1. SM_2D API [illegible]

×[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]×

1.1. [illegible]

SM_2D API [illegible] [illegible] [illegible] [illegible] [illegible]! [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] SM_2D (Simplified Monte Carlo 2D) [illegible] [illegible] [illegible] [illegible].

×[illegible] [illegible]×
SM_2D[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] - [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

×[illegible] [illegible]×
×API [illegible]×

- ×[illegible] (Structure)×: [illegible] [illegible] [illegible] C++ [illegible] [illegible]
- ×[illegible] (.hpp)×: [illegible]/[illegible] [illegible] [illegible] [illegible]
- ×[illegible] [illegible] (Binary search)×: [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
- ×[illegible] [illegible] (LUT)×: [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
- ×[illegible] [illegible] (Phase space)×: [illegible] [illegible] [illegible] [illegible] [illegible]
- ×[illegible]×: [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

1.2. [illegible] API

[illegible] API[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

1.2.1. EnergyGrid

[illegible] [illegible] [illegible]([illegible] [illegible] [illegible] [illegible])[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

```
struct EnergyGrid {
    const int N_E;              // [illegible]
    const float E_min;          // [illegible] [MeV]
    const float E_max;          // [illegible] [MeV]
    std::vector<float> edges;   // [illegible] (N_E + 1[illegible])
    std::vector<float> rep;     // [illegible] (N_E[illegible])

    // [illegible]: [illegible]
    EnergyGrid(float E_min, float E_max, int N_E);

    // [illegible] ([illegible])
    int FindBin(float E) const;

    // [illegible]
    float GetRepEnergy(int bin) const;
};
```

×C++ [illegible] [illegible]×
×C++ [illegible] [illegible]×

- `const int N_E`: [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
- `std::vector<float>`: [illegible] [illegible] [illegible] [illegible] [illegible] (Python [illegible] [illegible])
- `edges`: N[illegible] [illegible] N+1[illegible] [illegible] [illegible] N_E + 1[illegible] [illegible] [illegible]
  - [illegible]: 3[illegible] [illegible] 4[illegible] [illegible] [illegible]: [0, 10, 20, 30]
  - [illegible] 0: [0, 10), [illegible] 1: [10, 20), [illegible] 2: [20, 30]

- `::const`: □□□ □□□□ □□ □□□ (□□ □□)

1.2.1.1. □□□ □□□ □□□□ (□□ □□)

×□: □□□ □□□ □□ □□×

| □□ □□ | □ | □□ | |—|—|—| | E_min | 0.1 MeV | □□ □□□ | | E_max | 250 MeV | □□ □□□ | | □ □ | 256 | □□ □ □□ | | □ 0 □□ □□□ | 0.13 MeV | □□ □□ | | □ 242 □□ □□□ | 145.0 MeV | E=150.0 MeV□ □□ | | FindBin(150.0) | 242 | □ □□□ | | GetRepEnergy(242) | 145.0 MeV | □□ □□□ |

1.2.1.2. □□□

```
EnergyGrid(float E_min, float E_max, int N_E);
```

×□□:× □□ □□□□ □□□ □□□□ □□□□□

×□□□□:×

- `E_min`: □□ □□□ MeV (□: 0.1)
- `E_max`: □□ □□□ MeV (□: 250.0)
- `N_E`: □ □ (□: 256)

×□□:× EnergyGrid □□

×□□:×

```
// 0.1□□ 250 MeV□□ 256□ □□□ □□□ □□
EnergyGrid e_grid(0.1f, 250.0f, 256);

// □□□ □□:
// e_grid.N_E = 256
// e_grid.E_min = 0.1
// e_grid.E_max = 250.0
// e_grid.edges.size() = 257 (256□ □ + 1□ □□ □□)
// e_grid.rep.size() = 256 (□□ □□□ □□□)
```

1.2.1.3. FindBin

```
int FindBin(float E) const;
```

×□□:× □□□ □□□ □□ □□□□ □□□ □□ □□□□

×□□ □□:× □□ □□ □□ (O(log N) □□□)

×□□□□:×

- `E`: □□ □□□ □ MeV

×□□:× □ □□□ (0□□ N_E-1)

×□□:×

```
EnergyGrid e_grid(0.1f, 250.0f, 256);

// 150 MeV □□□□ □□ □ □□
int bin = e_grid.FindBin(150.0f);
// bin = 242 (□□)
```

1.2.1.4. GetRepEnergy

```
float GetRepEnergy(int bin) const;
```

×□□:× □□□ □□ □□(□□) □□□□ □□□□□

×□□ □□:× □ □□□ □□ □□ □□

×□□□□:×

- `bin`: □ □□□ (0□□ N_E-1)

×□□:× □□ □□□ MeV

×□□:×

```
EnergyGrid e_grid(0.1f, 250.0f, 256);

int bin = e_grid.FindBin(150.0f);
float E_rep = e_grid.GetRepEnergy(bin);

// □ □□□ [145.0, 155.0]□ □□:
// E_rep = sqrt(145.0 × 155.0) = 149.83 MeV
```

1.2.2. AngularGrid

□□□ □□□ □□□□□□. EnergyGrid□ □□□□□ □□□□□□. □□ □□□ □□□□ □ □□ □□□ □□ □□□ □□□□□.

```
struct AngularGrid {
    const int N_theta;          // □□ □ □
    const float theta_min;      // □□ □□ [□]
    const float theta_max;      // □□ □□ [□]
    std::vector<float> edges;   // □ □□ (N_theta + 1□ □)
    std::vector<float> rep;     // □□ □□ (N_theta□ □)

    // □□□: □□□ □□
    AngularGrid(float theta_min, float theta_max, int N_theta);

    // □□□ □□□ □□ □ □□ (O(1) □□ □□)
    int FindBin(float theta) const;

    // □□ □□ □□ □□□□
    float GetRepTheta(int bin) const;
};
```

1.2.2.1. □□ □□□ □□□□ (□□ □□)

×□: □□ □□□ □□ □□×

| □□ □□ | □ | □□ | |—|—|—| | theta_min | −90° | □□ □□ | | theta_max | +90° | □□ □□ | | □ □ | 512 | □□ □ □□ | | □ 0 □□ □ □ | −89.8° | □□□ | | □ 384 □□ □□ | +30.2° | theta=+30.0°□ □□ | | FindBin(+30.0) | 384 | □ □□□ | | GetRepTheta(384) | +30.2° | □□ □□ |

1.2.2.2. □□□

```
AngularGrid(float theta_min, float theta_max, int N_theta);
```

×□□:× □□ □□□□ □□ □□□□ □□□□□

×□□□□:×

- `theta_min`: □□ □□ □ (□: −90.0)
- `theta_max`: □□ □□ □ (□: +90.0)
- `N_theta`: □ □ (□: 512)

×□□:× AngularGrid □□

×□□:×

```
// -90□□ +90□□□ 512□ □□□ □□□ □□
AngularGrid a_grid(-90.0f, 90.0f, 512);

// □□□ □□:
// a_grid.N_theta = 512
// a_grid.theta_min = -90.0
```

```
// a_grid.theta_max = +90.0
// □ □□ 0.35□ □□
```

1.2.2.3. FindBin

```
int FindBin(float theta) const;
```

×□□:× □□□ □□□ □□□□ □□ □□ □□□□

×□□ □□:× O(1) □□ □□ □□ (□□ □□)

×□□□□:×

- theta: □□ □ □

×□□:× □ □□□ (0□□ N_theta-1)

×□□:×

```
AngularGrid a_grid(-90.0f, 90.0f, 512);

// +30□□ □□ □ □□
int bin = a_grid.FindBin(30.0f);
// bin = 384 (□□)

// -45□□ □□ □ □□
int bin_neg = a_grid.FindBin(-45.0f);
// bin_neg = 64 (□□)
```

1.2.3. PsiC (□□ □□ □□□□)

PsiC□ □□□□□ □ □□ □□ □□□ □□□□ □□ □□□ □□□□□. □□□ 3D □□□□□□□□ □□□□□ - □ □□ □□□ □□ □□□ □□□□ □□□ □□□ □□□□ □□□□□.

```
struct PsiC {
    const int Nx;                    // □□□ X □□
    const int Nz;                    // □□□ Z □□
    const int Kb;                    // □□ □□ □□ (□□ 32)

    // □□□ □□
    std::vector<std::array<uint32_t, 32>> block_id;
    std::vector<std::array<std::array<float, LOCAL_BINS>, 32>> value;

    // □□□: □□□□ □□
    PsiC(int Nx, int Nz, int Kb);

    // □□ □□ □□ □□ □ □□ □□
    int find_or_allocate_slot(int cell, uint32_t bid);

    // □□ □□□□ □□□ □□
    float get_weight(int cell, int slot, uint16_t lidx) const;
    void set_weight(int cell, int slot, uint16_t lidx, float w);

    // □□ □□□ □□□
    void clear();

private:
    int N_cells;                     // □□ □ (Nx × Nz)
};
```

×C++ □□ □□×

×C++ □□ □□×

- `uint32_t`: □□ □□ 32□□ □□ (0□□ 4,294,967,295)
- `uint16_t`: □□ □□ 16□□ □□ (0□□ 65,535)
- `std::array<T, N>`: □□ □□ □□
- □□ □□: block_id□ □□□ □□, value□ □□□ □□□ □□

1.2.3.1. PsiC □□□ □□ □□

×□: PsiC □□□ □□ □□×

| □□ | □□ □□ | □□ | □□ | |—|—|—|—| | 1 | □□ □□□ | 2D □ □□ | Nx × Nz | | 2 | □ | □□ □□ □□ | 32□ □□ | | 3 | □□ □□ | □□ □ □□ | 512□ □ | | 4 | □□ □ | □□□ □ | 1□ float |

1.2.3.2. □□ □□ □□□ □□

×□: □□ □□ □□×

□□	□□	□□
1	□□ □□	(x,z,theta,E) □□
2	□ □□□ □□	see code comment above
3	□□ ID □□□	encode_block(b_theta, b_E)
4	□□ □□/□□	find_or_allocate_slot(cell, block_id)
5	□□ □□□ □□□	encode_local_idx_4d(...)
6	□□□ □□	set_weight(cell, slot, local_idx, weight)

1.2.3.3. □□□

```
PsiC(int Nx, int Nz, int Kb);
```

×□□:× □□□ □□□□ □□ □□ □□□□□ □□□□□

×□□□□:×

- `Nx`: X □□ □ □ (□: 100)
- `Nz`: Z □□ □ □ (□: 200)
- `Kb`: □□ □□ □□ (□□ 32)

×□□:× PsiC □□

×□□:×

```
// 32□ □□□ 100×200 □□□ □□
PsiC psi(100, 200, 32);

// □□:
// psi.Nx = 100
// psi.Nz = 200
// psi.Kb = 32
// psi.N_cells = 20,000
```

1.2.3.4. find_or_allocate_slot

```
int find_or_allocate_slot(int cell, uint32_t bid);
```

×□□:× □□ □□□ □□□ □ □□□ □□□□□

×□□ □□:×

1. □□ ID□ □ □□ □□ □□□ □□

2. ⊠⊠⊠⊠: ⊠⊠ ⊠⊠ ⊠⊠⊠ ⊠⊠
3. ⊠⊠⊠: ⊠ ⊠⊠ ⊠ ⊠⊠⊠ ⊠⊠ ⊠⊠ ID ⊠⊠
4. ⊠⊠ ⊠⊠⊠ ⊠⊠ (0-31)

×⊠⊠⊠⊠:×

- cell: ⊠ ⊠⊠⊠ (0⊠⊠ N_cells-1)
- bid: ⊠⊠/⊠⊠⊠ ⊠⊠ ID

×⊠⊠:× ⊠⊠ ⊠⊠⊠ (0⊠⊠ 31)

×⊠⊠:×

```
PsiC psi(100, 200, 32);
uint32_t block_id = encode_block(50, 100);

int cell = 5000;
int slot = psi.find_or_allocate_slot(cell, block_id);
// slot = 0 (⊠ ⊠⊠ ⊠ ⊠⊠)

// ⊠ ⊠⊠: ⊠⊠ ⊠⊠ ⊠⊠
int slot2 = psi.find_or_allocate_slot(cell, block_id);
// slot2 = 0 (⊠⊠⊠ ⊠⊠⊠ ⊠⊠)
```

1.2.3.5. get_weight / set_weight

```
float get_weight(int cell, int slot, uint16_t lidx) const;
void set_weight(int cell, int slot, uint16_t lidx, float w);
```

×⊠⊠:×

- get_weight: ⊠⊠ ⊠⊠⊠⊠ ⊠⊠⊠ ⊠⊠
- set_weight: ⊠⊠ ⊠⊠⊠ ⊠⊠⊠ ⊠⊠

×⊠⊠⊠⊠:×

- cell: ⊠ ⊠⊠⊠
- slot: ⊠⊠ ⊠⊠⊠ (0-31)
- lidx: ⊠⊠ ⊠ ⊠⊠⊠ (0-511)
- w (set_weight⊠): ⊠⊠⊠ ⊠⊠⊠ ⊠

×⊠⊠:×

- get_weight: ⊠⊠⊠ ⊠ (float)

×⊠⊠:×

```
PsiC psi(100, 200, 32);

// ⊠⊠⊠ ⊠⊠
int cell = 5000;
int slot = 5;
uint16_t local_idx = 123;
float weight = 0.0015f;

psi.set_weight(cell, slot, local_idx, weight);

// ⊠⊠⊠ ⊠⊠
float retrieved = psi.get_weight(cell, slot, local_idx);
// retrieved = 0.0015
```

1.3. ⊠⊠ ⊠⊠⊠ API

⊠⊠ ⊠⊠⊠ API⊠ ⊠⊠ ⊠⊠⊠ ⊠⊠⊠ 24⊠⊠ ⊠⊠⊠⊠ ⊠⊠⊠⊠⊠.

×☒☒ ☒☒☒☒×

☒☒ ☒☒☒☒ ☒ ☒☒ ☒☒☒ ☒☒ ☒☒☒ ☒☒☒☒ ☒☒ ☒☒☒☒☒:

- ☒☒ ☒: 12☒☒ (0-4095)
- ☒☒☒ ☒: 12☒☒ (0-4095)

24☒☒ ☒☒(3☒☒☒)☒ ☒☒☒☒ ☒☒☒☒ ☒☒☒☒ ☒☒ ☒☒☒ ☒☒☒☒ ☒☒☒.

```
// (b_theta, b_E) -> 24☒☒ ☒☒ ID ☒☒☒
__host__ __device__ inline uint32_t encode_block(
    uint32_t b_theta,  // ☒☒ ☒ (0-4095)
    uint32_t b_E       // ☒☒☒ ☒ (0-4095)
);

// ☒☒ ID -> (b_theta, b_E) ☒☒☒
__host__ __device__ inline void decode_block(
    uint32_t block_id,
    uint32_t& b_theta,  // ☒☒: ☒☒ ☒
    uint32_t& b_E       // ☒☒: ☒☒☒ ☒
);

// ☒ ☒☒☒ ☒☒ ☒☒ ☒
constexpr uint32_t EMPTY_BLOCK_ID = 0xFFFFFFFF;
```

×C++ ☒☒ ☒☒×

×C++ ☒☒ ☒☒×

- `__host__ __device__`: ☒☒☒ CPU(☒☒☒)☒ GPU(☒☒☒☒) ☒☒☒☒ ☒☒
- `inline`: ☒☒☒☒☒☒ ☒☒ ☒☒☒ ☒☒ ☒☒☒☒☒ ☒☒ (☒ ☒☒)
- `uint32_t&`: ☒☒☒ ☒☒ ☒☒(☒☒), ☒☒ ☒☒ ☒☒
- `constexpr`: ☒☒☒ ☒☒ ☒☒
- `0xFFFFFFFF`: ☒☒ 32☒☒☒ 1☒ ☒☒☒ ☒

1.3.0.1. ☒☒ ID ☒☒ ☒☒☒☒

×☒: ☒☒ ID ☒☒ ☒☒×

| ☒☒ ☒☒ | ☒☒ | ☒☒ | ☒ ☒☒ | ☒☒ | |—|—|—|—|—| | 23-12 | b_E | 12☒☒ | 0-4095 | 100 (0x064) | | 11-0 | b_theta | 12☒☒ | 0-4095 | 50 (0x034) |

×☒: ☒☒☒ ☒☒×

| ☒☒ | b_theta | b_E | ☒☒ ID (16☒☒) | ☒☒ ID (10☒☒) | |—|—|—|—|—| | ☒☒ 1 | 50 | 100 | 0x64034 | 410,740 | | ☒☒ 2 | 0 | 0 | 0x0 | 0 | | ☒☒ 3 | 4095 | 4095 | 0xFFFFFF | 16,777,215 |

1.3.0.2. encode_block

```
__host__ __device__ inline uint32_t encode_block(
    uint32_t b_theta,
    uint32_t b_E
);
```

×☒☒:× ☒☒☒ ☒☒☒ ☒☒ ☒☒ 24☒☒ ID☒ ☒☒☒☒

×☒☒ ☒☒:× `return (b_theta & 0xFFF) \|\| ((b_E & 0xFFF) << 12);`

×☒☒☒☒:×

- `b_theta`: ☒☒ ☒ (0-4095)
- `b_E`: ☒☒☒ ☒ (0-4095)

×⊠⊠:× 24⊠⊠ ⊠⊠ ID

×⊠⊠:×

```
// ⊠⊠⊠ ⊠ 100, ⊠⊠ ⊠ 50 ⊠⊠⊠
uint32_t bid = encode_block(50, 100);
// bid = 410,740
```

1.3.0.3. decode_block

```
__host__ __device__ inline void decode_block(
    uint32_t block_id,
    uint32_t& b_theta,
    uint32_t& b_E
);
```

×⊠⊠:× ⊠⊠ ID⊠⊠ ⊠⊠⊠ ⊠⊠⊠ ⊠⊠ ⊠⊠⊠⊠

×⊠⊠ ⊠⊠:×

```
b_theta = block_id & 0xFFF;           // ⊠⊠ 12⊠⊠ ⊠⊠
b_E = (block_id >> 12) & 0xFFF;      // ⊠⊠ 12⊠⊠ ⊠⊠
```

×⊠⊠⊠⊠:×

- `block_id`: ⊠⊠⊠⊠ ⊠⊠ ID
- `b_theta`: ⊠⊠ ⊠ ⊠⊠ ⊠⊠
- `b_E`: ⊠⊠⊠ ⊠ ⊠⊠ ⊠⊠

×⊠⊠:× ⊠⊠ (b_theta⊠ b_E⊠ ⊠⊠ ⊠⊠)

×⊠⊠:×

```
// ⊠⊠⊠
uint32_t bid = encode_block(50, 100);

// ⊠⊠⊠
uint32_t theta_bin, energy_bin;
decode_block(bid, theta_bin, energy_bin);
// theta_bin = 50
// energy_bin = 100
```

1.4. ⊠⊠ ⊠ API

⊠⊠ ⊠ API⊠ ⊠ ⊠⊠ ⊠⊠⊠ ⊠⊠⊠ ⊠⊠⊠ ⊠⊠⊠⊠.

×⊠⊠ ⊠⊠⊠⊠×

⊠⊠ ⊠⊠⊠⊠ ⊠⊠⊠ ⊠⊠ ⊠⊠⊠/⊠⊠ ⊠⊠⊠ ⊠⊠⊠⊠ ⊠⊠, ⊠⊠ ⊠⊠ ⊠ ⊠⊠ ⊠⊠⊠ ⊠⊠⊠ ⊠⊠⊠ ⊠⊠⊠⊠:

- 8⊠ ⊠⊠ ⊠⊠⊠ (theta_local: 0-7)
- 4⊠ ⊠⊠⊠ ⊠⊠⊠ (E_local: 0-3)
- 4⊠ X ⊠⊠ ⊠⊠⊠ (x_sub: 0-3)
- 4⊠ Z ⊠⊠ ⊠⊠⊠ (z_sub: 0-3)

⊠⊠: 8 × 4 × 4 × 4 = ⊠⊠⊠ 512⊠ ⊠⊠ ⊠

```
// ⊠⊠
constexpr int N_theta_local = 8;   // ⊠⊠ ⊠⊠ ⊠⊠⊠
constexpr int N_E_local = 4;       // ⊠⊠ ⊠⊠⊠ ⊠⊠⊠
constexpr int N_x_sub = 4;         // X ⊠⊠ ⊠⊠⊠
constexpr int N_z_sub = 4;         // Z ⊠⊠ ⊠⊠⊠
constexpr int LOCAL_BINS = 512;    // 8 x 4 x 4 x 4

// 4⊠⊠ ⊠⊠⊠ 16⊠⊠ ⊠⊠ ⊠⊠⊠⊠ ⊠⊠⊠
```

```
__host__ __device__ inline uint16_t encode_local_idx_4d(
    int theta_local,  // 0-7
    int E_local,      // 0-3
    int x_sub,        // 0-3
    int z_sub         // 0-3
);

// □□ □□□□ 4□□ □□□ □□□
__host__ __device__ inline void decode_local_idx_4d(
    uint16_t lidx,
    int& theta_local,
    int& E_local,
    int& x_sub,
    int& z_sub
);

// □□ □□
__host__ __device__ inline float get_x_offset_from_bin(int x_sub, float dx);
__host__ __device__ inline float get_z_offset_from_bin(int z_sub, float dz);
```

1.4.0.1. □□ □□□ □□□ □□

×□: 4D ⟶ 1D □□□ □□×

| □□ (theta, E, x, z) | □□ □□ | lidx | |—|—|—| | (5, 2, 1, 3) | 5 + 8×(2 + 4×(1 + 4×3)) | 437 | | (0, 0, 0, 0) | 0 | 0 | | (7, 3, 3, 3) | 7 + 8×(3 + 4×(3 + 4×3)) | 511 |

1.4.0.2. □ □□□

×□: X □ □□□ (dx = 2.0 mm)×

| x_sub | □□□ | □□ | |—|—|—| | 0 | −0.750 mm | −3/8 × dx | | 1 | −0.250 mm | −1/8 × dx | | 2 | +0.250 mm | +1/8 × dx | | 3 | +0.750 mm | +3/8 × dx |

(Z □□ □□□ □□)

1.4.0.3. encode_local_idx_4d

```
__host__ __device__ inline uint16_t encode_local_idx_4d(
    int theta_local,
    int E_local,
    int x_sub,
    int z_sub
);
```

×□□:× 4□□ □□□ □□ 16□□ □□□□ □□□□□

×□□□□:×

- `theta_local`: □□ □□□ (0-7)
- `E_local`: □□□ □□□ (0-3)
- `x_sub`: X □□ □□□ (0-3)
- `z_sub`: Z □□ □□□ (0-3)

×□□:× □□ □□□ (0-511)

×□□:×

```
uint16_t lidx = encode_local_idx_4d(5, 2, 1, 3);
// lidx = 437
```

1.4.0.4. decode_local_idx_4d

```
__host__ __device__ inline void decode_local_idx_4d(
    uint16_t lidx,
```

```
    int& theta_local,
    int& E_local,
    int& x_sub,
    int& z_sub
);
```

×⊠⊠:× ⊠⊠ ⊠⊠⊠ 4⊠⊠ ⊠⊠⊠ ⊠⊠⊠⊠

×⊠⊠⊠:×

- `lidx`: ⊠⊠ ⊠⊠⊠ (0-511)
- ⊠⊠ ⊠⊠⊠: 4⊠⊠ ⊠⊠

×⊠⊠:×

```
uint16_t lidx = 437;
int theta_local, E_local, x_sub, z_sub;
decode_local_idx_4d(lidx, theta_local, E_local, x_sub, z_sub);
// (5, 2, 1, 3)
```

1.4.0.5. ⊠⊠ ⊠⊠ ⊠⊠

```
float get_x_offset_from_bin(int x_sub, float dx);
float get_z_offset_from_bin(int z_sub, float dz);
```

×⊠⊠:× ⊠ ⊠⊠⊠ ⊠ ⊠⊠⊠ ⊠⊠⊠ ⊠⊠⊠⊠ ⊠⊠⊠⊠

×⊠⊠⊠:×

- `x_sub/z_sub`: ⊠ ⊠⊠⊠ (0-3)
- `dx/dz`: ⊠ ⊠⊠ mm

×⊠⊠:× ⊠ ⊠⊠⊠ ⊠⊠⊠ mm

×⊠⊠:×

```
float dx = 2.0f;
float x_off = get_x_offset_from_bin(2, dx);
// x_off = +0.25 mm
```

1.5. ⊠⊠ API

⊠⊠ API⊠ ⊠⊠⊠ ⊠⊠⊠ ⊠⊠⊠⊠ ⊠⊠⊠⊠.

1.5.1. Highland ⊠⊠ (⊠⊠ ⊠⊠ ⊠⊠)

> ×⊠⊠ ⊠⊠⊠×
>
> ⊠⊠⊠ ⊠⊠⊠ ⊠⊠⊠ ⊠ ⊠⊠⊠ ⊠⊠⊠⊠⊠ ⊠⊠⊠⊠(⊠⊠). Highland ⊠⊠⊠ ×⊠⊠⊠⊠⊠(RMS)× ⊠⊠ ⊠⊠⊠ ⊠⊠⊠⊠ - ⊠⊠⊠ ⊠⊠ ⊠⊠⊠ ⊠⊠ ⊠⊠⊠ ⊠⊠ ⊠⊠⊠⊠.
>
> ⊠ ⊠⊠ ⊠⊠⊠ → ⊠ ⊠⊠ ⊠⊠, ⊠ ⊠⊠⊠ ⊠⊠ → ⊠ ⊠⊠ ⊠⊠

```
// MCS ⊠⊠ ⊠⊠⊠ [rad] ⊠⊠
__host__ __device__ float highland_sigma(
    float E_MeV,    // ⊠⊠⊠ ⊠⊠⊠ [MeV]
    float ds,       // ⊠⊠ ⊠⊠ [mm]
    float X0        // ⊠⊠⊠ ⊠⊠ [mm]
);

// ⊠⊠ ⊠⊠ ⊠⊠⊠ (Box-Muller ⊠⊠)
__device__ float sample_mcs_angle(
    float sigma_theta,      // RMS ⊠⊠ ⊠⊠
    unsigned& seed          // RNG ⊠⊠
);
```

```
// □□ □ □□ □□□ □□□□
__device__ void update_direction_after_mcs(
    float& mu,       // □□ □□□ X (□□/□□)
    float& eta,      // □□ □□□ Z (□□/□□)
    float delta_theta  // □□ □□ [rad]
);
```

×□□ □□×

×□□□ □□ (X0)×

□□□ □□□ □□ □□□□□:

- □: X0 ≈ 360 mm
- □: X0 ≈ 5.6 mm (□□□ □□ ⟶ □ □□ □□)
- □□: X0 ≈ 30,000 mm (□□□ ⟶ □ □□ □□)

1.5.1.1. □□ □□ □□ □□

×□: □□ □□×

| □□□ | RMS □□ □□ | □□ | |—|—|—| | 150 MeV | 0.13° | □□ □□□, □□ □□ | | 50 MeV | 0.41° | □□ □□□, □ □□ □□ |

×□: Highland □□ □□ □□×

| □□ □□ | □□ | |—|—| | 13.6 MeV | □□ | | E | □□□ □□□ | | ds | □□ □□ | | X0 | □□□ □□ | | □□ □ | □□ □□ |

1.5.1.2. highland_sigma

```
__host__ __device__ float highland_sigma(float E_MeV, float ds, float X0);
```

×□□:× Highland □□□□ RMS □□ □□□ □□□□□

×□□:× sigma_theta = frac(13.6 text(MeV))(E_text(MeV)) sqrtleft(frac(d_s)(X_0)right) left[1 + 0.038 lnleft(frac(d_s)(X_0)right)right]

×□□□□:×

- E_MeV: □□□ □□□ MeV
- ds: □□ □□ mm
- X0: □□□ □□ mm

×□□:× RMS □□ □□ □□□

×□□:×

```
// □□□ 1mm □□□□ 150 MeV □□□
float E = 150.0f;
float ds = 1.0f;
float X0 = 360.0f;

float sigma = highland_sigma(E, ds, X0);
// sigma ≈ 0.0022 rad ≈ 0.13°
```

1.5.1.3. sample_mcs_angle

```
__device__ float sample_mcs_angle(float sigma_theta, unsigned& seed);
```

×□□:× □□□□ □□□□ □□□ □□ □□□ □□□□□□

×□□□□:×

- sigma_theta: RMS □□ □□
- seed: □□□ □□ □□□ □□

×□□:× □□ □□ □□□

×□□:×

```
unsigned seed = 42;
float sigma = 0.0022f;
float delta_theta = sample_mcs_angle(sigma, seed);
```

1.5.2. [illegible] [illegible]

×[illegible] [illegible]×
[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] - [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]. "[illegible] (straggling)"[illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

```
// Vavilov [illegible] [illegible] [illegible]
__host__ __device__ float vavilov_kappa(
    float E_MeV,    // [illegible] [illegible] [MeV]
    float ds        // [illegible] [illegible] [mm]
);

// Bohr [illegible] [illegible]
__host__ __device__ float bohr_straggling_sigma(
    float E_MeV,    // [illegible] [illegible] [MeV]
    float ds        // [illegible] [illegible] [mm]
);

// [illegible] [illegible] [illegible] [illegible] [illegible]
__device__ float sample_energy_loss_with_straggling(
    float E_MeV,
    float ds,
    unsigned& seed
);
```

1.5.2.1. Vavilov [illegible] [illegible]

×[illegible]: [illegible] [illegible] [illegible] [illegible]×

| [illegible] [illegible] | [illegible] | [illegible] | |—|—|—| | < 0.01 | Landau | [illegible] [illegible] [illegible] [illegible] [illegible] | | 0.01 - 10 | Vavilov | [illegible] [illegible] | | > 10 | [illegible] | [illegible] [illegible] [illegible] [illegible] |

×[illegible]: [illegible] [illegible] [illegible]×

| [illegible] | [illegible] | [illegible] | |—|—|—| | 200 MeV | 0.005 | Landau | | 150 MeV | 0.01 | Vavilov | | 10 MeV | 0.1 | Vavilov |

1.5.2.2. vavilov_kappa

```
__host__ __device__ float vavilov_kappa(float E_MeV, float ds);
```

×[illegible]:× Vavilov [illegible] [illegible] [illegible]

×[illegible]:×
- E_MeV: [illegible] [illegible] MeV
- ds: [illegible] [illegible] mm

×[illegible]:× [illegible] [illegible] ([illegible])

×[illegible]:×

```
float kappa_high = vavilov_kappa(200.0f, 1.0f);
// kappa_high ≈ 0.005 (Landau)

float kappa_low = vavilov_kappa(10.0f, 1.0f);
// kappa_low ≈ 0.1 (Vavilov)
```

1.5.2.3. bohr_straggling_sigma

```
__host__ __device__ float bohr_straggling_sigma(float E_MeV, float ds);
```

×[illegible]:× [illegible]

×[illegible]:×
- E_MeV: [illegible] MeV
- ds: [illegible] mm

×[illegible]:× [illegible] MeV

×[illegible]:×

```
float E = 150.0f;
float ds = 1.0f;
float sigma = bohr_straggling_sigma(E, ds);
// sigma ≈ 0.15 MeV
```

1.5.3. [illegible]

> ×[illegible]×
> [illegible]:
> - [illegible]
> - [illegible] ([illegible] 1cm[illegible] 1% [illegible])
> - [illegible] "[illegible]"[illegible] - [illegible]

```
// [illegible] [mm^-1]
__host__ __device__ float Sigma_total(float E_MeV);

// [illegible]
__device__ void apply_nuclear_attenuation(
    float& weight,        // [illegible] ([illegible])
    double& energy_rem,   // [illegible] ([illegible])
    float E_MeV,          // [illegible]
    float ds              // [illegible]
);
```

1.5.3.1. [illegible]

×[illegible]: [illegible]×

| [illegible] | [illegible] | [illegible] | |–|–|–| | Sigma_total (150 MeV) | 0.01 mm^-1 | [illegible] | | [illegible] (1mm) | 0.99 | exp(-0.01) | | [illegible] | 1% | 1mm[illegible] |

×[illegible]: [illegible]×

| [illegible] | [illegible] | [illegible] | |–|–|–| | [illegible] | 1.0 | [illegible] | | 1mm [illegible] | 0.99 | 1% [illegible] | | 2mm [illegible] | 0.98 | [illegible] 1% [illegible] |

1.5.3.2. Sigma_total

```
__host__ __device__ float Sigma_total(float E_MeV);
```

×[illegible]:× [illegible]

×[illegible]:×
- E_MeV: [illegible] MeV

×[illegible]:× [illegible] mm^-1

×[illegible]:×

```
float sigma = Sigma_total(150.0f);
// sigma ≈ 0.01 mm^-1
```

1.5.3.3. apply_nuclear_attenuation

```
__device__ void apply_nuclear_attenuation(
    float& weight,
    double& energy_rem,
    float E_MeV,
    float ds
);
```

×[illegible]:× [illegible]

×[illegible]:×

- weight: [illegible] ([illegible])
- energy_rem: [illegible]
- E_MeV: [illegible]
- ds: [illegible]

×[illegible]:×

```
float weight = 1.0f;
double energy_rem = 150.0;
apply_nuclear_attenuation(weight, energy_rem, 150.0f, 1.0f);
// weight ≈ 0.99
```

1.5.4. [illegible]

×[illegible]×
[illegible] "[illegible]"[illegible]. R [illegible]: [illegible].

[illegible] ⟶ [illegible], [illegible] ⟶ [illegible]

```
// [illegible] (R [illegible])
__host__ __device__ float compute_max_step_physics(
    float E,
    const RLUT& lut
);

// [illegible] (R [illegible])
__device__ float compute_energy_after_step(
    float E_in,
    float ds,
    const RLUT& lut
);

// [illegible]
__device__ float compute_energy_deposition(
    float E_in,
    float ds,
    const RLUT& lut
);
```

1.5.4.1. [illegible]

×[illegible]: [illegible]×

[illegible] (MeV)	[illegible] (mm)	[illegible]
250	10	[illegible], [illegible]
150	5	[illegible]

□□□ (MeV)	□□ □□ (mm)	□□
50	1	□□ □□□, □□ □□
10	0.1	□□ □□ □□□

1.5.4.2. compute_max_step_physics

```
__host__ __device__ float compute_max_step_physics(float E, const RLUT& lut);
```

×□□:× □□□□ □□ □□ □□□ □□□□□

×□□:× `ds_max = max(0.1, 0.05 × remaining_range)`

×□□□□:×

- `E`: □□ □□□ MeV
- `lut`: □□□-□□□ □□ □□□

×□□:× □□ □□ □□ mm

×□□:×

```
RLUT lut = load_r_lut("water.txt");
float E = 150.0f;
float ds_max = compute_max_step_physics(E, lut);
// ds_max ≈ 5 mm
```

1.6. LUT API

LUT (□□ □□□) API□ □□ □□□ □□ □□□□ □□ □□□□ □□□□□.

1.6.1. RLUT (□□□-□□□ □□ □□□)

> ×□□ □□□□×
> RLUT□ NIST □□□□□□□ □□ □□□ □□□-□□□ □□□□ □□□□□:
> - □□□: □□□□ □□□ □□□ □□□□ □□
> - □□ □□: □□□ □□□ (dE/dx)
> - □□-□□ □□□ □□ □□ □□
>
> □□ □□□□ □□□□ □□ □□□□ □□□□ □□□□ □□□ □□ □□□□.

```
struct RLUT {
    EnergyGrid grid;
    std::vector<float> R;        // CSDA □□□ [mm]
    std::vector<float> S;        // Stopping power [MeV cm^2/g]
    std::vector<float> log_E;    // □□ □□□ log(E)
    std::vector<float> log_R;    // □□ □□□ log(R)
    std::vector<float> log_S;    // □□ □□□ log(S)

    // □□□□□ □□□ (□□-□□ □□)
    float lookup_R(float E_MeV) const;

    // □□□□□ stopping power
    float lookup_S(float E_MeV) const;

    // □□□□□ □□□ (□□□)
    float lookup_E_inverse(float R_mm) const;
};
```

1.6.1.1. □□□-□□□ □□

×□: □□□ □□□ □□□×

| □□□ [MeV] | □□□ [mm] | □□ □□ | |—|—|—| | 50 | 25 | □□ | | 100 | 75 | □□ | | 150 | 160 | □□ | | 200 | 260 | □□ □□ | | 250 | 380 | □□ |

1.6.1.2. lookup_R

```
float lookup_R(float E_MeV) const;
```

×□□:× □□□ □□□□ □□ CSDA □□□□ □□□□□

×□□□□:×

- `E_MeV`: □□□ MeV

×□□:× □□□ mm

×□□:×

```
RLUT lut = load_r_lut("water.txt");
float range = lut.lookup_R(150.0f);
// range ≈ 160 mm
```

1.6.2. NIST □□□ □□

```
struct NistDataRow {
    float energy_MeV;          // □□ □□□ [MeV]
    float stopping_power;      // dE/dx [MeV cm^2/g]
    float csda_range_g_cm2;    // CSDA □□□ [g/cm^2]
};

// □□□□ NIST PSTAR □□□ □□
std::vector<NistDataRow> load_nist_pstar(
    const std::string& filepath
);

// NIST □□□□□ RLUT □□
RLUT create_r_lut_from_nist(
    const std::vector<NistDataRow>& nist_data,
    float E_min,
    float E_max,
    int N_E
);
```

×□: NIST □□□ □□×

| □□ | □□ | □□ | |—|—|—| | energy_MeV | MeV | □□ □□□ | | stopping_power | MeV cm²/g | □□ □□ □□ | | csda_range_g_cm2 | g/cm² | □□ □□ □□□ |

1.7. □□ API

□□ API□ □□□□□□ □□□ □□□□□.

1.7.1. PencilSource

> ×□□ □□□□×
> □□ □□ □□ □□□ □□□□□:
> - (x0, z0)□ □□ □□
> - (theta0)□ □□ □□
> - (E0)□ □□ □□□
> - □□ □□□□ □□□ □□□

⊠⊠⊠⊠ ⊠⊠⊠ ⊠⊠⊠⊠⊠.

```cpp
struct PencilSource {
    float x0 = 0.0f;        // X ⊠⊠ [mm]
    float z0 = 0.0f;        // Z ⊠⊠ [mm]
    float theta0 = 0.0f;    // ⊠⊠ ⊠⊠ [rad]
    float E0 = 150.0f;      // ⊠⊠ ⊠⊠⊠ [MeV]
    float W_total = 1.0f;   // ⊠⊠ ⊠⊠⊠
};

void inject_pencil_source(
    const PencilSource& source,
    PsiC& psi,
    const EnergyGrid& e_grid,
    const AngularGrid& a_grid,
    int Nx, int Nz, float dx, float dz
);
```

×⊠: ⊠⊠ ⊠ ⊠⊠⊠⊠×

| ⊠⊠⊠⊠ | ⊠⊠ ⊠ | ⊠⊠ | |—|—|—| | x0 | 50 mm | X ⊠⊠ | | z0 | 0 mm | Z ⊠⊠ (⊠⊠) | | theta0 | 0 rad | ⊠⊠ (⊠⊠⊠) | | E0 | 150 MeV | ⊠⊠⊠ | | W_total | 1.0 | ⊠⊠ ⊠⊠⊠ |

1.7.2. GaussianSource

×⊠⊠ ⊠⊠⊠⊠×

⊠⊠⊠⊠ ⊠⊠⊠ ⊠⊠⊠⊠ ⊠⊠ ⊠⊠ ⊠⊠⊠⊠⊠⊠:

- ⊠⊠ ⊠⊠: sigma_x (⊠ ⊠⊠ ⊠⊠)
- ⊠⊠ ⊠⊠: sigma_theta (⊠ ⊠⊠)
- ⊠⊠⊠ ⊠⊠: sigma_E (⊠⊠⊠⊠⊠ ⊠⊠⊠ ⊠⊠)
- ⊠⊠⊠⊠ ⊠⊠⊠⊠ ⊠⊠ ⊠⊠

⊠⊠⊠⊠ ⊠⊠⊠⊠⊠⊠ ⊠⊠⊠⊠⊠.

```cpp
struct GaussianSource {
    float x0 = 0.0f;            // ⊠⊠ X ⊠⊠ [mm]
    float z0 = 0.0f;            // ⊠⊠ Z ⊠⊠ [mm]
    float theta0 = 0.0f;        // ⊠⊠ ⊠⊠ [rad]
    float sigma_x = 5.0f;       // X ⊠⊠ ⊠⊠ [mm]
    float sigma_theta = 0.01f;  // ⊠⊠ ⊠⊠ ⊠⊠ [rad]
    float E0 = 150.0f;          // ⊠⊠ ⊠⊠⊠ [MeV]
    float sigma_E = 1.0f;       // ⊠⊠⊠ ⊠⊠ ⊠⊠ [MeV]
    float W_total = 1.0f;       // ⊠⊠ ⊠⊠⊠
    int n_samples = 1000;       // ⊠⊠⊠⊠⊠ ⊠⊠
};

void inject_gaussian_source(
    const GaussianSource& source,
    PsiC& psi,
    const EnergyGrid& e_grid,
    const AngularGrid& a_grid,
    int Nx, int Nz, float dx, float dz,
    unsigned seed = 42
);
```

×⊠: ⊠⊠⊠⊠ ⊠⊠ ⊠⊠⊠⊠×

| □□□□ | □□ □ | □□ | |—|—|—| | x0 | 50 mm | □□ X □□ | | sigma_x | 5 mm | □ □□ □□ | | sigma_theta | 0.01 rad | □ □□ | | sigma_E | 1 MeV | □□□ □□ | | n_samples | 1000 | □□ □ |

1.8. □□ API

□□ API□ □□□□ □□□ □□□□ □□□ □□□□□.

```
enum class BoundaryType {
    ABSORB,     // □□□□ □□ □□
    REFLECT,    // □□□ □□ □□
    PERIODIC    // □□□ □□□ □□□ □□
};

struct BoundaryConfig {
    BoundaryType z_min = BoundaryType::ABSORB;
    BoundaryType z_max = BoundaryType::ABSORB;
    BoundaryType x_min = BoundaryType::ABSORB;
    BoundaryType x_max = BoundaryType::ABSORB;
};

struct BoundaryLoss {
    float weight[4];   // □□ □□□ □□□
    double energy[4];  // □□ □□□ □□□
    // □ □□□: 0=+z, 1=-z, 2=+x, 3=-x
};
```

×□□ □□□□×
×□□ □□×

- ×ABSORB×: □□□ □□□□□ (□□□□□ □□□ □□)
- ×REFLECT×: □□□ □□□ □□□□ (□□□□)
- ×PERIODIC×: □□□ □□□□□ □□□□□ (□□□□)

□□ □□ □□□□□□ □□:
- □□ (X) □□: □□□□□ REFLECT □□ PERIODIC
- □□□ (Z) □□: □□□□□ ABSORB (□□ □□ □□)

×□: □□ □□×

| □□ □□ | □□ | □□ □□ | |—|—|—| | ABSORB | □□ □□ | Z □□ (□□/□□) | | REFLECT | □□ | X □□ (□□) | | PERIODIC | □□ | X □□ (□□□) |

1.9. □□ API

□□ API□ □□ □□(□□□□ □□□)□ □□□□□.

×□□ □□□□×
□□ □□□ □□□□□:
- □□□ □□: □□□ □□ = □□□ □□ + □□
- □□□ □□: □□□ □□ = □□□ □□ + □□ + □□

□□ □□□□ □ □□□□ □ □□□□ □□□□ □□□ □□ □□□ □□□□□.

```
struct CellWeightAudit {
    float W_in;            // □□ □□□
    float W_out;           // □□ □□□
    float W_cutoff;        // □□□□ □□□ □□□
    float W_nuclear;       // □□□ □□□ □□□
    float W_error;         // □□ □□
```

```
    bool check() const {
        float W_expected = W_out + W_cutoff + W_nuclear;
        float W_rel = fabs(W_in - W_expected) / fmax(W_in, 1e-20f);
        return W_rel < 1e-6f;
    }
};

struct CellEnergyAudit {
    double E_in;            // ⌧⌧ ⌧⌧⌧
    double E_out;           // ⌧⌧ ⌧⌧⌧
    double E_dep;           // ⌧⌧⌧ ⌧⌧⌧
    double E_nuclear;       // ⌧⌧⌧ ⌧⌧⌧ ⌧⌧⌧
    double E_error;         // ⌧⌧ ⌧⌧

    bool check() const {
        double E_expected = E_out + E_dep + E_nuclear;
        double E_rel = fabs(E_in - E_expected) / fmax(E_in, 1e-20);
        return E_rel < 1e-5;
    }
};
```

1.9.0.1. ⌧⌧ ⌧⌧ ⌧⌧

×⌧: ⌧⌧⌧ ⌧⌧ ⌧⌧×

| ⌧⌧ | ⌧ | ⌧⌧ | |—|—|—| | W_in | 1.000 | ⌧⌧ ⌧⌧⌧ | | W_out | 0.990 | ⌧⌧ ⌧⌧⌧ | | W_cutoff | 0.005 | ⌧⌧ ⌧⌧ | | W_nuclear | 0.005 | ⌧ ⌧⌧ | | ⌧⌧ | 1.000 | ⌧⌧ ✓ |

×⌧: ⌧⌧⌧ ⌧⌧ ⌧⌧×

| ⌧⌧ | ⌧ | ⌧⌧ | |—|—|—| | E_in | 150.0 MeV | ⌧⌧ ⌧⌧⌧ | | E_out | 149.5 MeV | ⌧⌧ ⌧⌧⌧ | | E_dep | 0.4 MeV | ⌧⌧⌧ ⌧⌧⌧ | | E_nuclear | 0.1 MeV | ⌧ ⌧⌧ | | ⌧⌧ | 150.0 MeV | ⌧⌧ ✓ |

1.10. ⌧⌧ API

⌧⌧ API⌧ ⌧⌧⌧⌧ ⌧⌧⌧ ⌧⌧ ⌧⌧⌧⌧ ⌧⌧⌧⌧.

1.10.1. ⌧⌧⌧ ⌧⌧ ⌧⌧

```
struct BraggPeakResult {
    float position_mm;      // ⌧⌧ ⌧⌧ [mm]
    float peak_dose;        // ⌧⌧ ⌧⌧
    float fwhm_mm;          // ⌧⌧⌧
    float R80;              // 80% ⌧⌧ ⌧⌧
    float R20;              // 20% ⌧⌧ ⌧⌧
    float distal_falloff;   // R80 - R20
    float position_error;   // NIST ⌧⌧ ⌧⌧
    bool pass;              // ±2% ⌧⌧
};

BraggPeakResult analyze_bragg_peak(
    const std::vector<float>& z_mm,
    const std::vector<float>& dose,
    float reference_position_mm
);
```

×⌧⌧ ⌧⌧⌧⌧×

⌧⌧⌧ ⌧⌧⌧ ⌧⌧⌧ ⌧⌧⌧ ⌧⌧⌧⌧ ⌧⌧ ⌧⌧⌧⌧:

- ⌧⌧⌧ ⌧ ⌧⌧⌧ ⌧⌧⌧ ⌧⌧ ⌧⌧

- □□ □□ □□ □□ □□
- □□□ □□□ ±2% □□□ □□□□□ □□□

×□: □□□ □□ □□ □□×

| □□ | □□ | □□ □□ | |—|—|—| | □□ □□ | □□ □□ □□ | NIST ±2% □ | | FWHM | □□□ | □□□ | | R80 | 80% □□ □□ | □□□ | | R20 | 20% □□ □□ | □□□ | | □□□ □□ | R80 - R20 | □□□□ □□ |

×□: □□□ □□ □□×

| □□□ | □□ □□ (□) | FWHM | □□ | |—|—|—|—| | 100 MeV | 75 mm | 15 mm | □□ □□ | | 150 MeV | 160 mm | 20 mm | □□ □□ | | 200 MeV | 260 mm | 25 mm | □□ □□ |

1.11. □□□□ API

1.11.1. □□

```cpp
enum class LogLevel {
    TRACE,
    DEBUG,
    INFO,
    WARN,
    ERROR
};

Logger& get_logger();
void set_log_level(LogLevel level);

#define LOG_TRACE(...) get_logger().log(LogLevel::TRACE, __VA_ARGS__)
#define LOG_DEBUG(...) get_logger().log(LogLevel::DEBUG, __VA_ARGS__)
#define LOG_INFO(...)  get_logger().log(LogLevel::INFO, __VA_ARGS__)
#define LOG_WARN(...)  get_logger().log(LogLevel::WARN, __VA_ARGS__)
#define LOG_ERROR(...) get_logger().log(LogLevel::ERROR, __VA_ARGS__)
```

×C++ □□ □□×

×C++ □□□□ □□□×

- #define: □□□ □□ (□□□ □ □□□ □□)
- __VA_ARGS__: □□□□ □□ □□ □□
- □□□□ □□□□ □□ □□, □ □□, □□ □□□ □□

×□: □□ □□×

| □□ | □□ | □□ | |—|—|—| | TRACE | □□ □□ | □□ □□/□□ | | DEBUG | □□□ | □□ □ | | INFO | □□ | □□ □□ | | WARN | □□ | □□ □ □□ | | ERROR | □□ | □□ |

1.11.1.1. □□ □□ □□

```cpp
#include "utils/logger.hpp"

int main() {
    set_log_level(LogLevel::INFO);

    LOG_INFO("□□□□□ □□");
    LOG_WARN("□□: □□□ □□");
    LOG_ERROR("□□: □□□ □□");

    return 0;
}
```

1.11.2. □□□ □□□

```cpp
struct MemoryInfo {
    size_t free_bytes;      // □□ □□□
    size_t total_bytes;     // □□ □□□
    size_t used_bytes;      // □□□ □□□
    float utilization;      // □□□ / □□
};

MemoryInfo query_gpu_memory();
bool check_memory_warning(float threshold = 0.9f);
void print_memory_summary();
```

×C++ □□ □□×

×C++ □□ □□×

- `size_t`: □□□ □□ □□ □□ □□
- `1e9`: □□□ □□□ (10□)
- □□ □□□□ □□□ □□□□ □□ `size_t` □□

1.11.2.1. □□□ □□ □□

```cpp
#include "utils/memory_tracker.hpp"

int main() {
    MemoryInfo info = query_gpu_memory();

    printf("GPU □□□:\n");
    printf("  □□: %.2f GB\n", info.total_bytes / 1e9);
    printf("  □□:  %.2f GB\n", info.used_bytes / 1e9);
    printf("  □□:  %.2f GB\n", info.free_bytes / 1e9);
    printf("  □□:  %.1f%%\n", info.utilization × 100);

    if (check_memory_warning(0.9f)) {
        printf("□□: GPU □□□□ >90%% □□□□!\n");
    }

    return 0;
}
```

1.12. □□ □□□□□ API

```cpp
int run_simulation(
    const std::string& config_file = "sim.ini",
    const std::string& output_dir = "results"
);

struct SimulationResult {
    int exit_code;                      // 0 = □□
    int n_steps_completed;              // □□□ □□
    double wall_time_seconds;           // □□□

    GlobalAudit audit;                  // □□ □□
    BraggPeakResult bragg_peak;         // □□□ □□ □□

    std::string dose_file;              // □□ □□
    std::string pdd_file;
    std::string audit_file;
};
```

```
SimulationResult run_simulation_detailed(const SimulationConfig& config);
```

×□: □□□□□ □□×

| □□ | □□ | □□ | |—|—|—| | exit_code | □□ □□ | 0 (□□) | | n_steps_completed | □□□ □□ | 150 | | wall_time_seconds | □□□ | 2.34 □ | | audit.pass | □□ □□ | true | | bragg_peak.position_mm | □□ □□ | 158.23 mm | | bragg_peak.position_error | □□ □□ | 1.34% |

1.13. □□

×□□ □□×
×□□□□ □□×

1. ××□□□ □□××: PsiC □□ □□ □□□ □□
2. ××□□ □□□××: encode_block()□ □□ □□
3. ××□□ □××: encode_local_idx_4d()□ □□□ □□
4. ××□□××: □□ □□ → □□ → □□ → □ □□ □□
5. ××□□××: □ □□ □□ □ □□ □□
6. ××□□××: □ □□ □ □□ □□
7. ××□□××: □□ □□□□ □□ □□

×□□ □□×

- □□ □ □□ □□□ □□□
- □□ □□ □□ □□ □□
- □□□□ □□ □□ □□
- □□ □□□□ □□
- □□□ □□□□ □□ □□□□□

×□□ □□□□×
×□□ □□×

1. □□ API □□ (□□□, PsiC)
2. □□ □ □□ □ □□□ □□
3. □□ □□ (□□, □□, □)
4. □□□ □□ (□□, □□□□)
5. □□ □ □□ □□
6. □□□□ □□ □□□□□ □□

×□□ □□×

- `src/include/`□□ □□ □□ □□
- `tests/`□□ □□ □□
- □□ □□ □□
- □□ □□ □□
- □□□ □□□□□ □□

2. □□ A: □□ □□ □□

2.1. EnergyGrid □□ □□

```cpp
#include "core/grids.hpp"

int main() {
    // □□ 1: □□□ □□□ □□
```

```
    EnergyGrid e_grid(0.1f, 250.0f, 256);

    // □□ 2: □□□ □□□□ □□ □ □□
    float proton_energy = 150.0f;
    int bin = e_grid.FindBin(proton_energy);

    // □□ 3: □□ □□□ □□□□
    float rep_energy = e_grid.GetRepEnergy(bin);

    // □□ 4: □□□□□□□ □□
    printf("□□□ □□□: %.2f MeV\n", proton_energy);
    printf("□ □□□: %d\n", bin);
    printf("□□ □□□: %.2f MeV\n", rep_energy);

    return 0;
}
```

2.2. AngularGrid □□ □□

```
#include "core/grids.hpp"

int main() {
    // □□ 1: □□ □□□ □□
    AngularGrid a_grid(-90.0f, 90.0f, 512);

    // □□ 2: □□ □□□ □□ □ □□
    float angle = 30.0f;
    int bin = a_grid.FindBin(angle);

    // □□ 3: □□ □□ □□□□
    float rep_angle = a_grid.GetRepTheta(bin);

    // □□ 4: □□□□□□□ □□
    printf("□□ □□: %.2f°\n", angle);
    printf("□ □□□: %d\n", bin);
    printf("□□ □□: %.2f°\n", rep_angle);

    return 0;
}
```

2.3. PsiC □□ □□

```
#include "core/psi_storage.hpp"
#include "core/grids.hpp"
#include "core/block_encoding.hpp"
#include "core/local_bins.hpp"

int main() {
    // □□ 1: □□□ □□
    EnergyGrid e_grid(0.1f, 250.0f, 256);
    AngularGrid a_grid(-90.0f, 90.0f, 512);

    // □□ 2: PsiC □□
    int Nx = 100, Nz = 200;
    PsiC psi(Nx, Nz, 32);

    // □□ 3: □□□ □□ □□
    int ix = 50, iz = 100;
    int cell = iz × Nx + ix;
```

```cpp
    // □□ 4: □□ □□
    float E = 150.0f;
    float theta = 30.0f;
    float x_offset = 0.1f;
    float z_offset = -0.2f;
    float weight = 0.001f;

    // □□ 5: □ □□
    int b_E = e_grid.FindBin(E);
    int b_theta = a_grid.FindBin(theta);

    // □□ 6: □□ ID □□□
    uint32_t bid = encode_block(b_theta, b_E);

    // □□ 7: □□ □□ □□ □□
    int slot = psi.find_or_allocate_slot(cell, bid);

    // □□ 8: □□ □ □□
    int theta_local = b_theta % 8;
    int E_local = b_E % 4;
    float dx = 1.0f, dz = 1.0f;
    int x_sub = get_x_sub_bin(x_offset, dx);
    int z_sub = get_z_sub_bin(z_offset, dz);

    // □□ 9: □□ □□□ □□□
    uint16_t lidx = encode_local_idx_4d(theta_local, E_local, x_sub, z_sub);

    // □□ 10: □□□ □□
    psi.set_weight(cell, slot, lidx, weight);

    return 0;
}
```

2.4. □□ □□□ □□ □□

```cpp
#include "core/block_encoding.hpp"
#include <cassert>

int main() {
    // □ 1: □□ □□□/□□□
    uint32_t theta_bin = 50;
    uint32_t energy_bin = 100;

    // □□□
    uint32_t block_id = encode_block(theta_bin, energy_bin);
    printf("□□□□: %u (0x%X)\n", block_id, block_id);

    // □□□
    uint32_t theta_out, energy_out;
    decode_block(block_id, theta_out, energy_out);
    printf("□□□□: theta=%u, energy=%u\n", theta_out, energy_out);

    // □□
    assert(theta_out == theta_bin);
    assert(energy_out == energy_bin);

    return 0;
}
```

2.5. □□ □ □□ □□

```cpp
#include "core/local_bins.hpp"

int main() {
    // □ 1: □□□/□□□
    int theta_local = 5, E_local = 2, x_sub = 1, z_sub = 3;

    uint16_t lidx = encode_local_idx_4d(theta_local, E_local, x_sub, z_sub);
    printf("□□□ (5,2,1,3) -> %u\n", lidx);

    int t_out, E_out, x_out, z_out;
    decode_local_idx_4d(lidx, t_out, E_out, x_out, z_out);
    printf("□□□ %u -> (%d,%d,%d,%d)\n", lidx, t_out, E_out, x_out, z_out);

    return 0;
}
```

2.6. Highland □□ □□ □□

```cpp
#include "physics/highland.hpp"

int main() {
    // □□ □□□□
    float E = 150.0f;
    float ds = 1.0f;
    float X0 = 360.0f;

    // □□ 1: □□ □□ □□
    float sigma = highland_sigma(E, ds, X0);
    printf("RMS □□ □□: %.4f mrad\n", sigma × 1000);

    // □□ 2: □□ □□ □□□
    unsigned seed = 42;
    float delta_theta = sample_mcs_angle(sigma, seed);

    // □□ 3: □□ □□□□
    float mu = 0.0f;
    float eta = 1.0f;
    update_direction_after_mcs(mu, eta, delta_theta);

    return 0;
}
```

2.7. LUT □□ □□ □□

```cpp
#include "lut/nist_loader.hpp"
#include "lut/r_lut.hpp"

int main() {
    // □□ 1: NIST □□□ □□
    std::string nist_file = "data/pstar_water.txt";
    auto nist_data = load_nist_pstar(nist_file);

    // □□ 2: RLUT □□
    RLUT lut = create_r_lut_from_nist(nist_data, 0.1f, 250.0f, 256);

    // □□ 3: □□□ □□
    float E = 150.0f;
    float range = lut.lookup_R(E);
    printf("%.1f MeV□ □□ □□□: %.2f mm\n", E, range);
```

```
    return 0;
}
```

2.8. □□ □□ □□ □□

```cpp
#include "source/pencil_source.hpp"
#include "core/grids.hpp"
#include "core/psi_storage.hpp"

int main() {
    // □□ 1: □□□ □□
    EnergyGrid e_grid(0.1f, 250.0f, 256);
    AngularGrid a_grid(-90.0f, 90.0f, 512);

    // □□ 2: PsiC □□
    int Nx = 100, Nz = 200;
    float dx = 1.0f, dz = 1.0f;
    PsiC psi(Nx, Nz, 32);

    // □□ 3: □□ □□ □□
    PencilSource source;
    source.x0 = 50.0f;
    source.z0 = 0.0f;
    source.theta0 = 0.0f;
    source.E0 = 150.0f;
    source.W_total = 1.0f;

    // □□ 4: □□ □□
    inject_pencil_source(source, psi, e_grid, a_grid, Nx, Nz, dx, dz);

    return 0;
}
```

2.9. □□□□ □□ □□ □□

```cpp
#include "source/gaussian_source.hpp"
#include "core/grids.hpp"
#include "core/psi_storage.hpp"

int main() {
    // □□ 1: □□□ □□
    EnergyGrid e_grid(0.1f, 250.0f, 256);
    AngularGrid a_grid(-90.0f, 90.0f, 512);

    // □□ 2: PsiC □□
    int Nx = 100, Nz = 200;
    float dx = 1.0f, dz = 1.0f;
    PsiC psi(Nx, Nz, 32);

    // □□ 3: □□□□ □□ □□
    GaussianSource source;
    source.x0 = 50.0f;
    source.sigma_x = 5.0f;
    source.E0 = 150.0f;
    source.sigma_E = 1.0f;
    source.n_samples = 1000;

    // □□ 4: □□ □□
    inject_gaussian_source(source, psi, e_grid, a_grid, Nx, Nz, dx, dz, 42);
```

```
    return 0;
}
```

2.10. ☒☒ ☒☒☒☒☒ ☒☒

```
#include "core/config_loader.hpp"

int main() {
    // □□ 1: □□ □□
    std::string config_file = "configs/default_sim.ini";
    SimulationConfig config = load_config(config_file);

    printf("□□ □□□:\n");
    printf("  □□□: %.1f MeV\n", config.E_mean_MeV);
    printf("  □□□: %d x %d\n", config.Nx, config.Nz);

    // □□ 2: □□□□□ □□
    printf("\n□□□□□ □□ □...\n");
    SimulationResult result = run_simulation_detailed(config);

    // □□ 3: □□ □□
    printf("\n□□□□□ □□:\n");
    printf("  □□ □□: %d\n", result.exit_code);
    printf("  □□: %d\n", result.n_steps_completed);
    printf("  □□: %.2f □\n", result.wall_time_seconds);

    // □□ 4: □□ □□
    printf("\n□□ □□:\n");
    if (result.audit.pass()) {
        printf("  ✓ □□□ □□: □□\n");
        printf("  ✓ □□□ □□: □□\n");
    }

    // □□ 5: □□□ □□ □□
    printf("\n□□□ □□ □□:\n");
    printf("  □□: %.2f mm\n", result.bragg_peak.position_mm);
    if (result.bragg_peak.pass) {
        printf("  ✓ □□□ □□: □□\n");
    }

    return result.exit_code;
}
```

—