QED.jl - A Strong-field particle physics ecosystem

JuliaHEP 2023 Workshop

November 9, 2023 // Uwe Hernandez Acosta

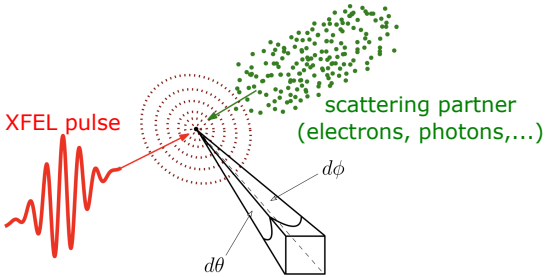
CASUS
CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

www.casus.science

HZDR HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF
TECHNISCHE UNIVERSITÄT DRESDEN
CBG Max Planck Institute of Molecular Cell Biology and Genetics
HELMHOLTZ CENTRE FOR ENVIRONMENTAL RESEARCH – UFZ
Uniwersytet Wrocławski

SPONSORED BY THE
Federal Ministry of Education and Research
STAATSMINISTERIUM FÜR WISSENSCHAFT UND KUNST
Freistaat SACHSEN

Motivation: our physics use-case



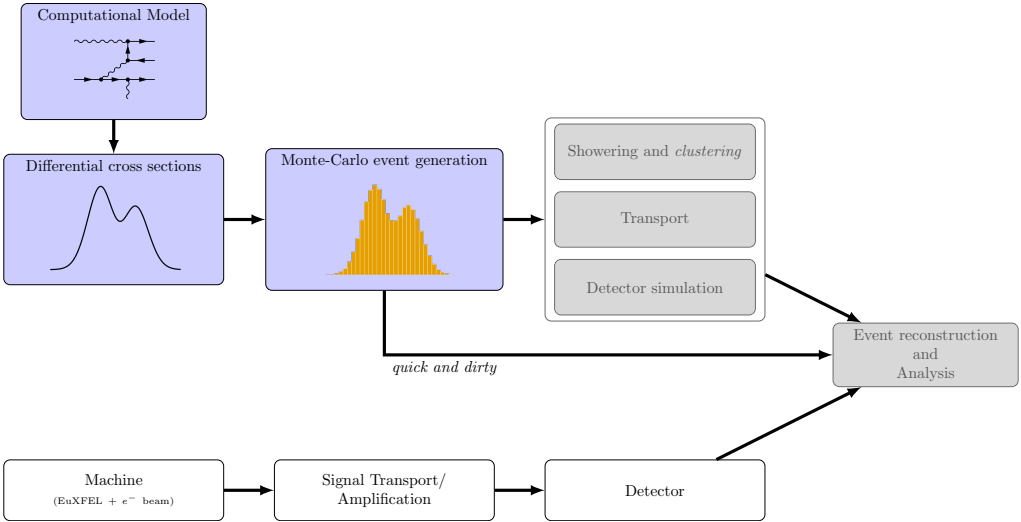
Phenomena

- multi-photon scattering
- non-perturbative effects
- unstable vacuum effects
- electromagnetic cascades

Applications

- Magnetars
- high-luminosity e^-e^+ collider
- Dirac/Weyl semi-metals
- relativistic plasma physics

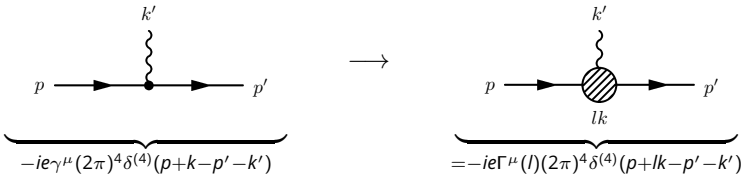
Particle-physics-like simulation workflow



[Stefan Gieseke - MCnet Vietnam summer school (2019)]

Model: strong-field quantum electrodynamics

- Feynman-rule: vertex



- vertex function

$$\Gamma^\mu(l,p,p',k) = \Gamma_0^\mu B_0(l) + \Gamma_1^{\mu\nu} B_{1\nu}(l) + \Gamma_2^\mu B_2(l)$$

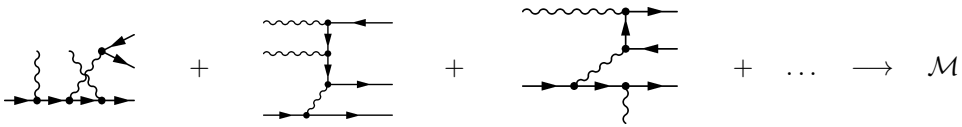
- Phase integrals

$$\left.\begin{matrix} B_0(l) \\ B_1^\mu(l) \\ B_2(l) \end{matrix}\right\} = \int_{-\infty}^{\infty} \mathrm{d}\phi \, \exp(il\phi + iG(\phi)) \left\{\begin{matrix} 1 \\ A^\mu(\phi) \\ A^\mu(\phi)A_\mu(\phi) \end{matrix}\right.$$

[UHA2021 PhD thesis - TU Dresden: ADD PRD]

Differential Cross Sections: $k_1, \ldots, k_M \to p_1, \ldots, p_N$

- Scattering Matrix Element



- Golden Rule

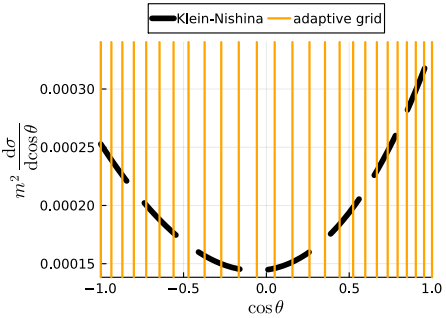
$$\mathrm{d}\sigma = \underbrace{\frac{1}{4\mathcal{I}}}_{\text{incident flux}} \times \underbrace{\sum_{\sigma_1\ldots,\lambda_1\ldots} |\mathcal{M}|^2}_{\text{squared matrix element}} \times \underbrace{\prod_{i=1}^{N} \frac{\mathrm{d}^3 p_i}{(2\pi)^3 2p_i^0} (2\pi)^4 H(k_1, \ldots, k_M, p_1, \ldots, p_N)}_{\text{phase space measure}}$$

$$H(k_1, \ldots, k_M, p_1, \ldots, p_N) = \delta^{(4)}(k_1 + \cdots + k_M - (p_1 + \cdots + p_N)) \times \Theta(\text{cuts})$$

Monte-Carlo Event-Generation: How to generate events

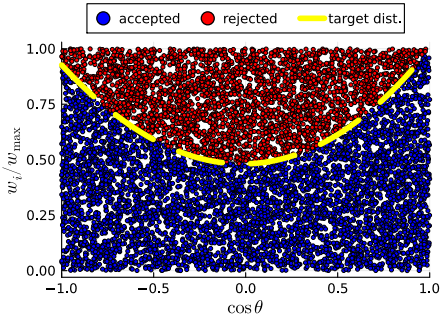
total cross section

$$\sigma \approx \frac{1}{N} \sum_{u \in \mathcal{R}[g]} \underbrace{\frac{\mathrm{d}\sigma/\mathrm{d}u}{g(u)}}_{\sim w_u}$$



sample drawing (unweighting)

$$(u, w_u) \to (\tilde{u}, 1)$$



QED.jl - Strong-field particle physics code

[https://github.com/QEDjl-project]

Requirements

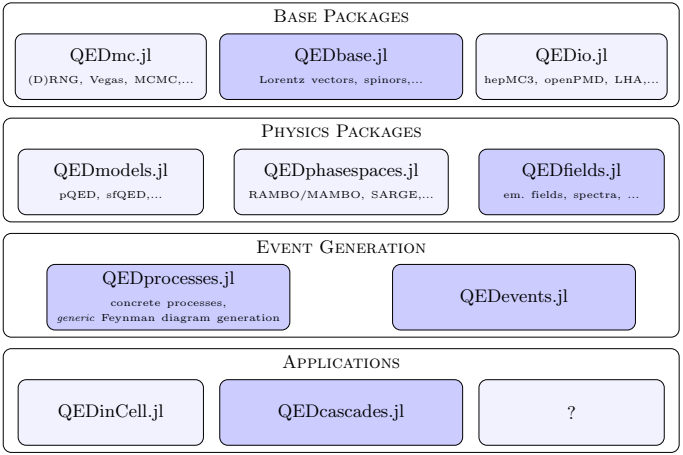
- open source
- user-friendly
- modularised
- extensible
- generic
- performant
- CPU + GPU

QED.jl - Strong-field particle physics code

[https://github.com/QEDjl-project]

Requirements

- open source
- user-friendly
- modularised
- extensible
- generic
- performant
- CPU + GPU

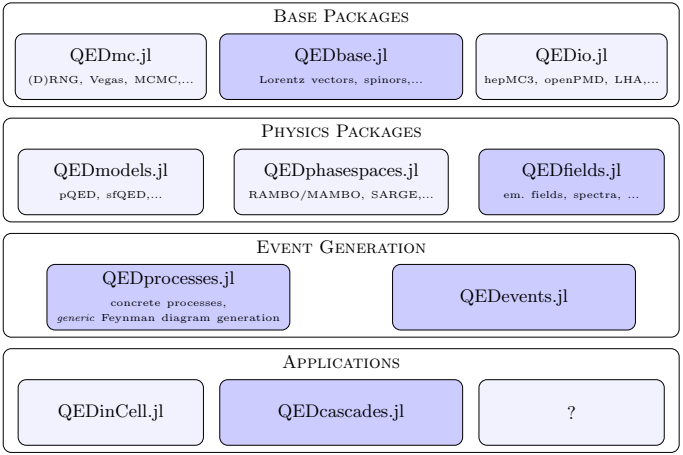


QED.jl - Strong-field particle physics code

[https://github.com/QEDjl-project]

Requirements

- open source
- user-friendly
- modularised
- extensible
- generic
- performant
- CPU + GPU



⟶ **not** restricted to quantum electrodynamics

QEDbase.jl

- Lorentz vectors (based on `StaticArrays.jl`)

```
E, m = 2.0, 1.0
p = SFourMomentum(E, 0, 0, sqrt(E^2 - m^2))
@assert isapprox(p*p, m^2)
```

- Spinors and Dirac matrices (based on `StaticArrays.jl`)

```
u = IncomingFermionSpinor(p,m)
u_bar = OutgoingFermionSpinor(p,m)
sp_prod = [u_bar(i)*(slashed(p)*u(j)) for i in 1:2 for j in 1:2]
@assert isapprox(  sum(sp_prod)  , 4*m^2)
```

- Particle types and `Incoming/Outgoing`

```
electron_charge = charge(Electron())
@assert electron_charge = -1
electron_state = base_type(Electron(),Incoming(),p, SpinUp())
@assert isapprox( electron_state, u(1))
```

`QEDfields.jl` – Electromagnetic fields

- Base type: `AbstractField`

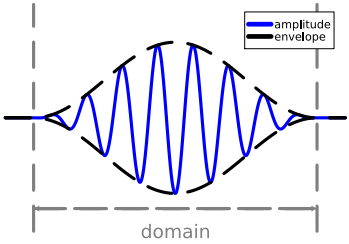
```
domain(::AbstractField)
reference_momentum(::AbstractField)
_amplitude(::AbstractField)
```

- Base type: `AbstractPPWField <: AbstractField`

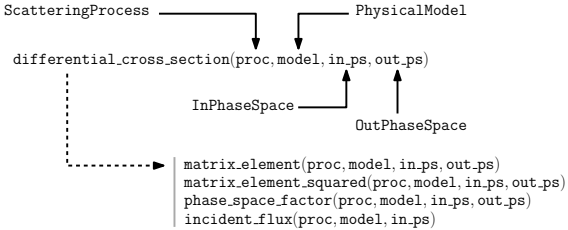
```
phase_duration(::AbstractPPWField)
phase_envelope(::AbstractPPWField)
```

- generic implementations

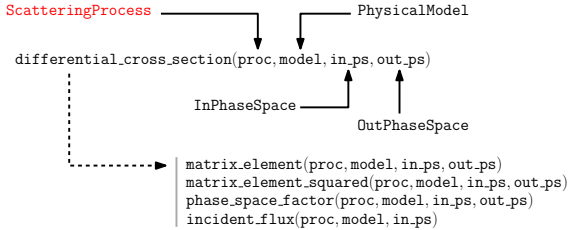
```
amplitude(::AbstractField, ... )
fourier_transform(::AbstractField, ... )
...
```



QEDprocesses.jl



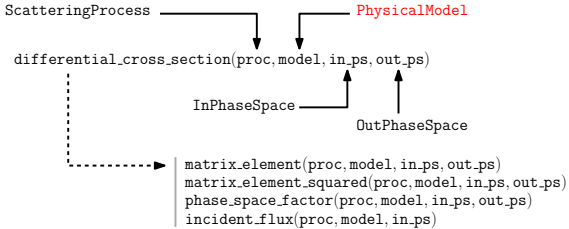
QEDprocesses.jl



- base type:
  `AbstractScatteringProcess`
- interface function
  `incoming_particles`
  `outgoing_particles`
  `spins`
  `polarizations`

```
julia> Compton(PolX(),SpinUp(),AllPol(),AllSpin())
Compton{PolarizationX, SpinUp, AllPolarization, AllSpin}()
```

QEDprocesses.jl



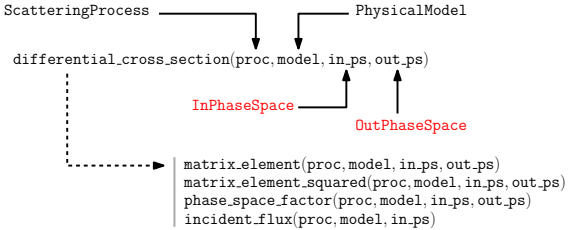
- base type:
  `AbstractPhysicalModel`
- interface function
  `fundamental_interaction`

  `in_ps_dim(proc, model)`
  `out_ps_dim(proc, model)`
  `matrix_element`

```
julia> model = PerturbativeQED()
PerturbativeQED()
```

```
julia> field = GaussianField()
GaussianField()

julia> model = StrongFieldQED(field)
StrongFieldQED{GaussianField}()
```

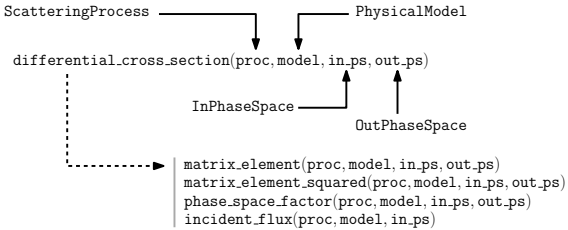
QEDprocesses.jl



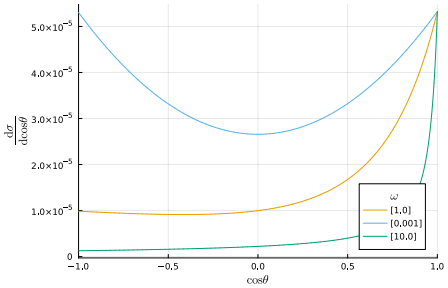
- base type

```
AbstractVector{T}
AbstractMatrix{T}
where {T<:AbstractFourMomentum}
```

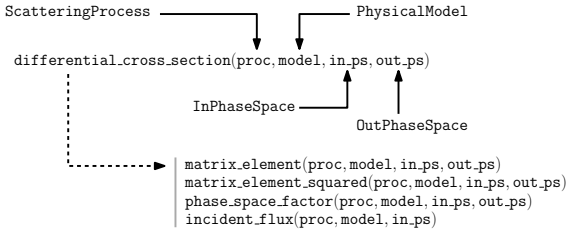
QEDprocesses.jl



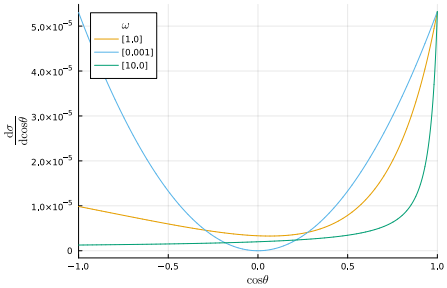

```
julia> proc = Compton(AllPol(),AllSpin(),AllPol(),AllSpin())
Compton{AllPolarization, AllSpin, AllPolarization, AllSpin}()

julia> model = PerturbativeQED()
PerturbativeQED()

julia> omega = [1.0 1e-3;]
1×2 Matrix{Float64}:
 1.0  0.001

julia> cth_arr = range(-1,1,length=10_000)
-1.0:0.0002000200020002003:1.0

julia> phi_arr = [0.0,]
1-element Vector{Float64}:
 0.0

julia> final_PS = collect_batch(cth_arr,phi_arr);

julia> diffCS = differential_cross_section(proc,model,omega,final_PS);
```



QEDprocesses.jl




```
julia> proc = Compton(PolX(),AllSpin(),AllPol(),AllSpin())
Compton{PolarizationX, AllSpin, AllPolarization, AllSpin}()

julia> model = PerturbativeQED()
PerturbativeQED()

julia> omega = [1.0 1e-3;]
1×2 Matrix{Float64}:
 1.0  0.001

julia> cth_arr = range(-1,1,length=10_000)
-1.0:0.0002000200020002:1.0

julia> phi_arr = [0.0,]
1-element Vector{Float64}:
 0.0

julia> final_PS = collect_batch(cth_arr,phi_arr);

julia> diffCS = differential_cross_section(proc,model,omega,final_PS);
```



Interlude: the computation setup interface

$$f(\underbrace{u_1, \ldots, u_n}_{\text{input vector}}; \underbrace{\lambda_1, \ldots, \lambda_m}_{\text{parameters}}) = \underbrace{R}_{\text{output}}$$

- base type: `AbstractComputeSetup`
- interface functions

```
_assert_valid_input(stp,input)

_compute(stp, input)

_post_processing(stp, input,
result)
```

- default implementation

```
compute(stp, input)
```

- conceivable input

```
SVector/SMatrix
Vector/Matrix
CuArray
ROCArray
...
```

QEDevents.jl

- General sampler
  base type: `AbstractSampler`

```
Base.eltype(sampler)
setup(sampler)
weight(sampler,x)
Distributions._rand!(rng,sampler, out)
```

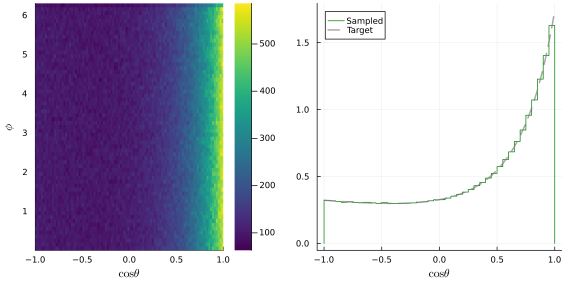
- Rejection sampler

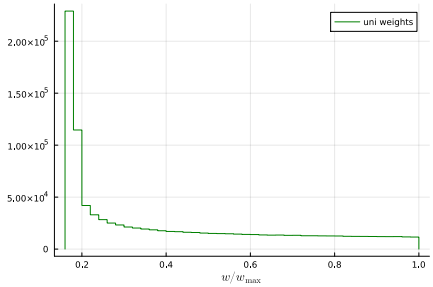
```
RejectionSampler<:AbstractSampler

...
proposal(reject_sampler)
```

- Proposal sampler
  - Uniform
  - Vegas
  - NIS
  - MCMC
  - ...
- Process sampler
  - Kahn
  - Koblinger
  - Özmutlu
  - KEK_BW
  - FieldSampler
  - ...

Event Generation

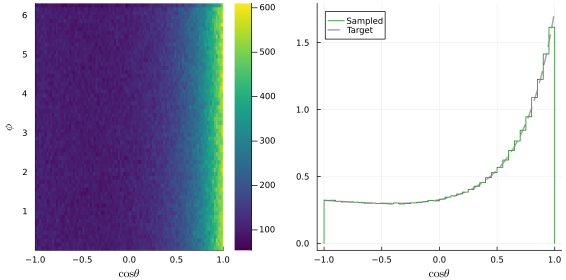
```
proc = Compton() # default: all spins/pols summed
model = PerturbativeQED()
in_ps = [1.0; ] # omega = 511 keV
stp = DifferentialCrossSection(proc, model, in_ps)
sampler = RejectionSampler(stp) # default: uniform proposal
samples = rand(sampler,1000_000)

plot_combine(stp,samples)
```

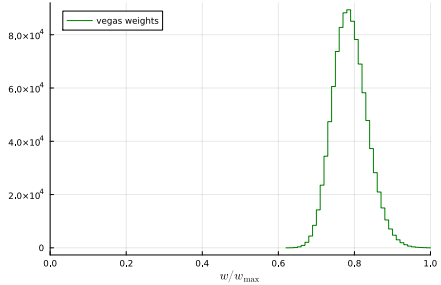




Event Generation

```
proc = Compton() # default: all spins/pols summed
model = PerturbativeQED()
in_ps = [1.0; ] # omega = 511 keV
stp = DifferentialCrossSection(proc, model, in_ps)
proposal = VegasSampler(stp)
sampler = RejectionSampler(stp, proposal)
samples = rand(sampler,1000_000)

plot_combine(stp,samples)
```





Instead of a summary: lessons learned

- Theory
  - ⇒ there is something besides SM physics
  - ⇒ dynamical Feynman rules are hard to model
- `QED.jl`
  - ⇒ easy interfaces are hard to design
  - ⇒ orthogonal design is mandatory
  - ⇒ Julia is your friend
- Event generation
  - ⇒ good proposals beat bad proposals
  - ⇒ good implementations beat good proposals
  - ⇒ knowing your problem beats everything

- Collaborators

Michael Bussmann
Klaus Steiniger
Simeon Ehrig
Tom Jungnickel
Anton Reinhard

Prof. Burkhard Kämpfer
René Widera

Prof. Thomas Cowan
Michal Smid
Toma Toncian

- Acknowledgements:

F. Siegert (TUD)

A. Cangi (CASUS)
Lokamani (CASUS)

Prof. W. Nagel (TUD)
Prof. J. Castrillon (TUD)