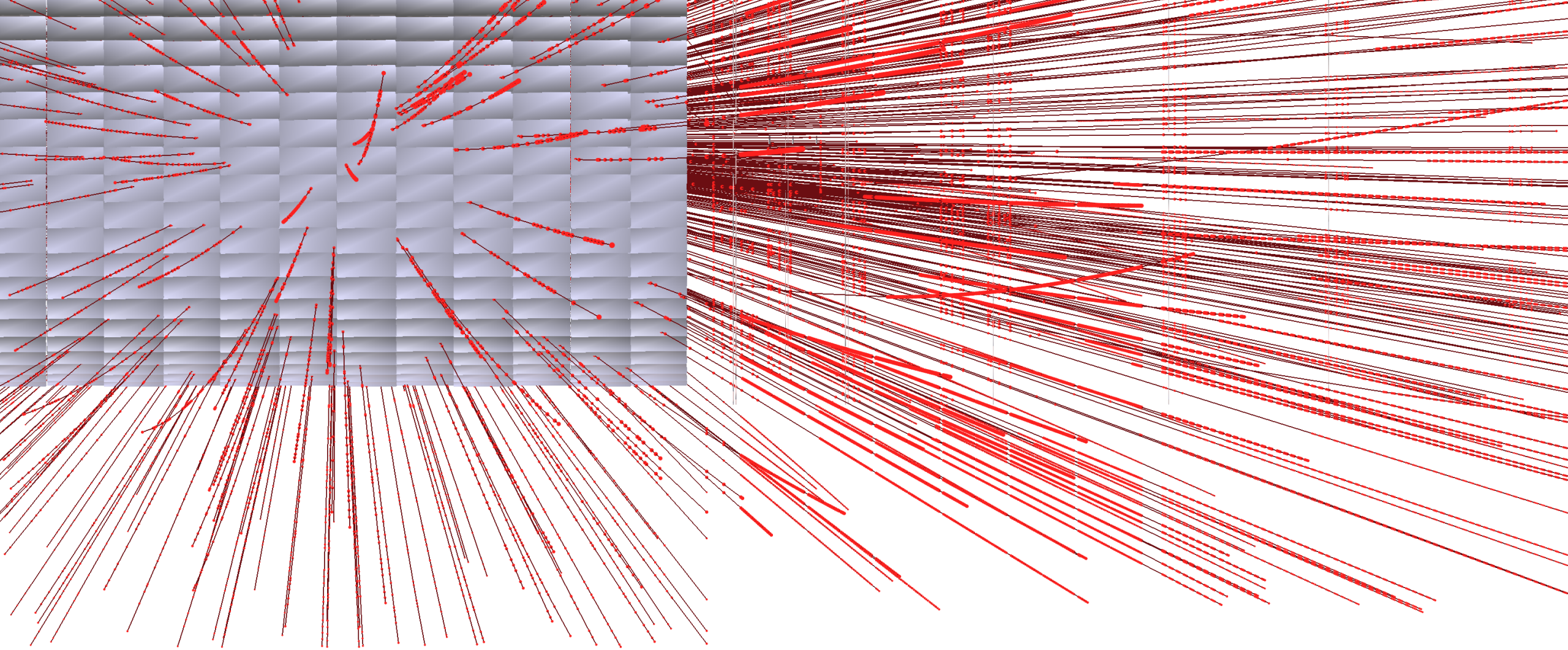

Surface memory management

Paul Gessinger

Acts developers meeting 07/11/2018

Surface memory management is homecooked

```
 template <typename... Tail>
  Measurement(const Surface&      surface,
              const identifier_t& id,
              CovMatrix_t         cov,
              typename std::enable_if<sizeof...(Tail) + 1 ==
sizeof...(params),
                                      ParValue_t>::type head,
              Tail... values)
    : m_oParameters(std::make_unique<const CovMatrix_t>(std::move(cov)),
                    head,
                    values...)
    , m_pSurface(surface.cloneIfFree())
    , m_oIdentifier(id)
  {
  }

 virtual ~Measurement()
  {
    if (m_pSurface && m_pSurface->isFree()) {
      delete m_pSurface;
      m_pSurface = nullptr;
    }
  }
```

Measurement.hpp

```
const Acts::PlaneSurface*
Acts::PlaneSurface::cloneIfFree() const
{
  if (isFree()) {
    return new PlaneSurface(*this);
  }
  return (this);
}
```

PlaneSurface.hpp

Surface memory management is homecooked

- Causes problems:
  - Swapping is complicated
  - Move / copy constructors are non-obvious
  - Need to be super careful with handling

```
Measurement(const Measurement<identifier_t, params...>& copy)
  : m_oParameters(copy.m_oParameters)
  , m_pSurface(copy.m_pSurface->cloneIfFree())
  , m_oIdentifier(copy.m_oIdentifier)
{}

Measurement(Measurement<identifier_t, params...>&& rhs)
  : m_oParameters(std::move(rhs.m_oParameters))
  , m_pSurface(rhs.m_pSurface)
  , m_oIdentifier(std::move(rhs.m_oIdentifier))
{
  rhs.m_pSurface = nullptr;
}

Measurement<identifier_t, params...>&
operator=(Measurement<identifier_t, params...>&& rhs)
{
  m_oParameters = std::move(rhs.m_oParameters);
  m_pSurface    = rhs.m_pSurface;
  m_oIdentifier = std::move(rhs.m_oIdentifier);
  return *this;
}
```

...

Proposed alternative: `std::shared_ptr`

- With `std::enable_shared_from_this`, this becomes possible:

```
const Surface* srf = get_from_wherever();
std::shared_ptr<const Surface> srf_ptr = srf->getSharedPtr();
```

- (with srf_ptr pointing at the right reference count)
- **Caveat: Surface needs to have been produced by `std::shared_ptr`'s constructors!**
- Don't (maybe even disallow) creation of Surface instances other that shared pointers:

```
std::shared_ptr<const Surface> srf = Surface::makeShared<PlaneSurface>(/*...*/);
```

- Then just continue as usual: binning, interlinking, all raw pointers
- Only difference: whenever you need shared ownership, you call `getSharedPtr()`
  - Performance impact should be negligible if done right (need to establish this somehow)

Is this viable?

- I think so, all the (move/copy) constructor, destructor logic goes away.
- Ownership is communicated clearly.
- Some spots in code need to handle vectors of shared_ptrs: unpack to raw pointers to avoid having to copy the vector (expensive)
- I have a WIP MR at !482