CORSIKA8 Technical report

Fan Hu

22/10/06

Non-absorbing Observation Plane

Air Shower Physics > corsika > Issues > #534

Open Issue created 1 week ago by **Fan Hu** Developer Close issue

Observation plane cause infinite loop if absorbing = false

If an `ObservationPlane` with `absorbing = false` is added to the processes, the cascade will be trapped in infinite loop.

To reproduce this bug: make the `ObservationPlane` in `example/corsika.cpp` non-absorbing.

Edited 1 week ago by Fan Hu

```
inline void Cascade<TTracking, TProcessList, TOutput, TStack>::step(
    particle_type& particle) {

  ... // find process with minimum step length

  // apply all continuous processes on particle + track
  if (sequence_.doContinuous(particle, step, limitingId) ==
      ProcessReturn::ParticleAbsorbed) {
    particle.erase();
    return; // particle is gone -> return
  }
  particle.setTime(particle.getTime() + step.getDuration());
  particle.setPosition(step.getPosition(1));
  particle.setDirection(step.getDirection(1));
  if (isContinuous) {
    return; // there is nothing further, step is finished
  }
}
```

```
ProcessReturn ObservationPlane::doContinuous(
    TParticle& particle, TTrajectory& step, bool const stepLimit) {
  if (!stepLimit) {
      ...
      return ProcessReturn::Ok;
  }
  this->write(...)
  if (deleteOnHit_) {
    return ProcessReturn::ParticleAbsorbed;
  } else {
    return ProcessReturn::Ok;
  }
}
```

```
LengthType ObservationPlane::getMaxStepLength(
    TParticle const& particle, TTrajectory const& trajectory) {
  auto const intersection = TTracking::intersect(particle, plane_);

  TimeType const timeOfIntersection = intersection.getEntry();

  if (timeOfIntersection < TimeType::zero()) {
    return std::numeric_limits<double>::infinity() * 1_m;
  }
  if (timeOfIntersection > trajectory.getDuration()) {
    return std::numeric_limits<double>::infinity() * 1_m;
  }
  double const fractionOfIntersection = timeOfIntersection /
      trajectory.getDuration();

  return trajectory.getLength(fractionOfIntersection);
}
```

Quick solution

```
LengthType ObservationPlane::getMaxStepLength(
    TParticle const& particle, TTrajectory const& trajectory) {
  auto const intersection = TTracking::intersect(particle, plane_);

  TimeType const timeOfIntersection = intersection.getEntry();

  if (timeOfIntersection <= TimeType::zero()) {
    return std::numeric_limits<double>::infinity() * 1_m;
  }
  if (timeOfIntersection > trajectory.getDuration()) {
    return std::numeric_limits<double>::infinity() * 1_m;
  }
  double const fractionOfIntersection = timeOfIntersection /
      trajectory.getDuration();

  return trajectory.getLength(fractionOfIntersection);
}
```

Possible problem:

Q: What if the particle is produced right at the observation plane?
A: Not likely to happen in double precision floating points. Unless the observation plane itself produce secondary particles.

Q: What about stacking planes (two planes located together)?
A: This implementation can't solve the stacking plane problem (Issue 397). They have to be placed with small displacements.

Observation Box

Air Shower Physics > corsika > Issues > #471

Open Issue created 10 months ago by **Fan Hu** Developer Close issue

Add ObservationCubic to support volumetric observatories

It would be good to have a `ObservationCubic` that works similar to the existing `ObservationPlane`.

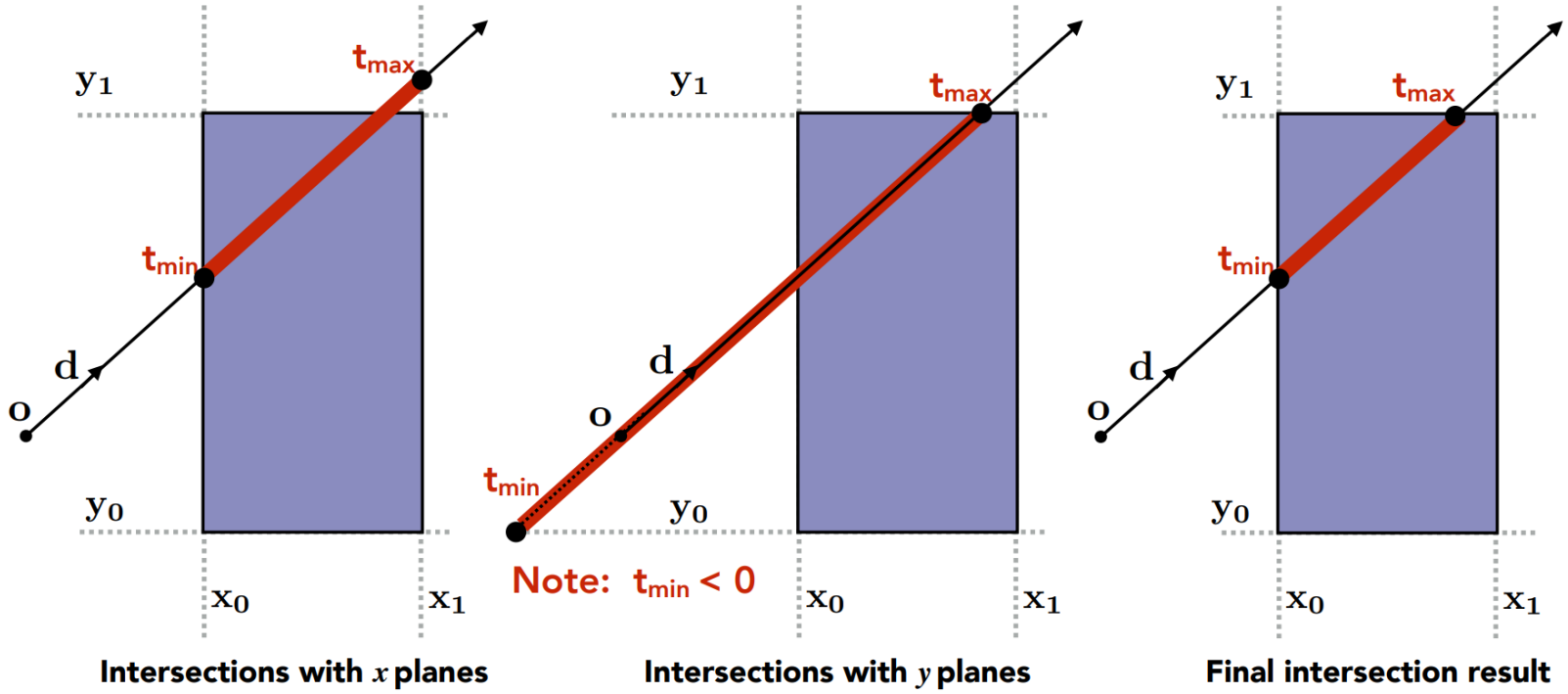
The `ObservationCubic` can store particles arriving at the `Cubic` surface in a output file. And then, user can send those particles to further detector simulations.

Similar to Observation Plane, but use a box for intersection test.

Observation Box

Ray Intersection with Axis-Aligned Box

2D example; 3D is the same! Compute intersections with slabs and take intersection of t_{min}/t_{max} intervals



How do we know when the ray intersects the box?

Observation Box

Ray Intersection with Axis-Aligned Box

- Recall: a box (3D) = three pairs of infinitely large slabs
- Key ideas
  - The ray enters the box **only when** it enters all pairs of slabs
  - The ray exits the box **as long as** it exits any pair of slabs
- For each pair, calculate the t_{min} and t_{max} (negative is fine)
- For the 3D box, t_{enter} = **max**$\{t_{min}\}$, t_{exit} = **min**$\{t_{max}\}$
- If $t_{enter} < t_{exit}$, we know the ray **stays a while** in the box (so they must intersect!) (not done yet, see the next slide)

Observation Box

Ray Intersection with Axis-Aligned Box

- However, ray is not a line
  - Should check whether t is negative for physical correctness!
- What if $t_{exit} < 0$?
  - The box is “behind” the ray — no intersection!
- What if $t_{exit} >= 0$ and $t_{enter} < 0$?
  - The ray’s origin is inside the box — have intersection!
- In summary, ray and AABB intersect iff
  - $t_{enter} < t_{exit}$ && $t_{exit} >= 0$

One step forward: if the user put Box/Cylinder/... in VolumeTreeNode

```
template <typename TParticle, typename TBaseNodeType>
inline Intersections Tracking::intersect(TParticle const& particle,
                                         TBaseNodeType const& volumeNode) {
  if (Sphere const* sphere = dynamic_cast<Sphere const*>(&volumeNode.getVolume());
      sphere) {
    return Tracking::intersect<TParticle>(particle, *sphere);
  } else if (Box const* box = dynamic_cast<Box const*>(&volumeNode.getVolume()); box) {
    return Tracking::intersect<TParticle>(particle, *box);
  } else if (Cylinder const* cylinder = dynamic_cast<Cylinder const*>(&volumeNode.getVolume());
cylinder) {
    return Tracking::intersect<TParticle>(particle, *cylinder);
  } else {
    throw std::runtime_error(
        "The Volume type provided is not supported in "
        "Intersect(particle, node)");
  }
}
```

Cascade type conversion!

Cascade type conversion

Three tracking methods:

```
tracking_leapfrog_curved::Tracking
tracking_leapfrog_straight::Tracking
tracking_line::Tracking
```

Multiple geometry types:

```
IVolume
- Sphere
- Box
- Cylinder
```