Simulation of segmented HPGe detectors in

Lukas Hauertmann ⟨lhauert@mpp.mpg.de⟩, Oliver Schulz ⟨oschulz@mpp.mpg.de⟩,
Martin Schuster ⟨schuster@mpp.mpg.de⟩, Anna Julia Zsigmond ⟨azsigmon@mpp.mpg.de⟩

LEGEND Analysis Workshop Meeting, Dec 1st, 2018

Detector Definition

In [6]:
```
detector = SolidStateDetector(SSD_examples[:BEGe])
```

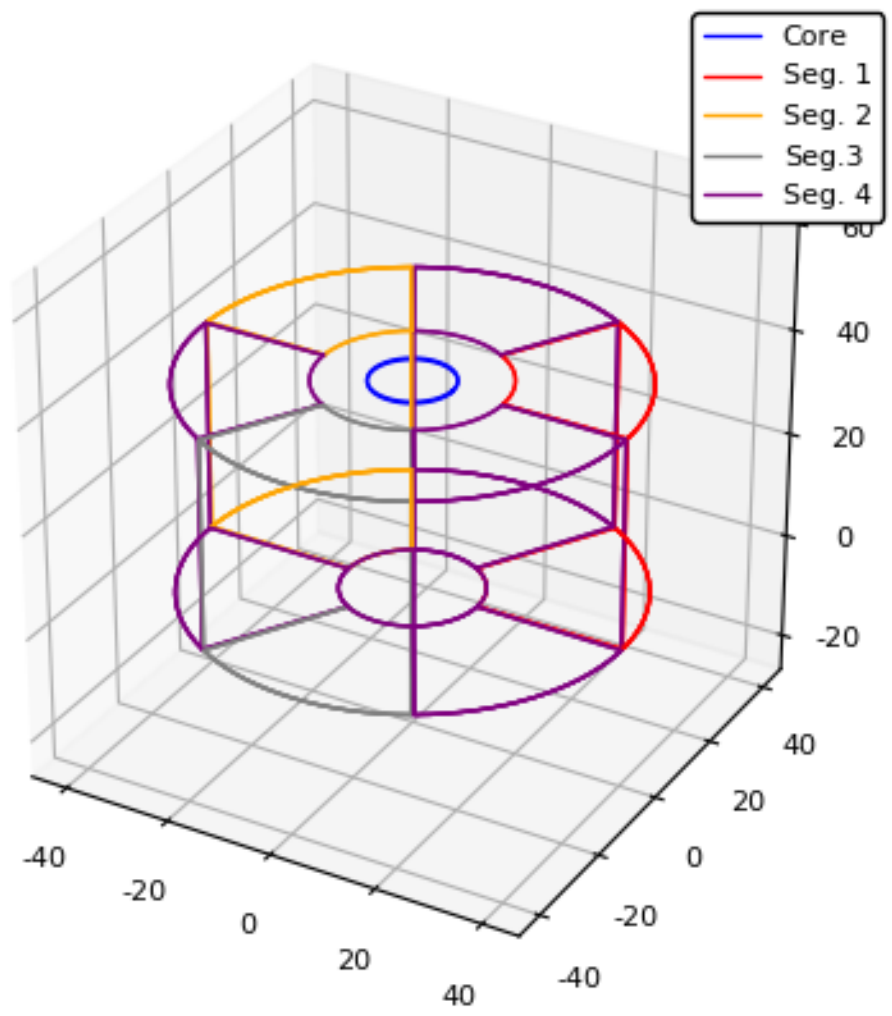
```
BEGe
________ExampleSegmentedBEGe________

---General Properties---
Detector Material:        High Purity Germanium
Environment Material:     Vacuum
Bulk type:                ntype
```

Out[6]:

```
Core Bias Voltage:        4500.0 V
Mantle Bias Voltage:      0.0 V

---Geometry---
Outer Crystal Dimensions:
Crystal length:           40.0 mm
Crystal diameter:         80.0 mm
```

In [7]:
```
plot(detector)
```

Out[7]:



Electric Potential Calculation

To simulate charge drift, we need the E-field. For the E-field, we need E_{pot}. Simple physics problem:

$$(\frac{\partial}{\partial x^2} + \frac{\partial}{\partial x^2} + \frac{\partial}{\partial x^2})E_{pot} = \frac{\rho(x, y, z)}{\epsilon_0}$$

Relaxation algorithm (simplified):

- Discretize problem by creating a grid of spacial points
- For each point:
    - Determine potential divergence numercially
    - Compare with $\rho(x, y, z)/\epsilon_0$
    - Apply correction to potential value of current point
    - Repeat until converged
- Challenge: Achieve numerical stability and fast convergence

Choosing a smart grid and strategy

- Red-black grid: Update only odd or even points in each cycle
- Use over-relaxation for faster convergence

In [9]:
```
rb_plot
```

Out[9]:

3D Electrical Potential Calculation

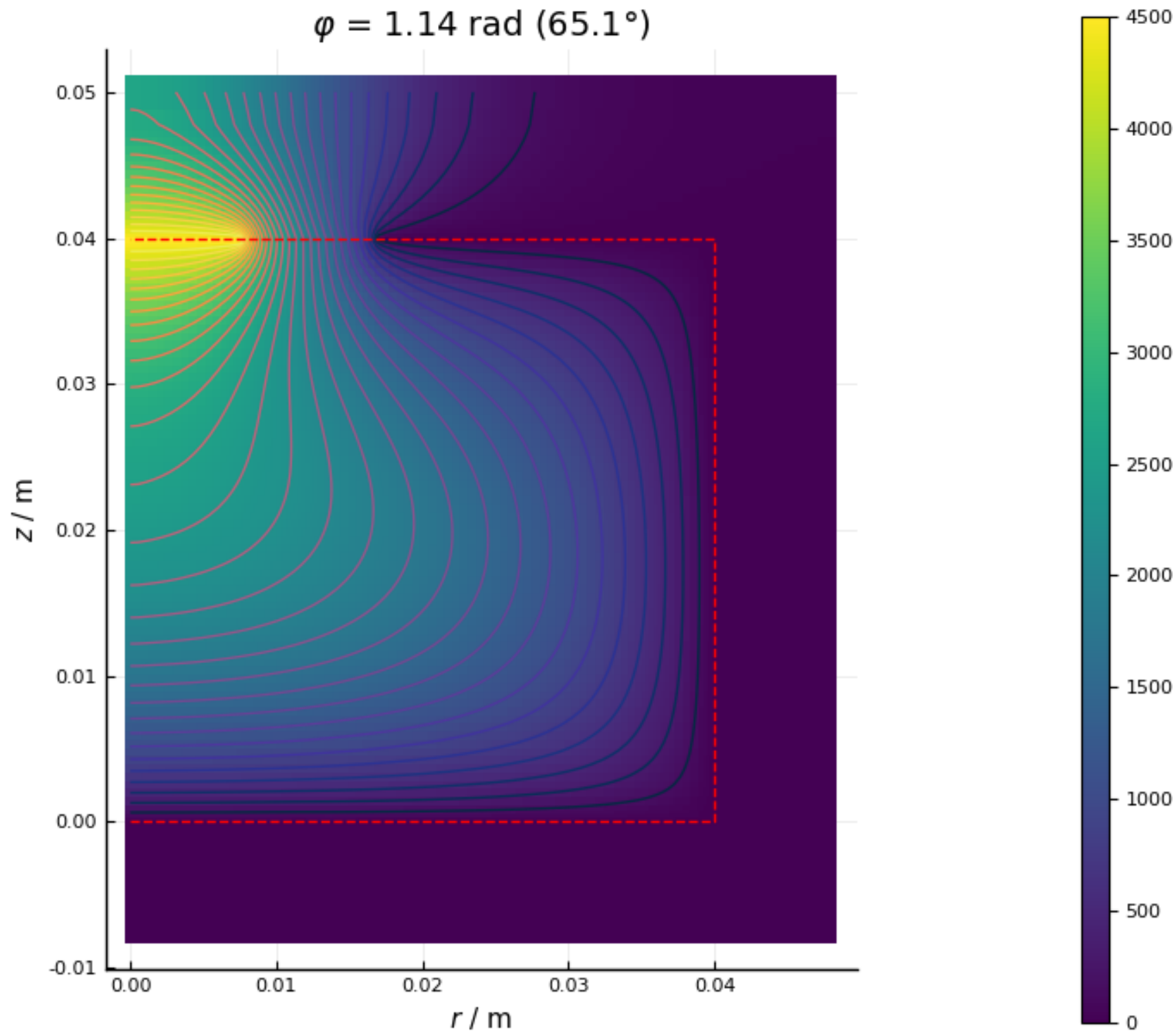
- Full 3D potential takes longer than in 2D
- But: Can now store and load potentials (HDF5)

In [10]:
```
E_pot, point_types = if !isfile("detector-potentials-segbege.h5")
    println("Calculating electrical potential, this may take a while ...")
    E_pot, point_types = calculate_electric_potential(detector,convergence_limit=1e-
5,max_refinements=2)
else
    println("Reading electrical potential from file.")
    HDF5.h5open("detector-potentials-segbege.h5", "r") do input
        CylindricalGrid(readdata(input, "detpot/E_pot")),
        PointTypes(readdata(input, "detpot/pointflags"))
    end
end;
```

```
Reading electrical potential from file.
```

In [98]:
```
plot(E_pot, φ=1.12, contours_equal_potential=true, levels=30, size = (800, 600))
plot!(detector, :plane, φ=0)
```

Out[98]:



Simulating Undepleted Detectors

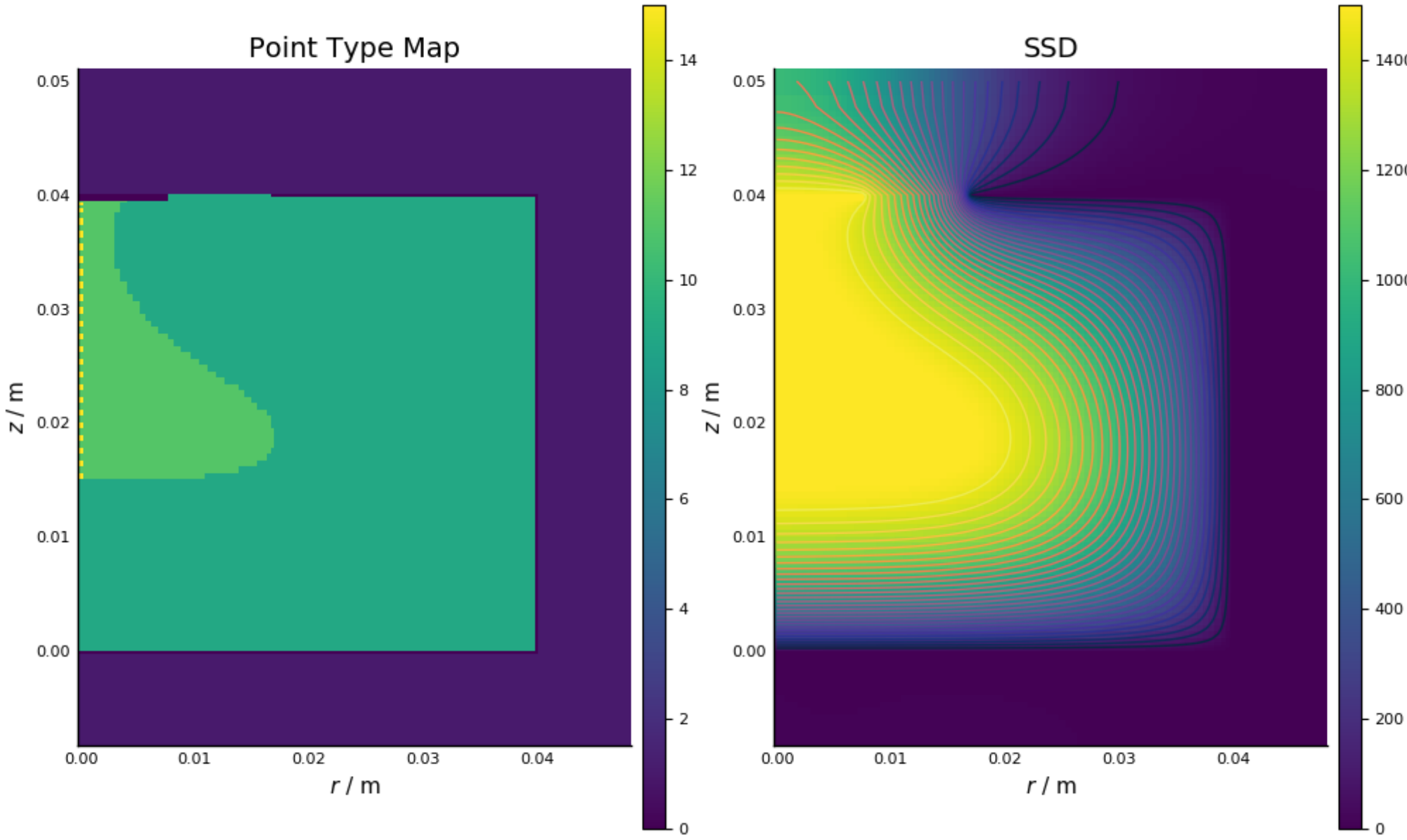
In [80]:
```
detector_undep = deepcopy(detector)
detector_undep.segment_bias_voltages[1] = 1500;
```

In [97]:
```
E_pot_undep, point_types_undep = calculate_electric_potential(
    detector_undep, depletion_handling=true,
    convergence_limit=1e-5,max_refinements=2
);
```

```
Electric Potential Calculation
Bulk type: ntype
Bias voltage: 1500.0 V
φ symmetry: cyclic = 120.0° -> just calculating 1/3 of the detector in φ.
Precision: Float32
Convergence limit: 1.0e-5 => 0.015 V
Threads: 4
Refine? -> true
Refinement parameters:
        maximum number of refinements: 2
        minimum grid spacing:
                r: 0.0001 m
                φ: 0.01 rad
                z: 0.0001 m
        Refinement limits:
                r: 0.0001 -> 0.15 V
                φ: 0.0001 -> 0.15 V
                z: 0.0001 -> 0.15 V

Convergence:  (thresh = 0.015, value = 61.712)
```

In [101]:
```
p_SSD = plot(E_pot_undep, title="SSD", contours_equal_potential=true,φ=deg2rad(40),
levels=30)
p_pts = plot(point_types_undep, title="Point Type Map",φ=deg2rad(40))
plot(p_pts, p_SSD, layout=(@layout [a b]), size=(1000,600))
```

Out[101]:



Next Step: Calculate Electric Field from Potential

In [17]:
```
E_field = SolidStateDetectors.get_electric_field_from_potential(E_pot);
```

In [85]:
```
plot(E_field, detector, E_pot, size = (800, 600));
plot!(detector, :plane)
```

Out[85]:

Calculate Drift Fields

- Precalculation of drift vector fields in increases performance of charge drift

In [19]:
```
drift_model = ADLChargeDriftModel(); # drift_model_vacuum = VacuumChargeDriftModel
();
```

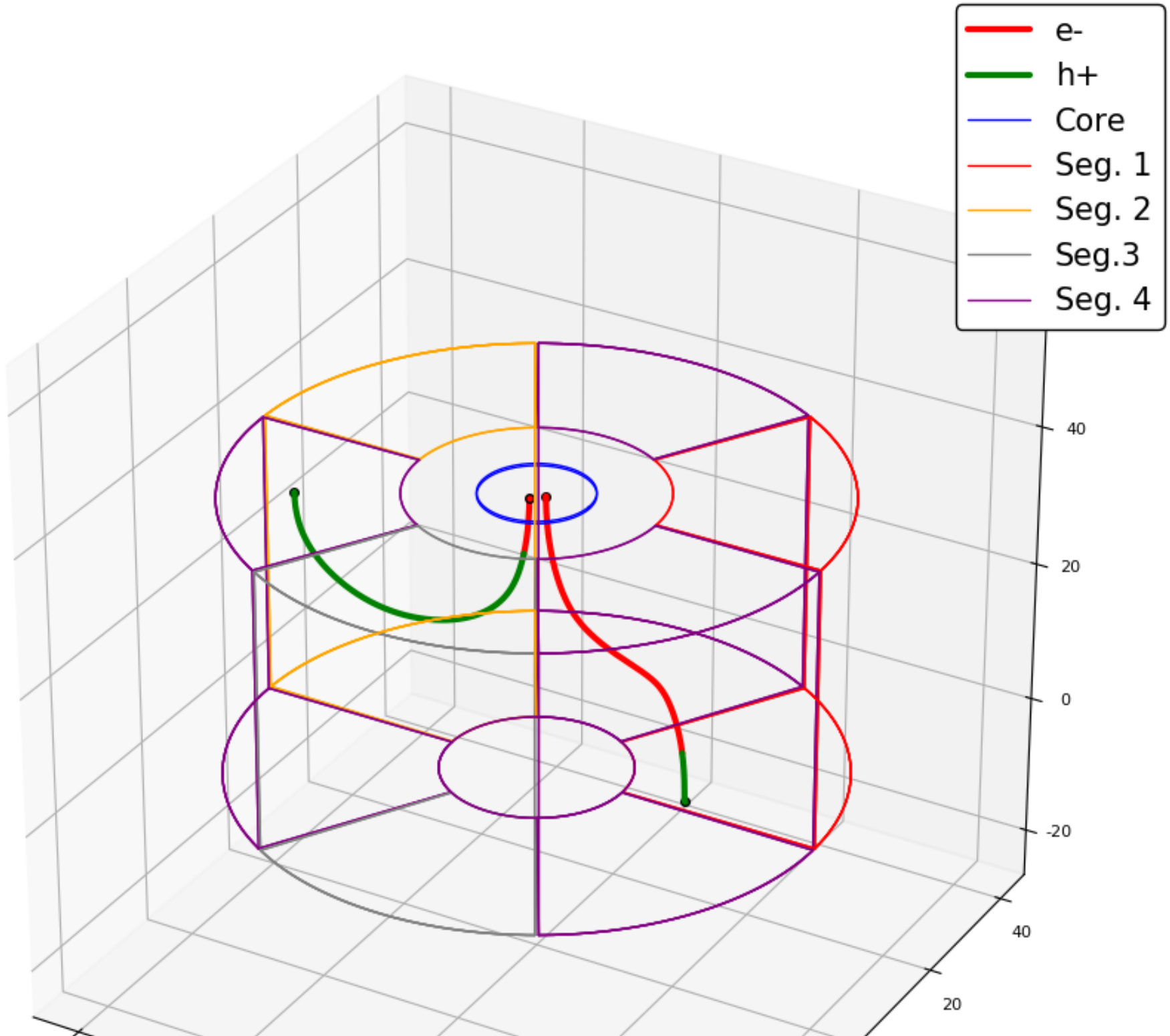
In [20]:
```
electron_drift_field = get_electron_drift_field(E_field, drift_model);
hole_drift_field = get_hole_drift_field(E_field, drift_model);
```

In [21]:
```
# Interpolate
electron_drift_field_interpolated = SolidStateDetectors.get_interpolated_drift_field
(electron_drift_field, E_pot);
hole_drift_field_interpolated = SolidStateDetectors.get_interpolated_drift_field(hol
e_drift_field, E_pot);
```

In [23]:
```
T = Float32
pos = [SVector{3,T}(-0.0007,- 0.002,0.032),SVector{3,T}(0.02, 0.002,0.007)]
drift_paths=SolidStateDetectors.drift_charges(detector, pos, electron_drift_field_in
terpolated, hole_drift_field_interpolated);
```

In [24]:
```
plot(drift_paths, scaling=1e3)
plot!(detector,size = (900,900))
```

Out[24]:



Weighting Potential Calculation

- Detector has a point contact and 4 segments - takes a few minutes
- But: Can now store to and load from HDF5:

In [25]:
```
wps = if !isfile("detector-potentials-segbege.h5")
    println("Calculating weighting potentials, this may take a while ...")
    [SolidStateDetectors.calculate_weighting_potential(detector, i, verbose=false,ma
x_refinements=4, max_n_iterations=500,convergence_limit=1e-5) for i in 1:5]
else
    println("Reading weighting potentials from file.")
    wps = HDF5.h5open("detector-potentials-segbege.h5", "r") do input
        [CylindricalGrid(readdata(input, "detpot/W_pot/$i")) for i in 1:detector.n_t
otal_contacts]
    end
end;
```

```
Reading weighting potentials from file.
```

Software Documentation

- Still needs work, but:

In [28]:
```
?calculate_weighting_potential
```

```
search: calculate_weighting_potential
```

Out[28]:
```
calculate_weighting_potential(det::SolidStateDetector, channels::Array{Int, 1};
 <keyword arguments>)::Grid
```

Compute the weighting potential of the `channels` of the given Detector `det` on an adaptive grid through successive over relaxation. `channels` is a list of the channels which are fixed to 1. All other channels are fixed to 0.

There are serveral `<keyword arguments>` which can be used to tune the computation:

Keywords

- `coordinates::Symbol`: the kind of the coordinate system of the grid. Right now only `:cylindrical` is possible.
- `convergence_limit::Real`: `convergence_limit` times the bias voltage sets the convergence limit of the relaxation. The convergence value is the absolute maximum difference of the potential between two iterations of all

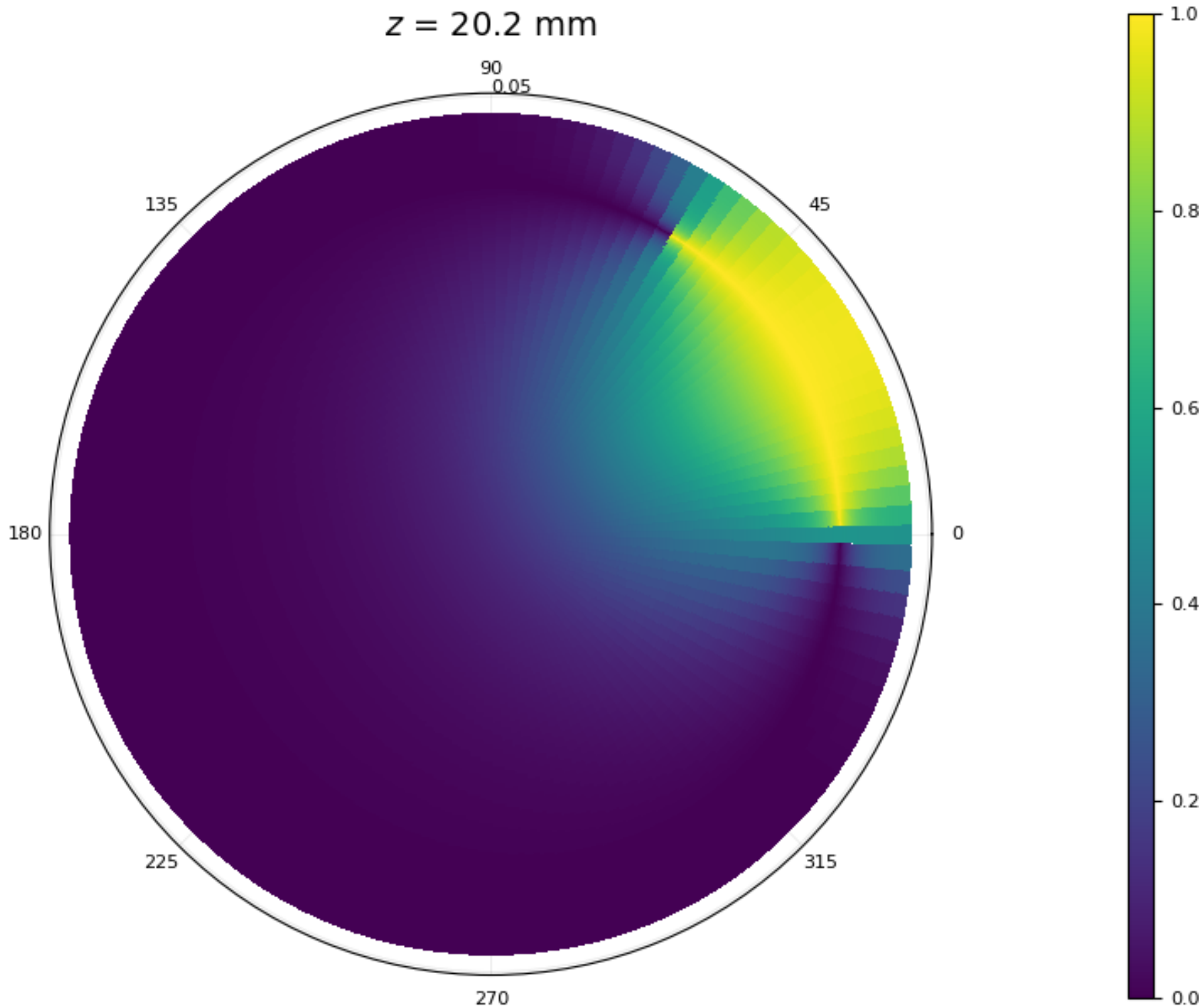
Segment Weights

In [89]:
```
seg_wpots_plot
```

Out[89]:

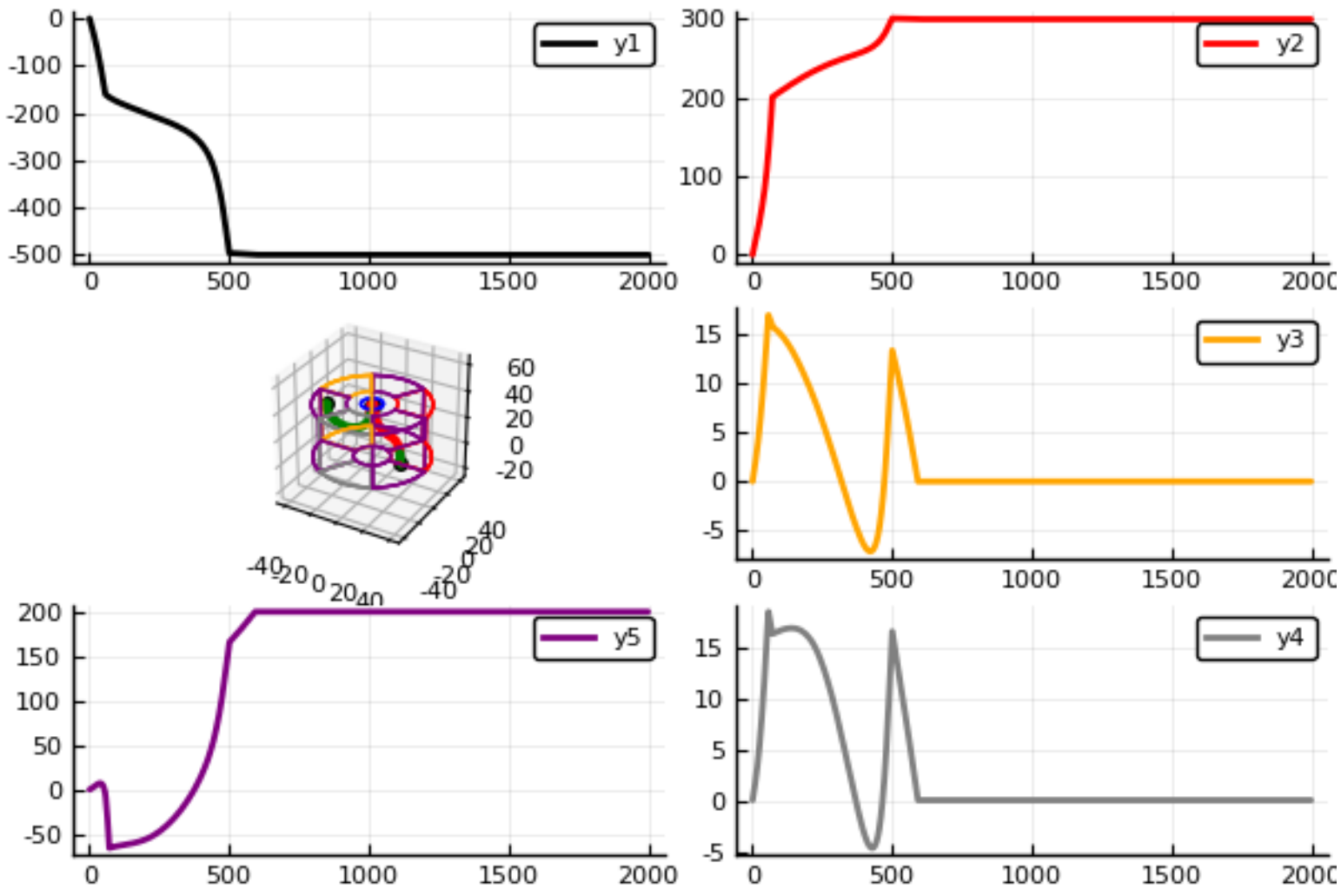
In [90]:
```
plot(wps[2], z = 0.02, size = (800, 600))
```

Out[90]:



In [75]:
```
seg_pulses_plot
```

Out[75]:



Let's load some MC data (from HDF5)

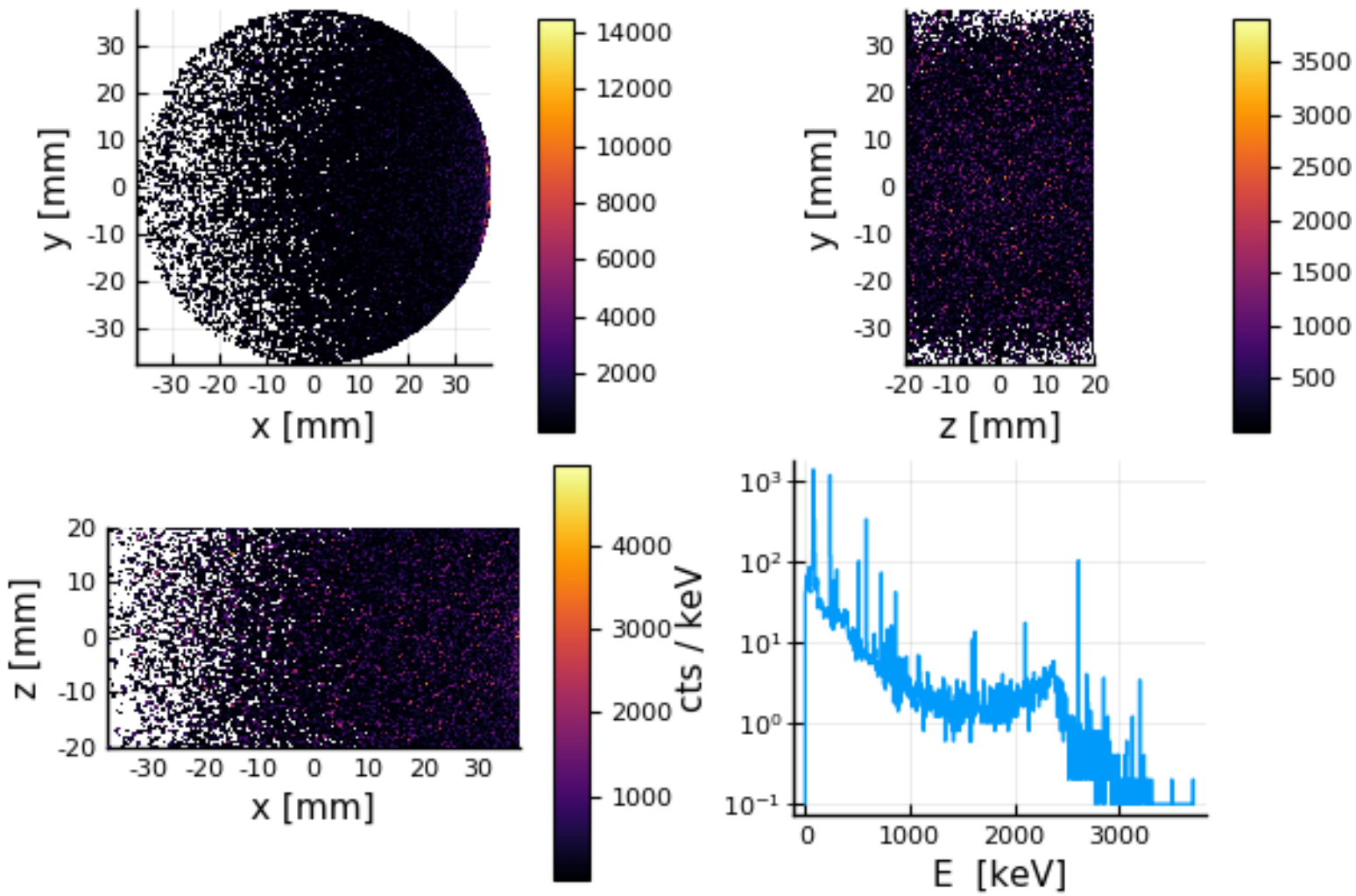
In [36]:
```
mctruth_filename = "seg-bege-mctruth.h5"
mc_events = @time HDF5.h5open(mctruth_filename, "r") do input
    readdata(input, "mctruth")
end
println("$(length(flatview(mc_events.edep))) original hits")

mc_events_clustered = @time cluster_detector_hits(mc_events, 0.2u"mm")
println("$(length(flatview(mc_events_clustered.edep))) clustered hits")
```

```
  1.226765 seconds (3.08 M allocations: 221.108 MiB, 7.05% gc time)
517456 original hits
  4.050771 seconds (16.96 M allocations: 1.114 GiB, 16.82% gc time)
81097 clustered hits
```

Hit Distribution

In [37]:
```
plot(mc_events_clustered)
```

Out[37]:

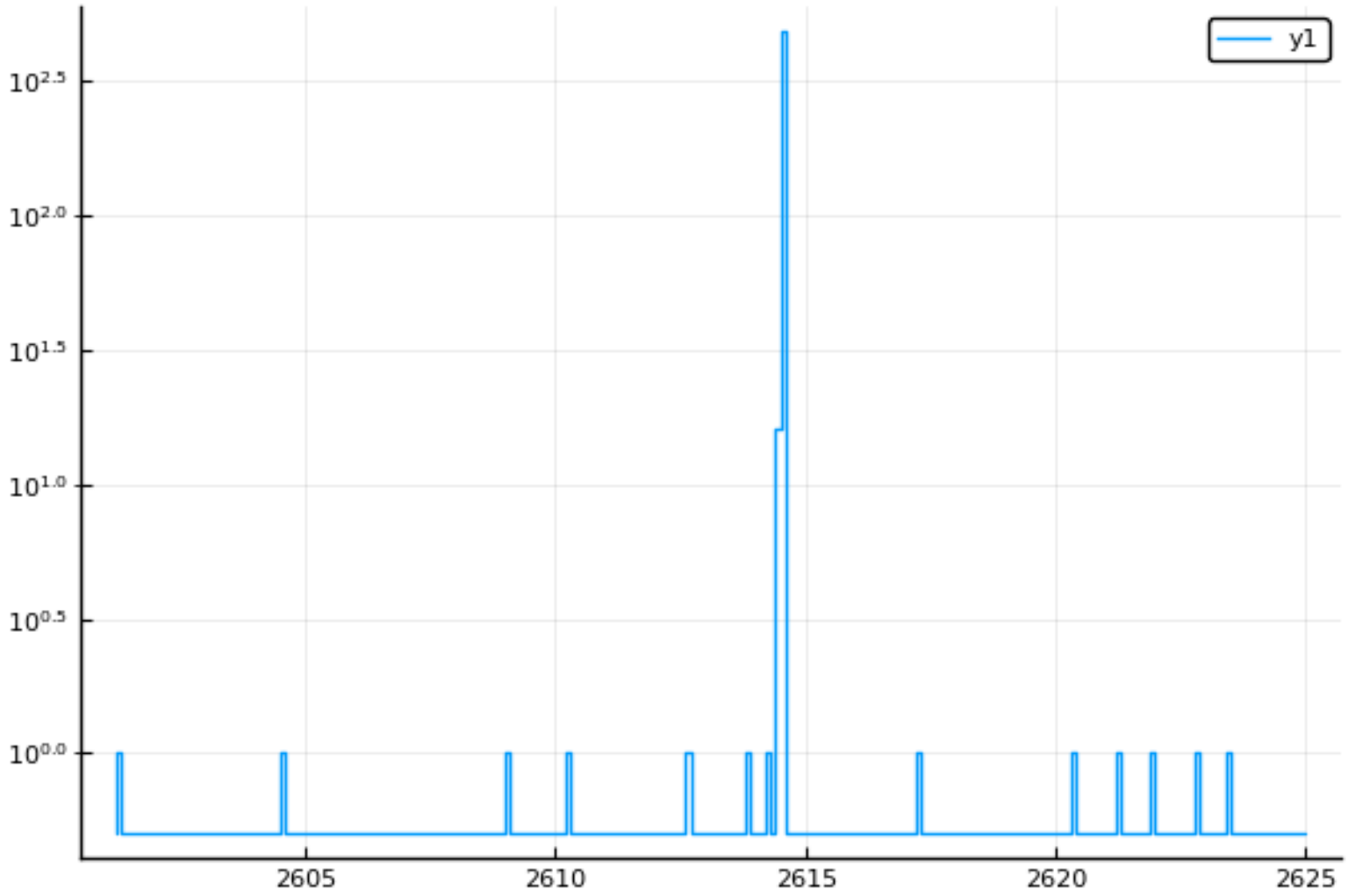


Raw MC Spectrum

- Not very realistic yet ...

In [38]:
```
stephist(ustrip.(sum.(mc_events_clustered.edep)), bins = 2600:0.1:2625, yscale = :log10)
```

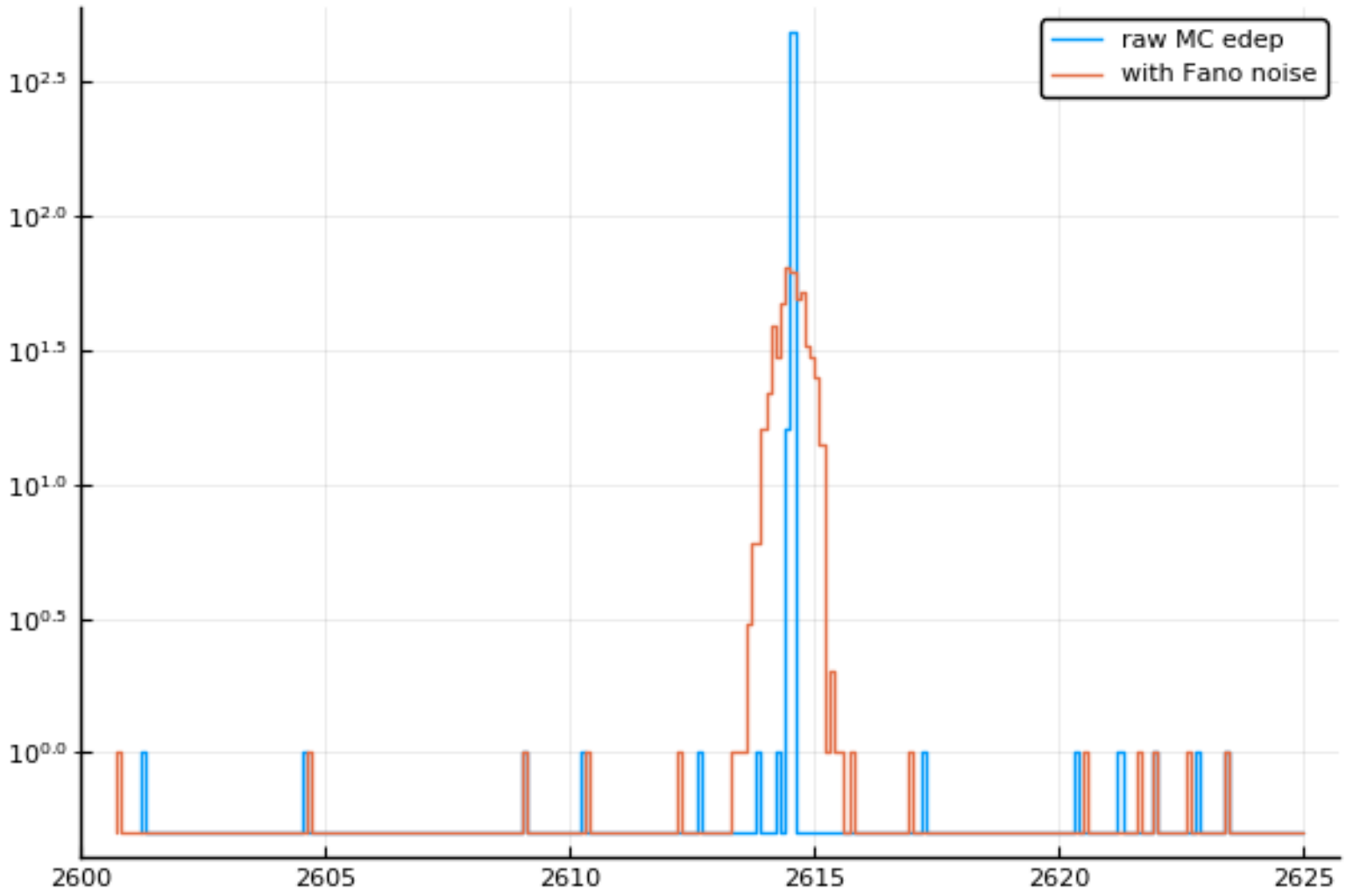
Out[38]:



More Realism: Add Fano noise

In [39]:
```
det_material = detector.material_detector
mc_events_fnoise = add_fano_noise(mc_events_clustered, det_material.E_ionisation, de
t_material.f_fano)
stephist(ustrip.(sum.(mc_events_clustered.edep)), bins = 2600:0.1:2625, label = "raw
 MC edep", yscale = :log10)
stephist!(ustrip.(sum.(mc_events_fnoise.edep)), bins = 2600:0.1:2625, label = "with
 Fano noise", yscale = :log10)
```

Out[39]:



FIlter out hits outside of detector

In [50]:
```
filtered_events = mc_events_fnoise[findall(pts -> all(p -> p in detector, pts), mc_e
vents_fnoise.pos)]
```

Out[50]:

Table with 5 columns and 16856 rows:

	evtno	detno	thit	edep	…
1	3	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
2	5	[1, 1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
3	7	[1, 1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
4	10	[1, 1, 1, 1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
5	61	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
6	69	[1, 1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
7	81	[1, 1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
8	95	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
9	119	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
10	135	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
11	143	[1, 1, 1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
12	157	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
13	159	[1, 1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
14	166	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
15	194	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
16	219	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
17	222	[1, 1, 1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
18	233	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
19	234	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
20	243	[1, 1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
21	244	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
22	302	[1, 1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
23	316	[1]	Quantity{Float64,**T**,…	Quantity{Float64,**L**^…	⋯
⋮	⋮	⋮	⋮	⋮	⋱

Generate Pulses for all MC Events

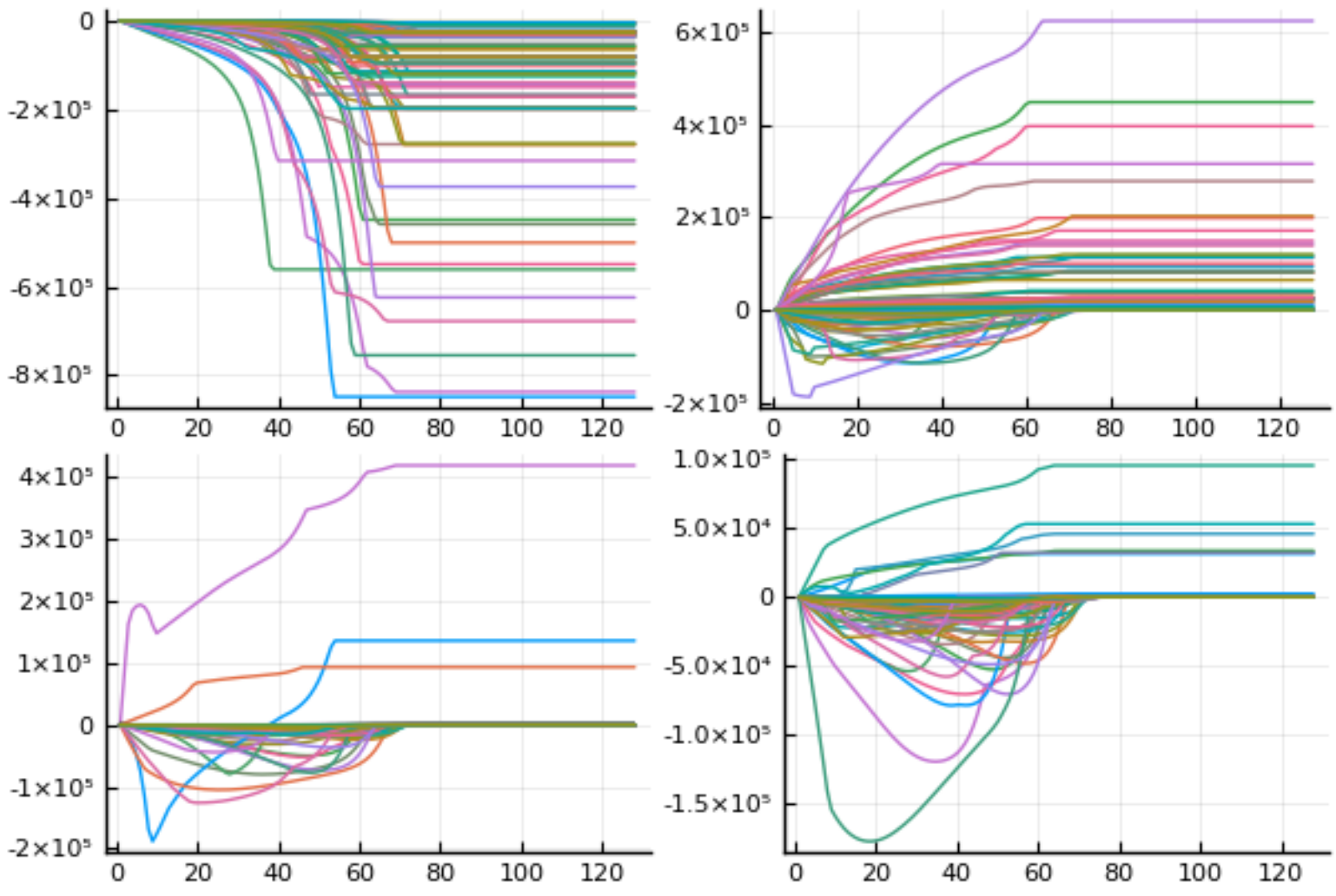
In [52]:
```
delta_t = 10u"ns"
contact_charge_signals, drift_times = @time generate_charge_signals(
    detector, electron_drift_field_interpolated, hole_drift_field_interpolated, wps_
interp,
    filtered_events,
    128, delta_t
);
```

```
  8.866608 seconds (15.11 M allocations: 328.088 MiB, 2.63% gc time)
```

Let's take a look at the pulses

In [53]:
```
plot([plot(contact_charge_signals[i][1:100], legend = false) for i in 1:4]...)
```

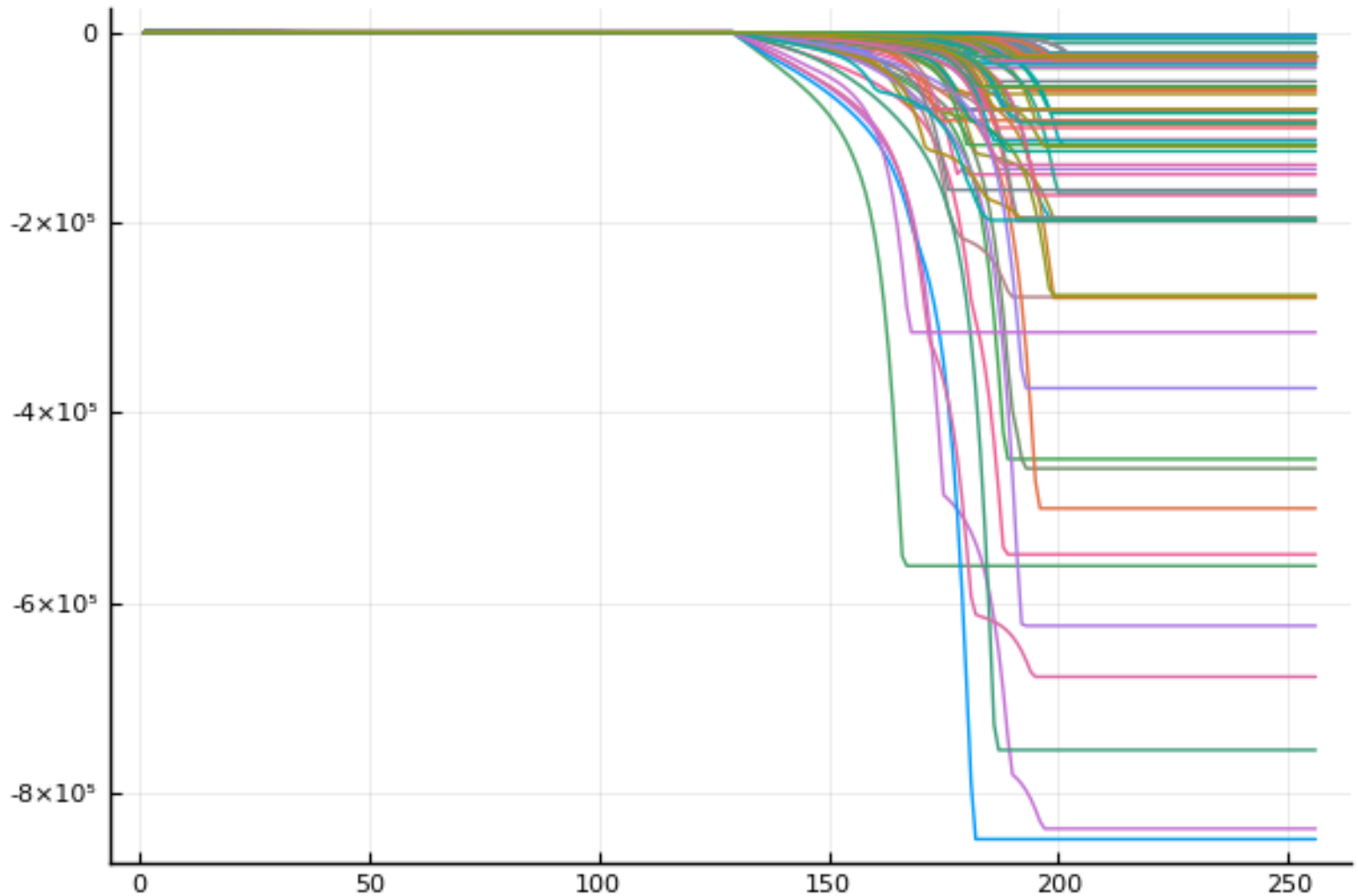
Out[53]:



Adding a Pre-Pulse Baseline

In [54]:
```
core_signals = contact_charge_signals[1]
signals_with_baseline = nestedview(vcat(fill!(similar(flatview(core_signals)), 0), f
latview(core_signals)))
plot(signals_with_baseline[1:100], legend = false)
```

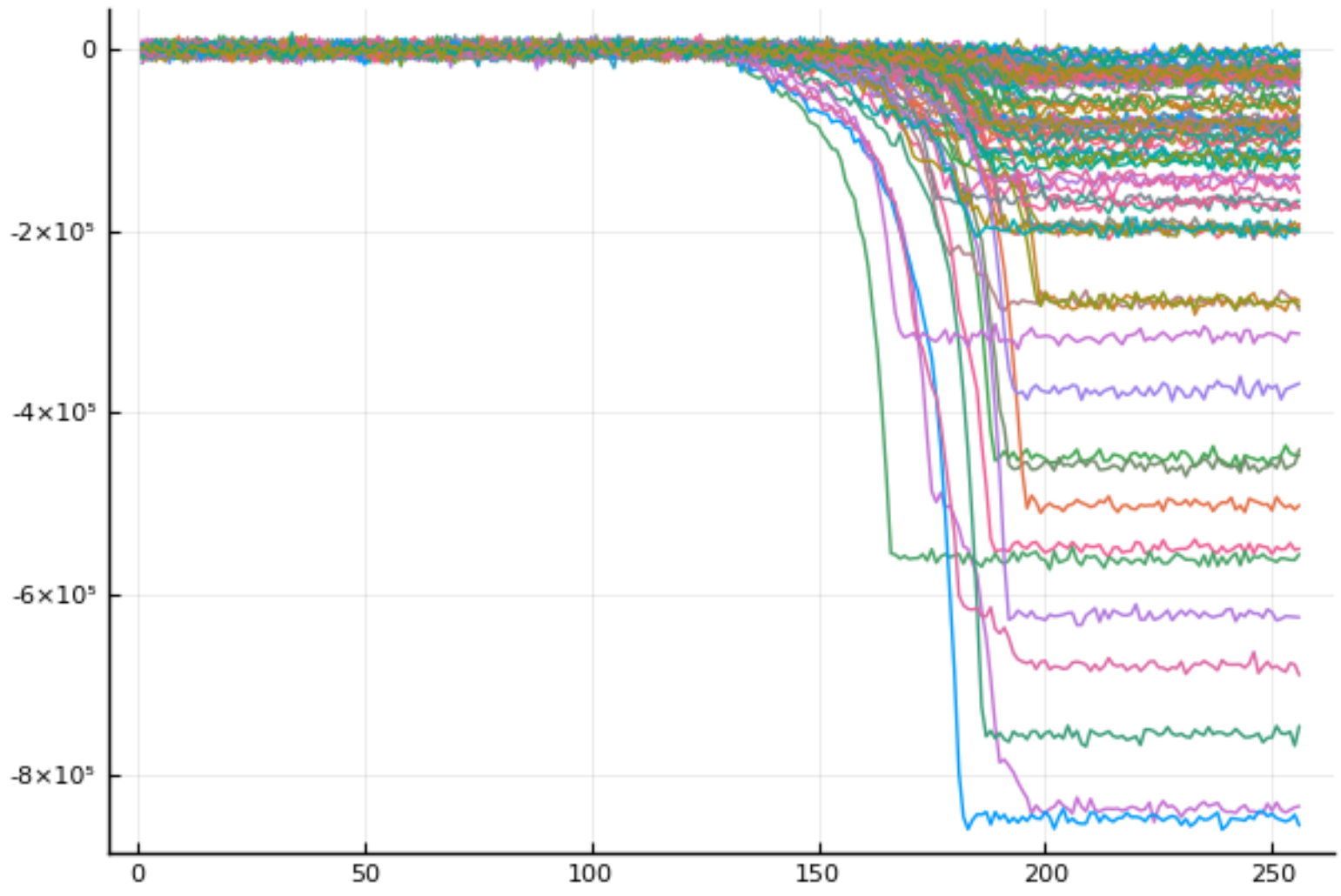
Out[54]:



Electronics are Noisy

In [95]:
```
noise = rand!(Normal(0,5e3), similar(flatview(signals_with_baseline)))
noisy_signals = nestedview(flatview(signals_with_baseline) .+ noise)
plot(noisy_signals[1:100], legend = false)
```

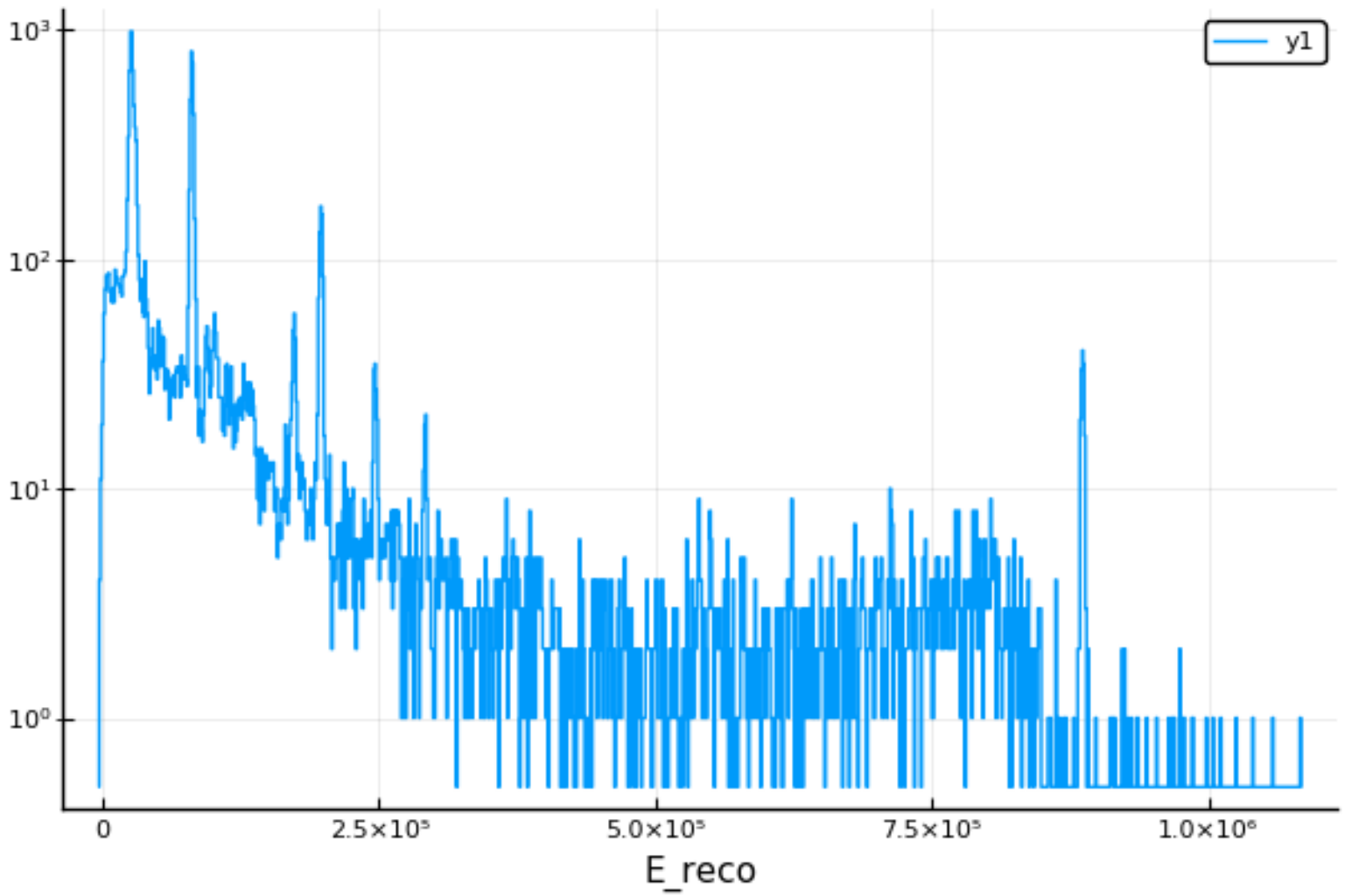
Out[95]:



(Simple) Energy Reconstruction on Simulated Pulses

In [96]:
```
pre_pulse_mean = vec(mean(-flatview(noisy_signals)[1:30, :], dims = 1))
post_pulse_mean = vec(mean(-flatview(noisy_signals)[230:255, :], dims = 1))
E_reco = post_pulse_mean .- pre_pulse_mean
stephist(E_reco, nbins = 1000, yscale = :log10, xlabel = "E_reco")
```

Out[96]:



Simulation Verification and Detector Bulk Behaviour

- Simulation looks nice, but is it real?
- Surface scans provide part of the answer (drift times, etc.)
- But how to "look inside" of the detector?
- Common technique: 90° Compton-scanning (Agata, Gretina, etc.), but slow and bulky setup
- What if we were not restricted to 90°? Would increase efficiency a lot!

New MPP Compton Scanner Setup

MPP Compton Scanner, First Data