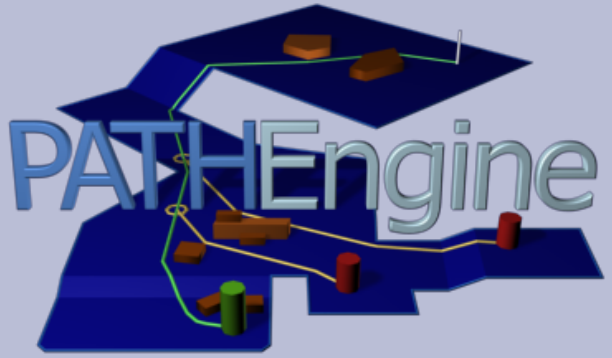
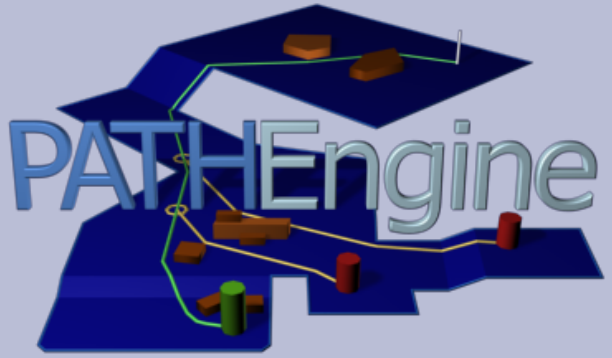


Detail Issues in Robust Pathfinding

Thomas Young

www.pathengine.com

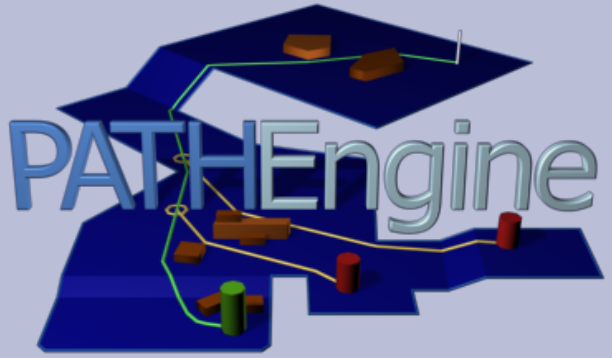


Starting point

- Robust pathfinding is difficult
- Paul Tozour on Game/AI

  'Fixing Pathfinding Once and For All'

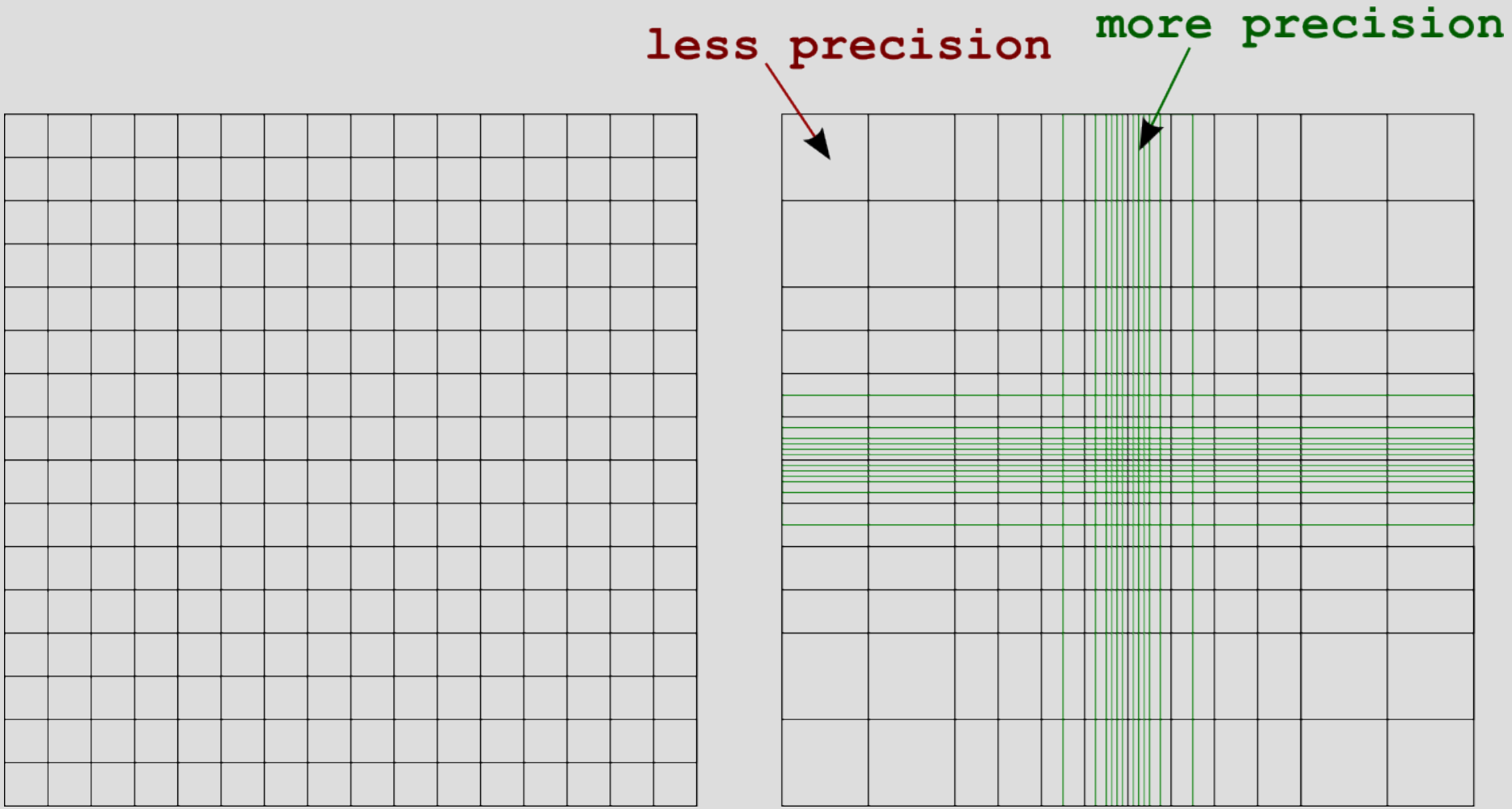
  - movement space is important
    - e.g. nav-meshes instead of waypoints
- Robust geometry is difficult
  - geometry error can cause similar issues
- I'm going to talk about some details of how we adress this in PathEngine...



Issues with floats

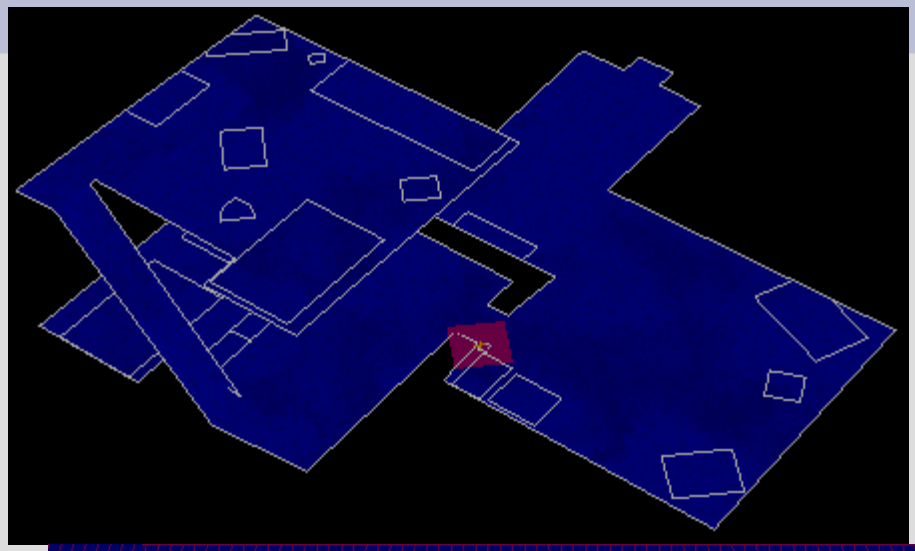
- Avoid floating point operations because:
  - results vary with execution context
    - build/platform/CPU state
  - they don't actually give us extra precision, in the general case
  - geometric error is hard to predict and control
- A precise treatment of contained space is very difficult (or impossible) with floating point!

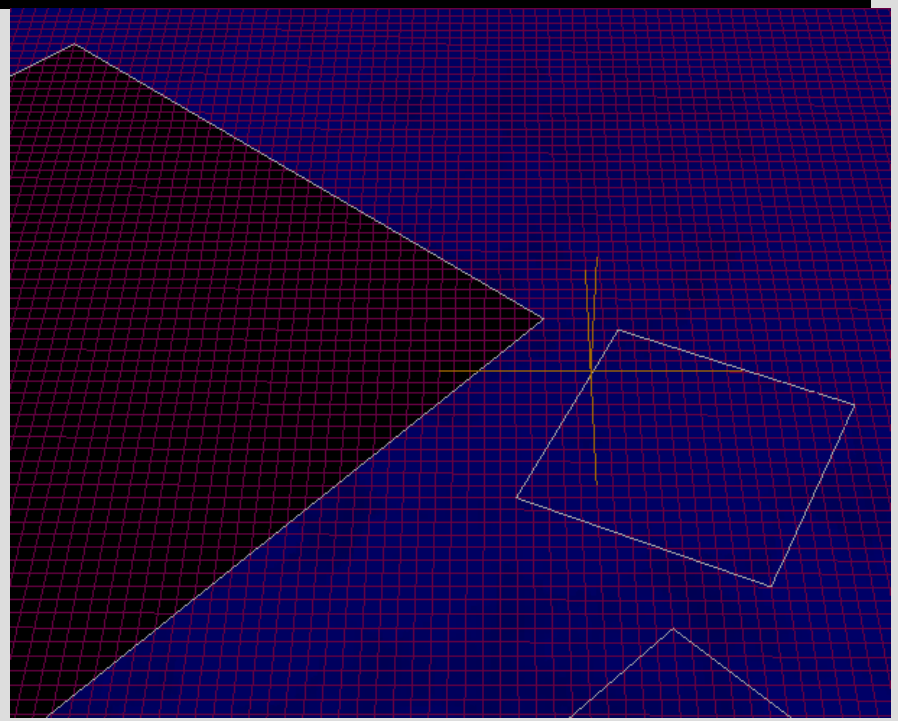
Floating point 'grid'

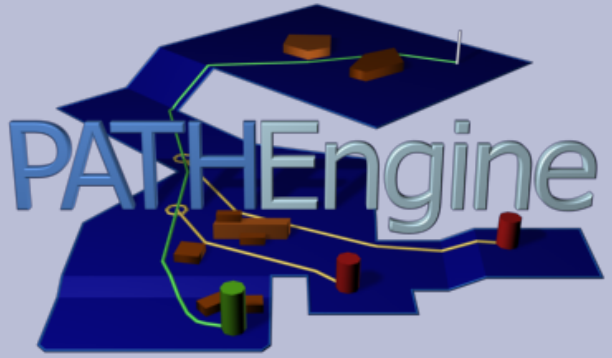


Integer space

- Integer horizontal coordinates for:
  - mesh vertices
  - obstruction shape corners
- No fixed scale factor
  - user chooses a 'pathfinding unit'
  - e.g. 1cm







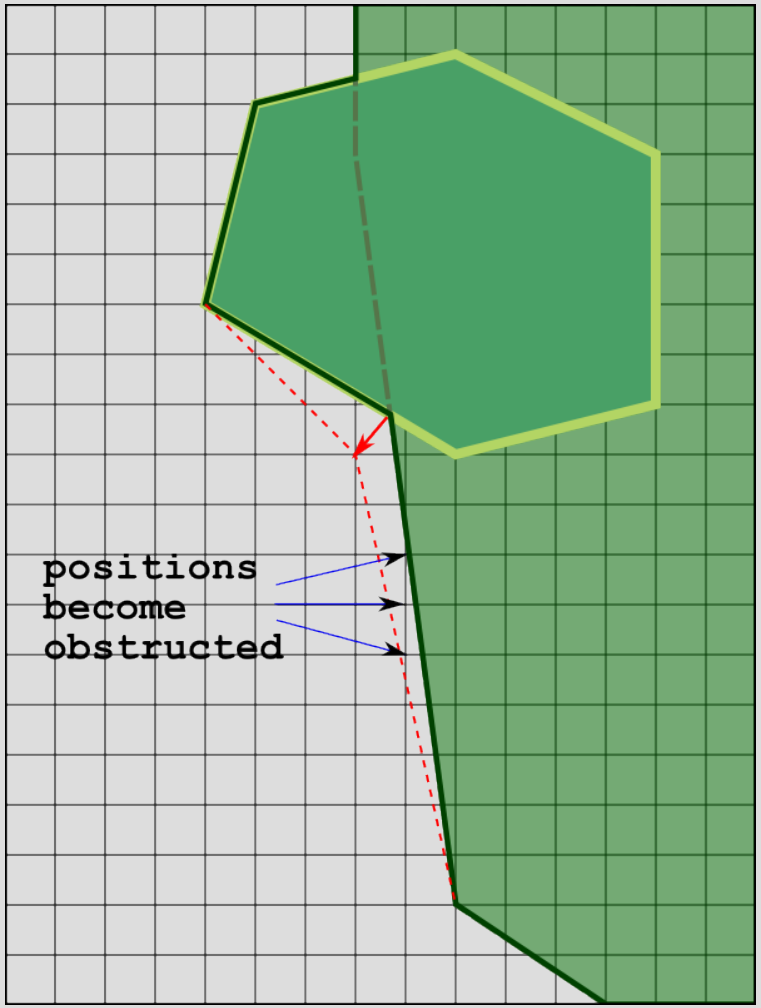
'Multiplying up'

- Avoid division and irrationals
  - just addition, subtraction, and multiplication
  - never need to work with anything smaller then one 'pathfinding unit'
  - no need for fixed point fractions
- Some operations are naturally implemented this way
  - e.g. side of line test
  - others need to be reimplimented

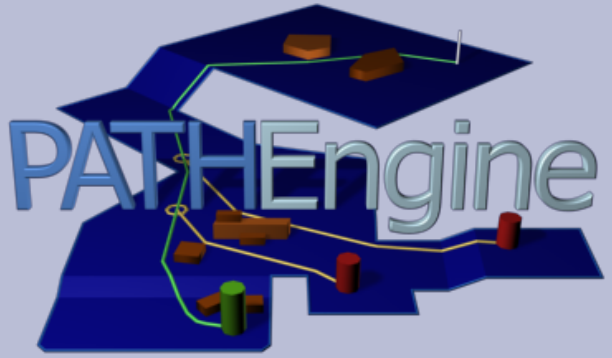
Range constraint

- That leaves overflow!
  - Define a starting range
  - Carefully track variable ranges
  - Multiply up to larger integers as necessary
- PathEngine world range is $^{+/-}$1500000
  - = 30km by 30km, with 1cm units
  - when streaming coordinate origin is local to each streaming unit

Intersections

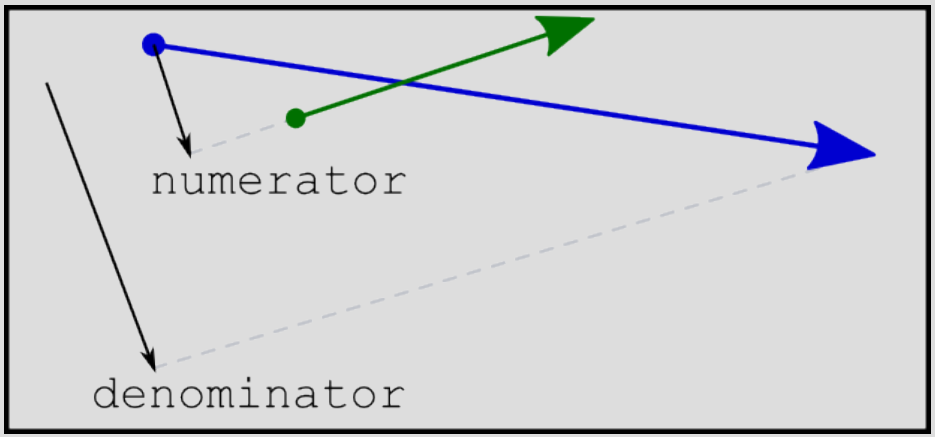


- Boundary intersections a key source of error
  - e.g. around dynamic obstacles
- Approximating can change the boundary
- Representing intersections *exactly* can avoid this



Vector fractions

- Replace division with multiplication
  - 'vector fractions'

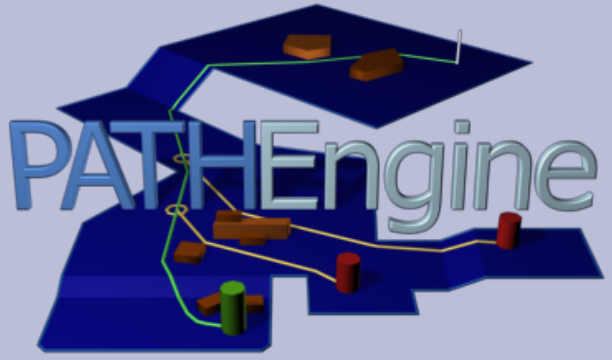


(see my article in Gems 3)

  - multiply out denominators to compare

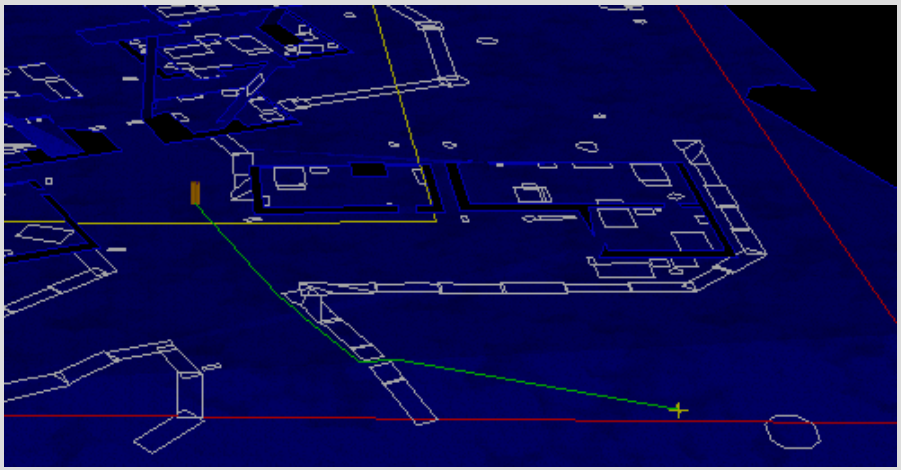
Choice of approximation

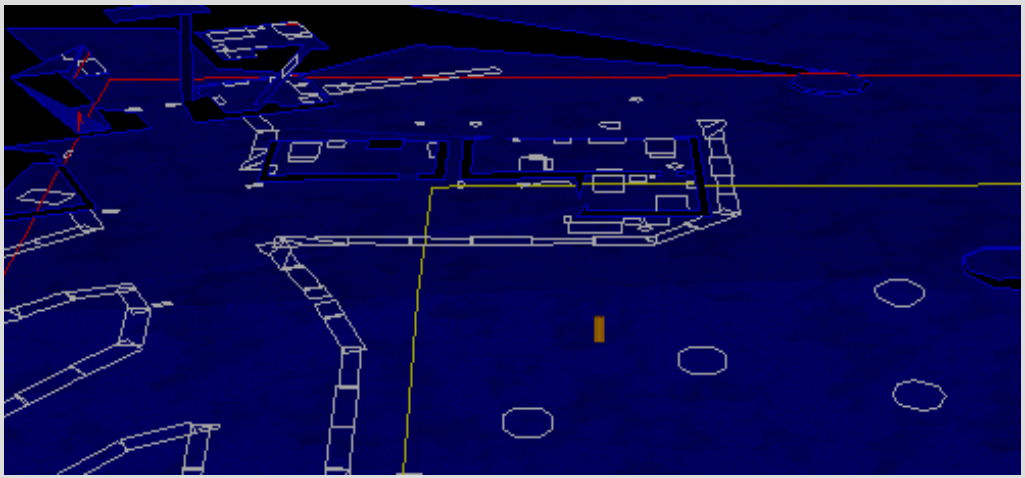
- With an exact treatment in place we can choose when to approximate
  - important for performance
- Robust approximation is difficult
  - often more difficult than an exact treatment
  - having the exact treatment in place can help with approximating robustly

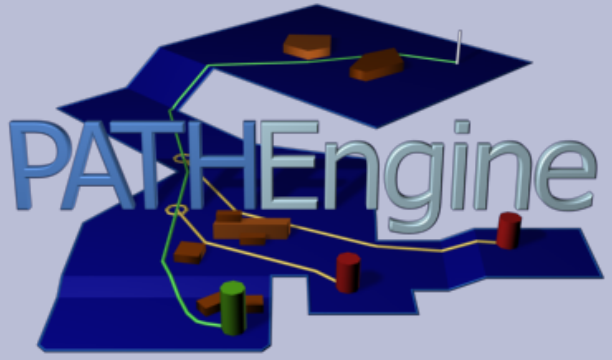


Boundary invariants

- Boundary invariants are very powerful
  - simplify higher level code
  - open up additional possibilities
    - e.g. seamless pathfinding across streaming boundaries
      - geometry replication across overlaps

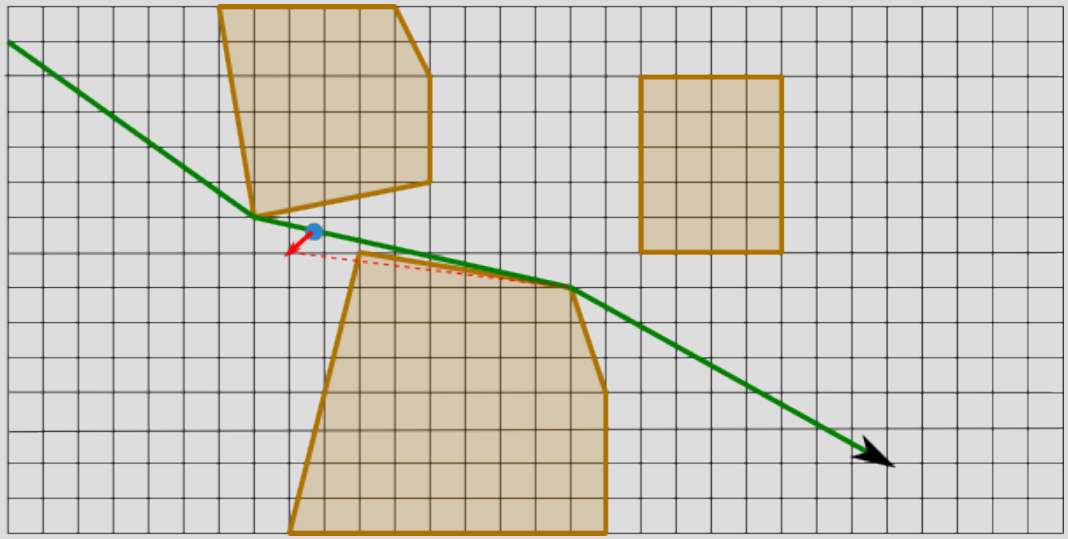


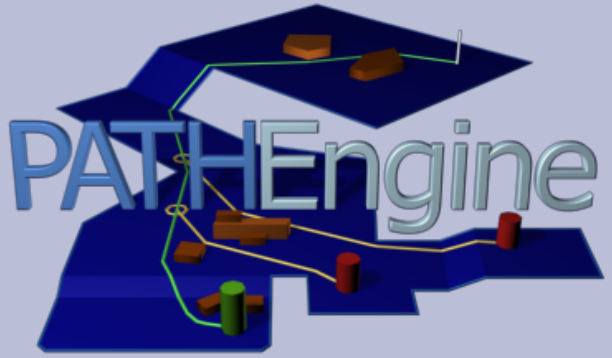




Point on path segment

- Be careful about interpolation along path segments
  - approximation can't be avoided here
  - can push an agent inside obstructed space
  - or 'snag' path segments against obstructions





Wrapping up

- Eliminating geometry failure cases can be difficult
  - but worth thinking about up front
  - easier if included by design, as opposed to retrofitted
  - working around failure cases can be much harder
- If errors are possible then take this into account in your movement architecture!
  - e.g. collision override