

# Разработка под WebAssembly: реальные грабли и примеры

Андрей Нагих (Инетра, Peers.TV)



**dev(nsk)**

# Привет!

- Работаю в Инетре, делаю PEERS.TV
- ByteFog — peer-2-peer доставка видео
- Реализовали проект по внедрению Wasm



**dev(nsk)**

# ByteFog

- C++
- EventDriven
- 900+ классов
- 95000+ строк кода
- Windows, Linux, Android, iOS, Web, Tizen



**dev(nsk)**

# О чем поговорим?

- Что такое WebAssembly?
- Какие вам понадобятся инструменты?
- Как соединить JS и Wasm
- Какие есть проблемы и как их решать
- Производительность

**dev(nsk)**

# Что такое WebAssembly?

- Бинарный формат байткода виртуальной машины JavaScript для исполнения в браузере

Сергей Рубанов. Понятно о WebAssembly —  
<https://2018.codefest.ru/lecture/1324/>



**dev(nsk)**

# Возможности WebAssembly

- быстрый синтаксический анализ
- быстрая компиляция
- компилируется пока качается
- целевая платформа для системных языков
- можно портировать нативный код в браузер
- можно писать ассемблер руками (.wast)

**dev(nsk)**

# Ограничения WebAssembly

- Не ускорит ваш JS сразу и даром
- Нет сборщика мусора
- Только 32-bit доступ к памяти
- **Не может больше, чем может JavaScript**

**dev(nsk)**

# Чем заменяем?

C++	Wasm + JS
FileSystem	LocalStorage, Cookie, IndexedDB
Network	XHR, fetch, WebSocket
Random	Math.random
printf	console.log
Асинхронщина	Poll + setTimeout
3D графика	Canvas, WebGL

**dev(nsk)**





# ИНСТРУМЕНТЫ

# Чем компилировать?

- **Binaryen** (*голый компилятор и утилиты*)
  - C/C++ Source  $\Rightarrow$  asm.js  $\Rightarrow$  asm2wasm  $\Rightarrow$  WebAssembly
  - C/C++ Source  $\Rightarrow$  Wasm LLVM backend  $\Rightarrow$  s2wasm  $\Rightarrow$  WebAssembly
  - assembler, disassembler, interpreter, tests, js polyfill, etc.
- **Emscripten** (*browserify для C++*)
  - Воссоздаст среду для C++ приложения
  - Пробросит объекты из C++ в JS
  - Даст вызвать JS код из C++

**dev(nsk)**

# Как компилировать?

Скомпилировать 1 файл:

```
// build.bat
```

```
call D:\_sys\emsdk\emsdk_env.bat
```

```
em++.bat hello.cpp -s WASM=1 -o hello.js -g4 --bind
```

**dev(nsk)**

# Чепез CMake

```
//CMakeLists.txt
```

```
cmake_minimum_required(VERSION 2.8)
```

```
project(my_project)
```

```
add_executable(my_target hello.cpp)
```

```
set_target_properties(my_target PROPERTIES SUFFIX ".js")
```

```
LINK_FLAGS " -s WASM=1 -g4 --bind "
```

**dev(nsk)**

# Через CMake

```
//build.sh
```

```
cmake . -DCMAKE_TOOLCHAIN_FILE=  
${EMSCRIPTEN}/cmake/Modules/Platform/Emscripten.cmake
```

Toolchain входит в поставку Emscripten

```
$EMSCRIPTEN = ../emsdk/emscripten/x.xx.xx
```

**dev(nsk)**

# SourceMap

- Только в FireFox
- `--sourcemap-base=http://localhost/`
- Доступ к исходникам по http
- Проблема с «:»
- Абсолютные пути не годятся

**dev(nsk)**

# Как связать два мира?

- `ccall + cwrap` (plain C functions)
- WebIDL Binder (C++ functions, classes)
- **Embind** (C++  $\longleftrightarrow$  JS)

**dev(nsk)**



**КАК ИСПОЛЬЗОВАТЬ EMBIND?**

**dev(nsk)**



# Функцию C++ в JS

JS

```
var result = Module.my_function(...)
```

Call

C++

```
float my_function(...) {...; return float_result;}
```

Definition

**dev(nsk)**

# ФУНКЦИЮ C++ В JS

```
#include <emscripten/bind.h>
```

```
using namespace emscripten;
```

```
float my_function(...) {  
    return ...;  
}
```

```
EMSCRIPTEN_BINDINGS(my_module) {  
    function("my_function", &my_function);  
}
```

**dev(nsk)**

# Функцию C++ в JS

Используем:

```
var number = Module.my_function(...)
```

**dev(nsk)**

# Класс C++ в JS

JS

```
var instance = new MyClass(...)
```

Call

C++

```
class MyClass {...};
```

Definition

**new(nsk)**

# Класс C++ в JS

```
class MyClass {  
  public:  
    MyClass(int);  
    void my_method();  
};
```

**dev(nsk)**

# Класс C++ в JS

```
EMSCRIPTEN_BINDINGS(my_class_example) {  
    class_<MyClass>("MyClass")  
        .constructor<int>()  
        .function("my_method", &MyClass::my_method)  
}
```

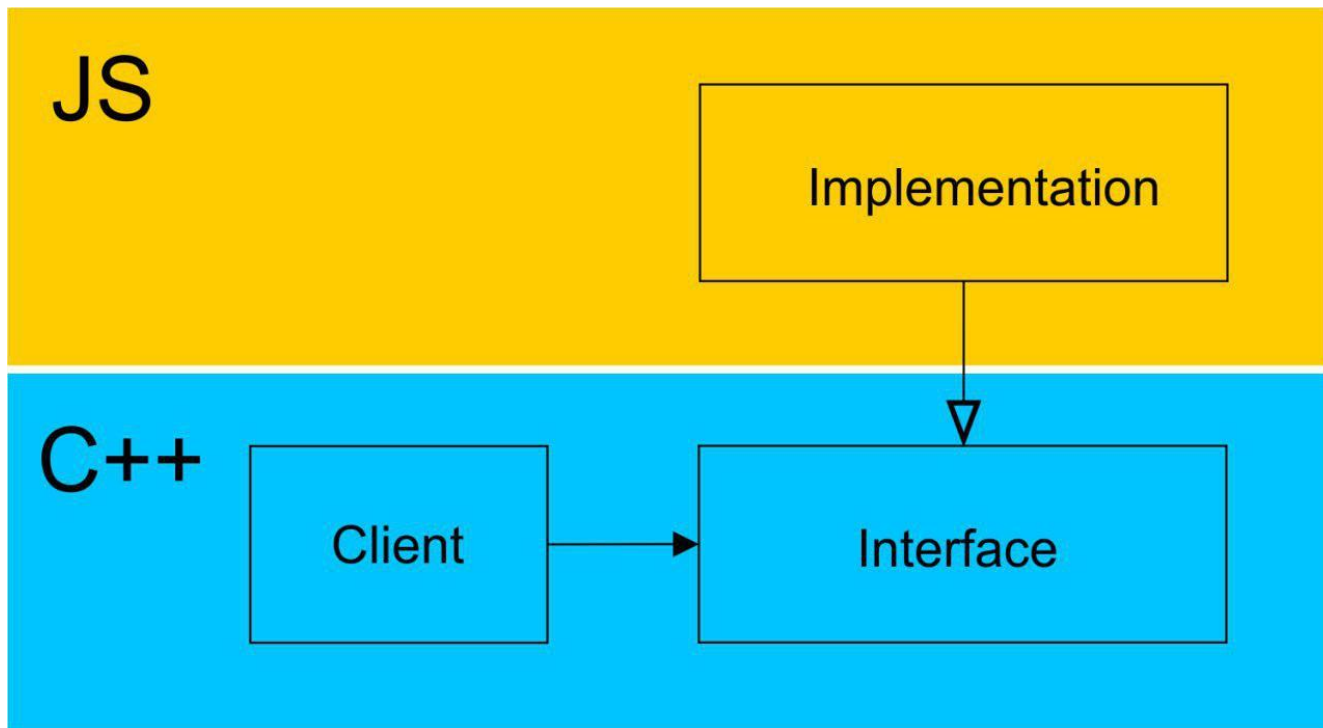
**dev(nsk)**

# Класс C++ в JS

```
var instance = new Module.MyClass(10);  
instance.my_method();  
instance.delete();
```

**dev(nsk)**

# Интерфейс C++ в JS



dev(nsk)



# Интерфейс C++ в JS

```
class Interface {  
    public:  
    virtual void my_method(int& number) = 0;  
};
```

**dev(nsk)**

# Интерфейс C++ в JS

```
class InterfaceWrapper : public wrapper<Interface> {  
    EMSCRIPTEN_WRAPPER(InterfaceWrapper);  
    void my_method(int& number) {  
        return call<void>("my_method", number);  
    }  
};
```

**dev(nsk)**

# Интерфейс C++ в JS

```
EMSCRIPTEN_BINDINGS(interface) {  
    class_<Interface>("Interface")  
        .function("my_method",  
                  &Interface::my_method,  
                  pure_virtual())  
        .allow_subclass<InterfaceWrapper>("InterfaceWrapper");  
}
```

**dev(nsk)**

# Интерфейс C++ в JS (extend)

```
var DerivedClass =  
Module.Interface.extend("Interface", {  
    my_method: function(number) { ... },  
});
```

```
var instance = new DerivedClass();  
instance.my_method(10);
```

**dev(nsk)**

# Интерфейс C++ в JS (implement)

```
var x = {  
    my_method: function(number) { ... }  
};  
var instance = Module.Interface.implement(x);  
instance.my_method(10);
```

**dev(nsk)**

# ВЫЗОВ JS ИЗ C++

```
using namespace emscripten;
```

```
void log_message(const char* message) {  
    val console = val::global("console");  
    console.call<void>("log", val(message) );  
}
```

**dev(nsk)**

An illustration showing a black leather boot with a brown sole stepping onto the head of a black metal rake. The rake is lying on a light-colored surface, and several other rake heads and wooden handles are scattered around it. The boot is positioned in the center-right of the frame, with its foot firmly on the rake head. The rake head has several sharp, curved tines. The handles are made of light brown wood and are scattered in various directions around the central boot and rake head. The overall style is a simple, clean illustration with bold outlines and flat colors.

**НАШИ ГРАБЛИ**

# Биндим правильно

- Совпадает имя
- Совпадают типы
- Корректный синтаксис Embind
- Реализация JS части

**dev(nsk)**



# Extend

- **extend** это не **extends** из ES2015
- **extend** скрывает ошибки биндинга до момента вызова метода, в отличие от **implement**, который стреляет сразу

**dev(nsk)**

# Extend

- если переопределяем **\_\_constructor**, или **\_\_destructor**, то не забыть вызвать их для **this.\_\_parent**

```
__construct: function() {  
  this.__parent.__construct.call(this);  
},  
__destruct: function() {  
  this.__parent.__destruct.call(this);  
},
```

**dev(nsk)**

# Extend

- **extend** не совместим с классами ES2015
- Внимание: костыль!

```
function enumeratePrototype(class2expose) {  
  Object.getOwnPropertyNames(class2expose.prototype).forEach(  
    propName => Object.defineProperty(  
      class2expose.prototype, propName, {enumerable: true}  
    )  
  );  
}
```

**dev(nsk)**

# Передача объекта через границу

- **Предусловие:** абстрактный класс
- **Задача:** передать объект внутрь JS функции
- **Проблема:** при выходе из функции объект за указателем разрушается
- **Решение:** делаем `ptr.clone()`
- **Важно:** не забыть `ptr.delete()`

**dev(nsk)**

# Биндинг примитивных типов

- Примитивные типы по значению связываются автоматически

C++ type	JavaScript type
void	Undefined
bool	Boolean
char, short, int, long, float, double <i>signed, unsigned</i>	Number
std::string	String, ArrayBuffer, Uint8Array
std::wstring	String (UTF-16)

**dev(nsk)**

# Указатель на примитивный тип

- Никак. :-)
- Внимание: костыль!

```
void my_func(size_t data) {  
    uint8_t* real_data = (uint8_t*) data;  
    ...  
}
```

**dev(nsk)**

# Записать в память WASM

```
var newData = new Uint8Array(...);
```

```
var size = newData.byteLength;
```

```
var ptr = Module._malloc(size);
```

```
var dataHeap = new Uint8Array(Module.buffer, ptr, size);
```

```
dataHeap.set(newData);
```

**dev(nsk)**



# AdBlock

- 3<sup>rd</sup>-party .wasm блокируется списком RU AdList  
`.wasm|$third-party,xmlhttprequest,domain=~lite.boxshot.com`
- Подключается по умолчанию в России
- AdBlock, AdBlock Plus, uBlock Origin
- **Решение:** хранить на своем домене
- **Решение:** переименовать .wasm файл

<https://forums.lanik.us/viewforum.php?f=102>

**dev(nsk)**



# Производительность

## Фильтры для обработки изображения

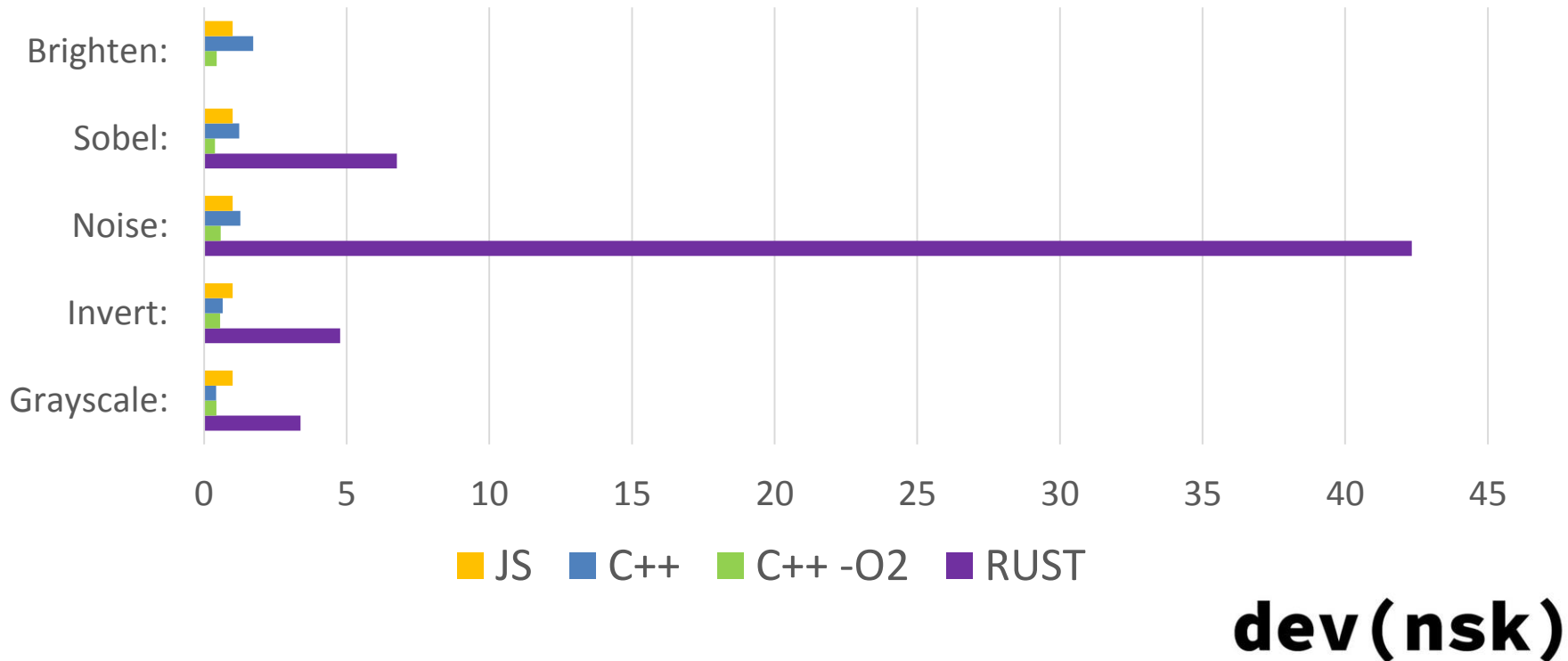
- Rust ([https://github.com/koteld/ll\\_images\\_processing](https://github.com/koteld/ll_images_processing))
- C++ ([https://github.com/andragn/wasm\\_cpp\\_bench](https://github.com/andragn/wasm_cpp_bench))
- JS

Chrome 65.0.3325.181 (64-bit), Core i5-4690, 24gb ram

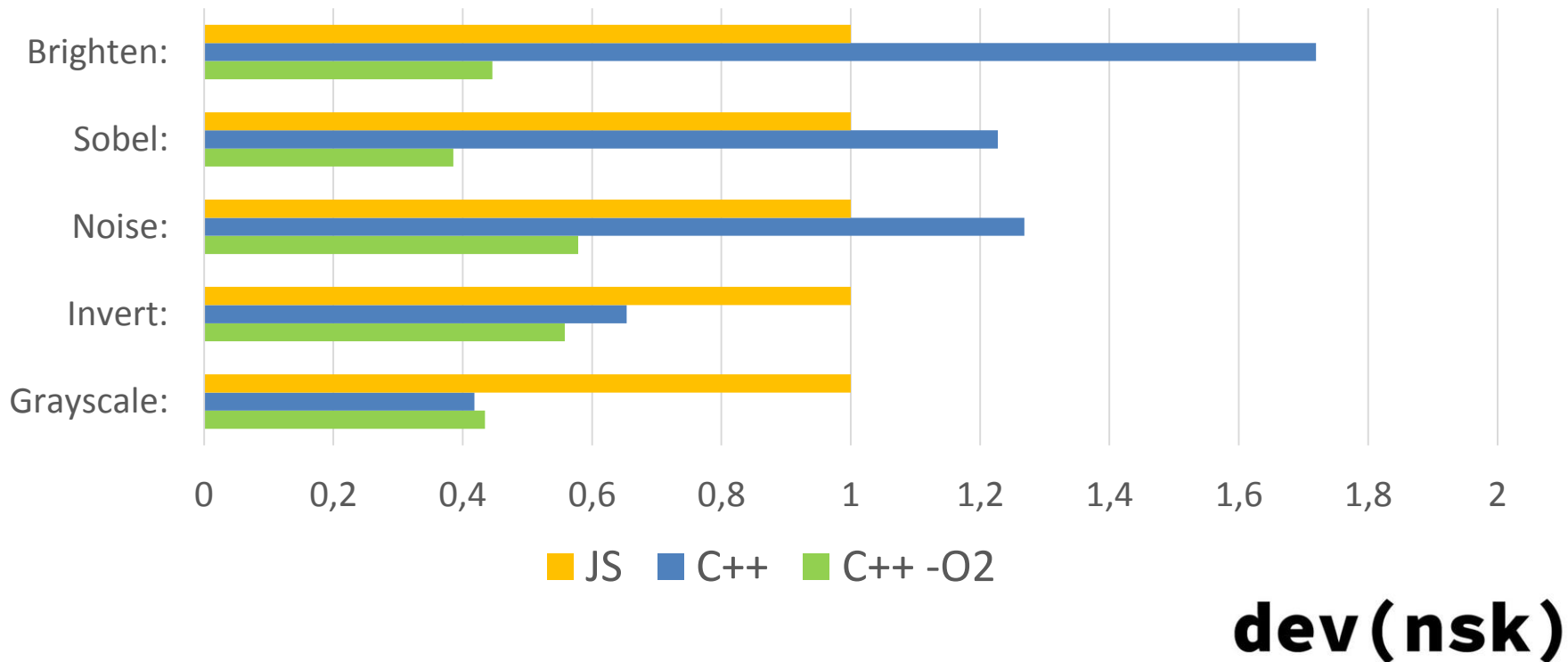
Укрощаем WebAssembly: обработка изображений в браузере  
<https://events.yandex.ru/lib/talks/5505/>

**dev(nsk)**

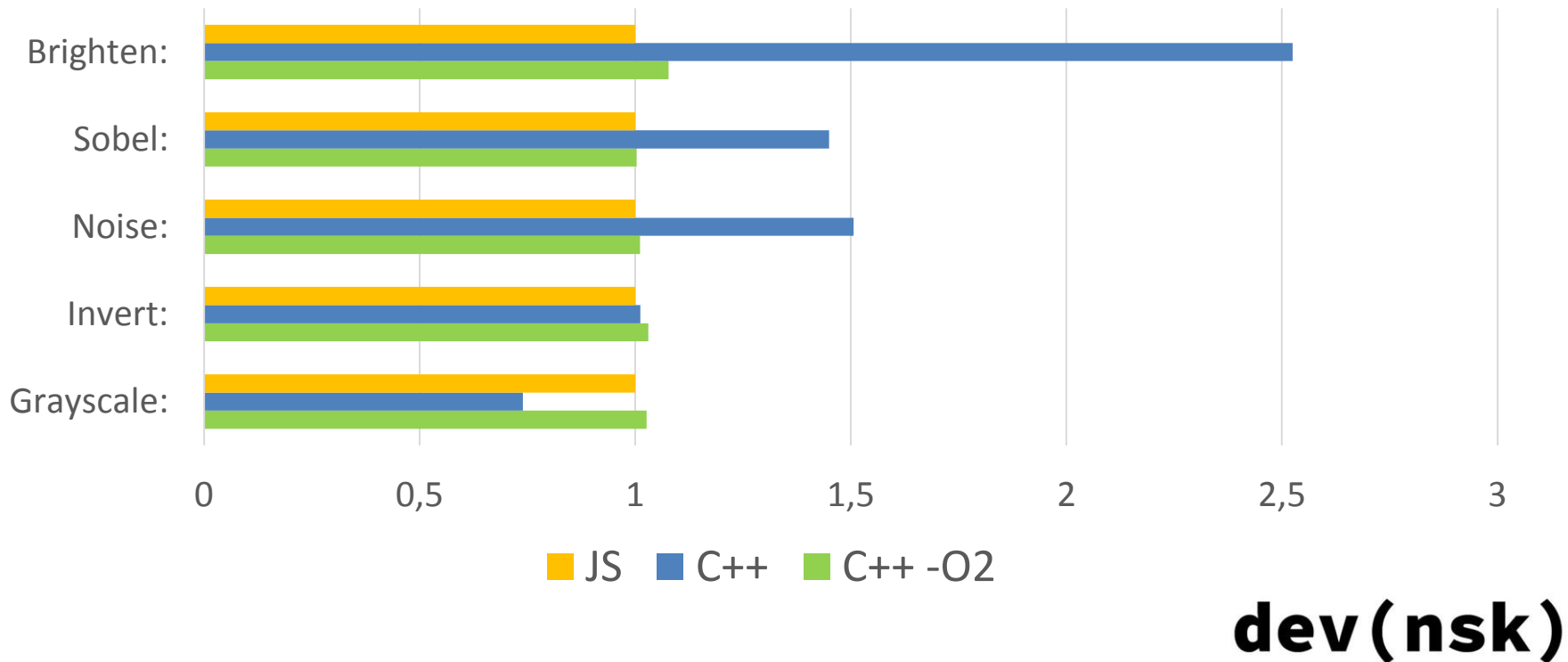
# 1280x720



# 1280x720 (no RUST)



# 5472x3078



# Выводы

- Технологию уже можно использовать в бою
- Портировать большое приложение — реально
- Скорость — не хуже JS
- Берите Emscripten и Embind
- Начинайте с небольших прототипов
- Делайте минимальный тест для проблем

**dev(nsk)**

# Спасибо! Вопросы?

Андрей Нагих

Инетра, Peers.TV

[andrey@nagih.ru](mailto:andrey@nagih.ru)



<https://goo.gl/mhMkeb>

**dev(nsk)**

# Доступные языки

- C/C++
- Rust
- Lua, Brainfuck — интерпретаторы в WASM
- Kotlin/Native, Go — заявили, надо следить
- Wah — ассемблер с человеческим лицом
- Все остальное — в стадии эксперимента

<https://github.com/appcypher/awesome-wasm-langs>

<https://stackoverflow.com/a/47483989>

**dev(nsk)**

# Smart Pointers

Создаем объект сразу как shared\_ptr

```
EMSCRIPTEN_BINDINGS(construct_as_smart_pointer) {  
    class_<MyClass>("MyClass")  
        .smart_ptr_constructor(  
            "MyClass",  
            &std::make_shared<MyClass>  
        );  
}
```

**dev(nsk)**



# Smart Pointers

Возможность передавать как shared\_ptr

```
EMSCRIPTEN_BINDINGS(pass_as_smart_pointer) {  
    class_<MyClass>("MyClass")  
        .constructor<>()  
        .smart_ptr<std::shared_ptr<MyClass>>("MyClass");  
}
```

**dev(nsk)**

# Интерфейс + SmartPtr

Интерфейс, реализацию которого передаем в виде умного указателя

```
EMSCRIPTEN_BINDINGS(MyClass) {  
    class_<MyClass>("MyClass")  
        .smart_ptr<std::shared_ptr<MyClass>>("shared_ptr<MyClass>")  
        .function("my_method", &MyClass::My_method, pure_virtual())  
        .allow_subclass<MyClassWrapper, std::shared_ptr<MyClassWrapper>>  
            ("MyClassWrapper", "MyClassWrapperPtr");  
}
```

**dev(nsk)**

# extend vs. implement

## extend

- Можно расширять интерфейс
- Скрывает ошибки биндинга
- На выходе: конструктор (класс)
- Запись можно разделить

## implement

- Нельзя расширять интерфейс
- Сразу проверяет биндинг
- На выходе: объект
- Запись короче

**dev(nsk)**