🔧 Как собрать antenna_module.cpp из частей

Файл `antenna_module.cpp` был разделён на 4 части для удобства. Чтобы собрать финальный файл:

Вариант 1: Вручную

Объедините файлы в следующем порядке:

1. **antenna_module-part1.cpp** (Конструктор + Initialize + вспомогательные методы)
2. **antenna_module-part2.cpp** (CreateKernels + FFT планы)
3. **antenna_module-part3.cpp** (ProcessNew + ProcessSingleBatch)
4. **antenna_module-part4.cpp** (ProcessMultiBatch + ProcessBatch + FindMaximaOnGPU)

Удалите строки вида `// КОНЕЦ ЧАСТИ X/4` между частями.

Вариант 2: Команда cat

```
cat antenna_module-part1.cpp
antenna_module-part2.cpp
antenna_module-part3.cpp
antenna_module-part4.cpp
> modules/antenna/src/antenna_module.cpp
```

Финальная структура файла:

```
#include "antenna_module.hpp"
#include "common/logger.hpp"
#include <fstream>
#include <sstream>
#include <stdexcept>
#include <cmath>
#include <iostream>

namespace drv_gpu_lib {
namespace antenna {

// ═══ ЧАСТЬ 1: Конструктор, Initialize, вспомогательные методы ═══
AntennaModule::AntennaModule(...) { ... }
~AntennaModule() { ... }
void Initialize() { ... }
void Cleanup() { ... }
size_t CalculateNFFT(...) { ... }
size_t EstimateRequiredMemory() { ... }
```

```
bool CheckAvailableMemory(...) { ... }
size_t CalculateBatchSize(...) { ... }
void UpdateParams(...) { ... }

// ═══ ЧАСТЬ 2: Kernels и FFT планы ═══
std::string LoadKernelSource(...) { ... }
void CreateKernels() { ... }
void ReleaseKernels() { ... }
void CreateOrReuseFFTPlan() { ... }
void CreateBatchFFTPlan(...) { ... }
void ReleaseFFTPlan() { ... }

// ═══ ЧАСТЬ 3: ProcessNew и ProcessSingleBatch ═══
AntennaFFTResult ProcessNew(SVM/GPU/cl_mem) { ... }
AntennaFFTResult ProcessSingleBatch(...) { ... }

// ═══ ЧАСТЬ 4: Batch processing ═══
AntennaFFTResult ProcessMultiBatch(...) { ... }
std::vector<BeamFFTResult> ProcessBatch(...) { ... }
std::vector<BeamFFTResult> FindMaximaOnGPU(...) { ... }

} // namespace antenna
} // namespace drv_gpu_lib
```

✅ Проверка

После сборки файл должен компилироваться без ошибок:

```
cd build
cmake ..
make antenna_module
```

📁 Структура модуля Antenna

```
modules/antenna/
├── CMakeLists.txt # Конфигурация сборки
├── include/
│ ├── antenna_module.hpp # Главный класс AntennaModule
│ ├── antenna_params.hpp # Параметры (AntennaParams, BatchConfig)
│ └── antenna_result.hpp # Результаты (FFTMaxValue, BeamFFTResult,
AntennaFFTResult)
├── src/
│ └── antenna_module.cpp # Реализация (собрать из 4 частей!)
└── kernels/
└── antenna_fft.cl # OpenCL kernels (padding, post, reduction)
```

🎯 Главная функция: ProcessNew()

Автоматический выбор стратегии обработки:

```
// Оценить память
size_t required = EstimateRequiredMemory();
bool fits = CheckAvailableMemory(required);

if (fits) {
// SINGLE-BATCH: обработать все лучи за раз
return ProcessSingleBatch(input_signal);
} else {
// MULTI-BATCH: разбить на батчи
return ProcessMultiBatch(input_signal);
}
```

SINGLE-BATCH режим:

- Создаёт буферы для всех лучей
- Создаёт FFT план для beam_count лучей
- Обрабатывает всё за один проход
- **Кэширует буферы и план для повторного использования! ♻️**

MULTI-BATCH режим:

- Рассчитывает размер батча (22% от total_beams)
- Создаёт буферы для максимального батча
- Создаёт FFT план для батча
- Обрабатывает лучи батчами с beam_offset
- **Кэширует буферы и план для повторного использования! ♻️**

🔑 Ключевые особенности

1. Batch Offset Support

```
// padding_kernel получает beam_offset для чтения нужных лучей!
clSetKernelArg(padding_kernel_, 2, sizeof(cl_uint), &beam_offset);

// Пример:
// - beam_offset = 0 → лучи 0-9 (первый батч)
// - beam_offset = 10 → лучи 10-19 (второй батч)
```

2. Кэширование ресурсов

```
// Буферы создаются один раз
if (!buffer_fft_input_ || buffer_fft_input_->GetSize() != required_size) {
buffer_fft_input_ = mem_mgr.CreateGPUBuffer<std::complex<float>>(required_size);
}

// FFT план создаётся один раз
if (main_plan_handle_ == 0) {
CreateOrReuseFFTPlan(); // Создать
```

```
} else {
// Переиспользовать существующий! ♻
}
```

3. SVM Support (Zero-Copy)

```
// Данные создаются на CPU, доступны на GPU без копирования
auto signal = mem_mgr.CreateSVMBuffer<std::complex<float>>(size);
std::complex<float>* data = signal->GetHostPtr();

// Заполнить на CPU
for (size_t i = 0; i < size; ++i) {
data[i] = /* your signal */;
}

// Обработать на GPU (нулевое копирование!)
AntennaFFTResult result = antenna->ProcessNew(signal);
```

🚀 Использование

```
#include "antenna/antenna_module.hpp"

// 1. Инициализация DrvGPU
DrvGPU gpu(BackendType::OPENCL, 0);
gpu.Initialize();

// 2. Создать модуль
using namespace drv_gpu_lib::antenna;

AntennaParams params(5, 1000, 512, 3); // 5 beams, 1000 points, 512 FFT out, 3 peaks
auto antenna = std::make_shared<AntennaModule>(&gpu.GetBackend(), params);
antenna->Initialize();

// 3. Создать SVM буфер (zero-copy!)
auto& mem_mgr = gpu.GetBackend().GetMemoryManager();
auto signal = mem_mgr.CreateSVMBuffer<std::complex<float>>(5 * 1000);

// 4. Заполнить данными на CPU
std::complex<float>* data = signal->GetHostPtr();
// ... fill data ...

// 5. Обработать (автоматический выбор стратегии!)
AntennaFFTResult result = antenna->ProcessNew(signal);

// 6. Получить результаты
for (size_t beam = 0; beam < result.results.size(); ++beam) {
auto& beam_result = result.results[beam];
for (auto& peak : beam_result.max_values) {
std::cout << "Peak: index=" << peak.index_point
<< " amp=" << peak.amplitude << "\n";
}
}
```

Pipeline обработки

```
ProcessNew()
│
├─ EstimateRequiredMemory()
├─ CheckAvailableMemory()
│
├─ IF памяти хватает:
│  └─ ProcessSingleBatch()
│  ├─ CreateOrReuseFFTPlan() (кэш!)
│  ├─ Создать буферы (кэш!)
│  ├─ Padding kernel (offset=0)
│  ├─ FFT transform
│  ├─ Post kernel (magnitude + select)
│  └─ Reduction kernel (топ-N максимумов)
│
└─ ELSE (памяти не хватает):
└─ ProcessMultiBatch()
├─ CalculateBatchSize()
├─ CreateBatchFFTPlan() (кэш!)
├─ Создать буферы (кэш!)
│
└─ FOR каждый батч:
└─ ProcessBatch(start_beam, num_beams)
├─ Padding kernel (offset=start_beam!) ← ВАЖНО!
├─ FFT transform
├─ Post kernel
└─ Reduction kernel
```

Все части содержат полный рабочий код - просто объедините их! ✅

См. также:

- **ANTENNA-COMPLETE-DOCS.md** - подробная документация по модулю
- **ANTENNA-TESTS-README.md** - инструкция по запуску тестов

ТЕСТЫ

Созданы 2 теста:

Тест 1: CPU -> SVM -> GPU (Zero-Copy)

- Генерация синусоидального сигнала на CPU
- Создание SVM буфера (нулевое копирование!)
- Обработка FFT через ProcessNew(svm_buffer)
- Получение результатов

Тест 2: CPU -> External OpenCL -> Import to DrvGPU -> GPU

- Генерация сигнала на CPU
- Создание внешнего OpenCL буфера

- **Вариант A:** Передача cl_mem напрямую
- **Вариант B:** Импорт через GPUBuffer wrapper
- Обработка FFT и получение результатов

Параметры сигнала:

10 лучей (антенн)
1500 точек на луч
Синусоиды с растущей частотой: w_i = 500 * (1.5^i)
Сдвиг фазы: 5° на луч

Запуск тестов:

cd build
cmake .. -DBUILD_TESTS=ON
make test_antenna_module
./bin/test_antenna_module

См. **ANTENNA-TESTS-README.md** для деталей!