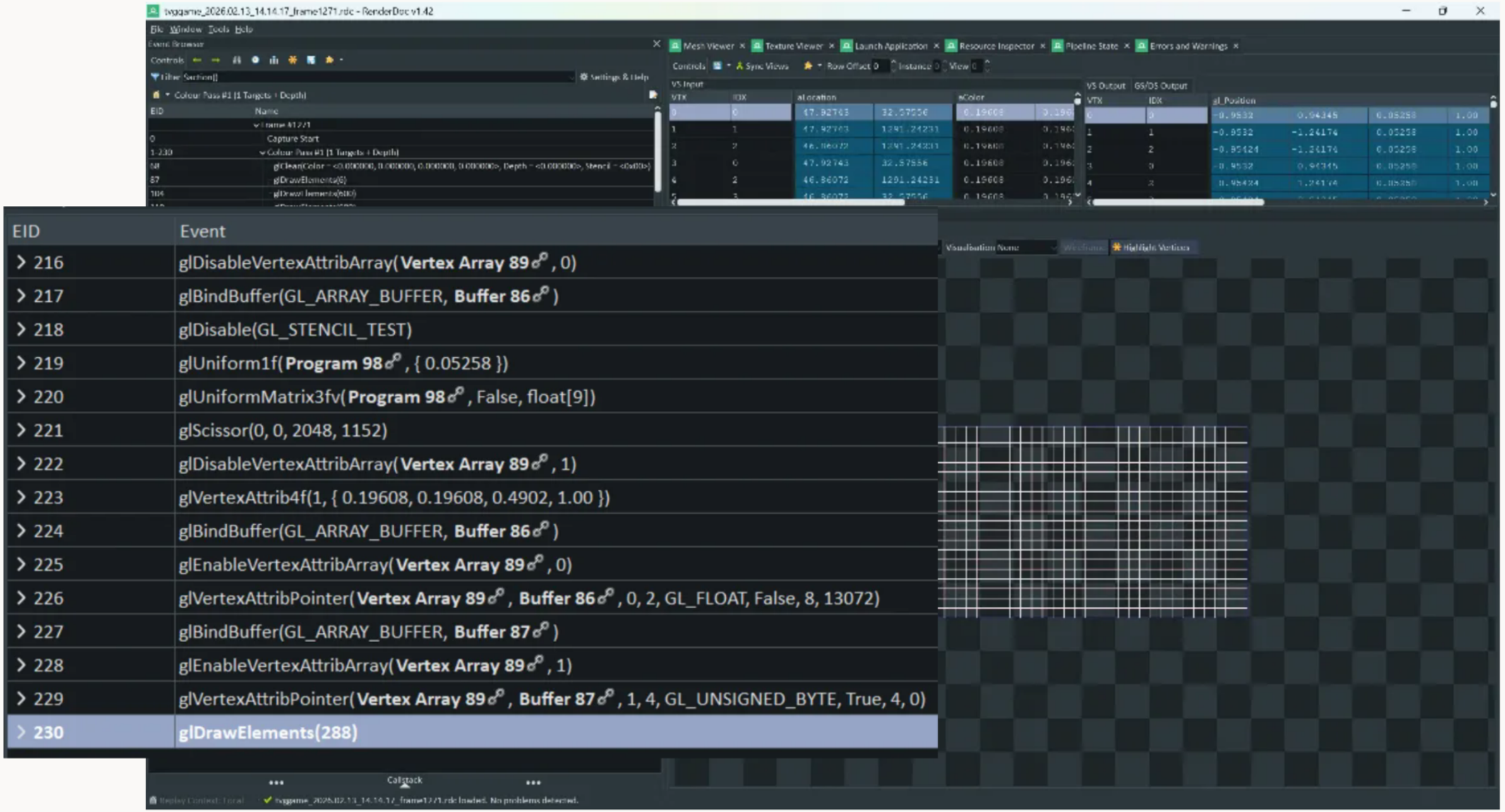
Profiling: CLion



Profiling with AI



For GPU: RenderDoc



Thank You

SoonGeon Noh
OSSCA2026/ThorVG