Bullet Time Effect with Local Perception Filters

(danny perry)

Contents

- Introduction to Perception Filters (most of the paper)
- Use of Perception Filters with bullet time effect

Bullet Time Effect

- Introduced in The Matrix

Bullet Time Effect

- Introduced in The Matrix
- Now used in computer games:
  - Max Payne, Jedi Knight II, Matrix games, etc.
- Any others to mention?

Bullet Time Effect

- Trivial in single player - “just slow down rendering”
- Not many multiplayer games do it: why?

Bullet Time Effect

- Trivial in single player - “just slow down rendering”
- Not many multiplayer games do it: why?
  - If slow down whole game, bad for other players
  - Instead, some speed up the player
- This paper presents an alternate method: change perception of the different players

Perception Filters

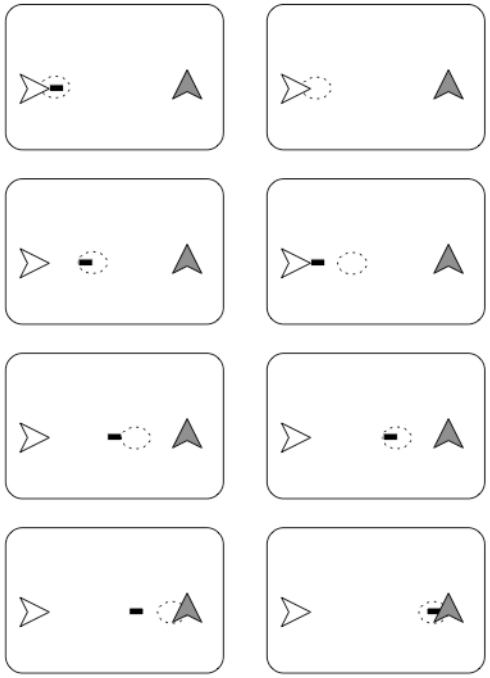
- “a method used to hide communication delays in networked virtual environments” (introduced by Sharkey, et al.)

- Essentially: speed up and slow down speed of passive entities when they are near local and remote players

Perception Filters

- Verbage:
  - Local players = “local players” = p
  - Remote players = “remote players” = r
  - Passive entities = “entities” = e
- Perception Filters approach:
  - Local players rendered using up-to-date information
  - Remote players rendered using d seconds old information (with d second delay)
  - (This is different from dead reckoning, we are not predicting)
  - Entities are rendered at the time of the player they are closest to.

Perception Filters



Perception Filters

- (java demo)

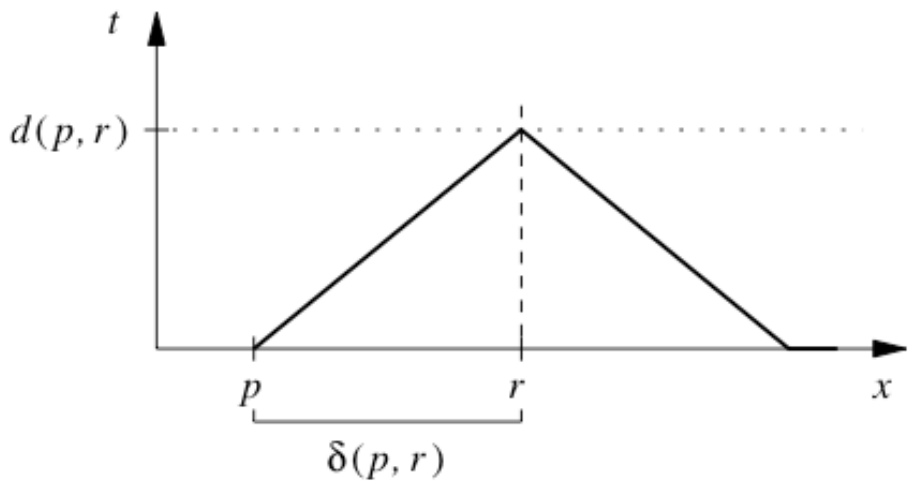
Perception Filters

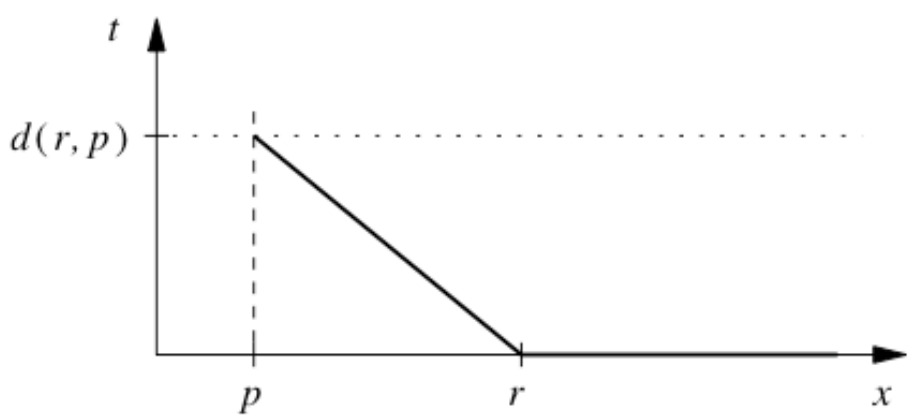
- How its done:
  - d(i,j) is delay, delta(i,j) is distance

$$d(p,e) = \begin{cases} 0, & \text{if } \delta(p,e) = 0, \\ d(p,r), & \text{if } \delta(r,e) = 0. \end{cases}$$

$$d(p,e) = d(p,r) \cdot \max\left\{1 - \frac{\delta(r,e)}{\delta(p,r)},\ 0\right\}$$

- From p: slower uphill, faster down (top graph)
- From r: faster at top (bottom graph)





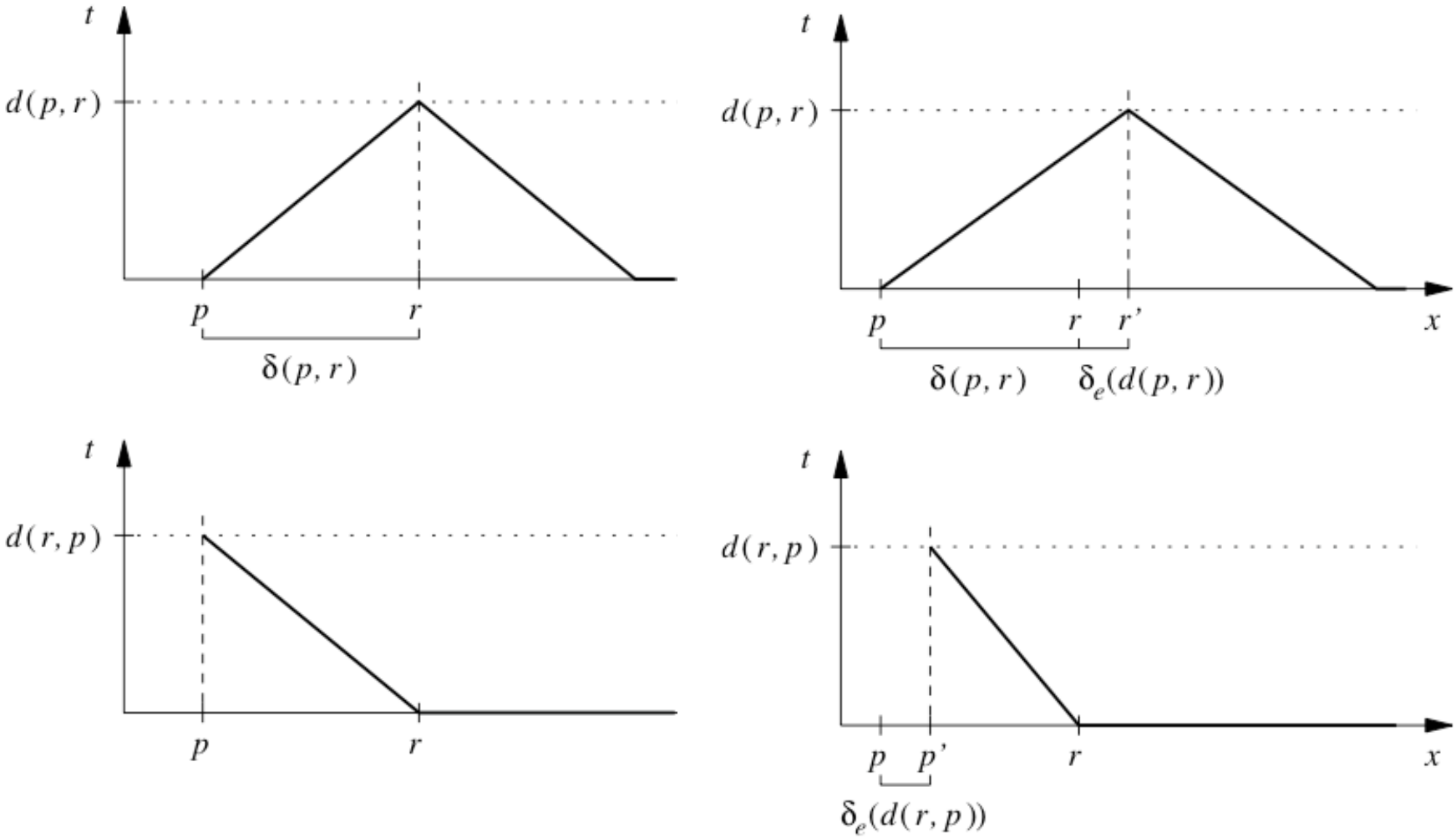
Perception Filters

- Problem: when e reaches r, p’s view of e is not there yet, but e will speed up anyway in p’s view.
- Solution ?

Perception Filters

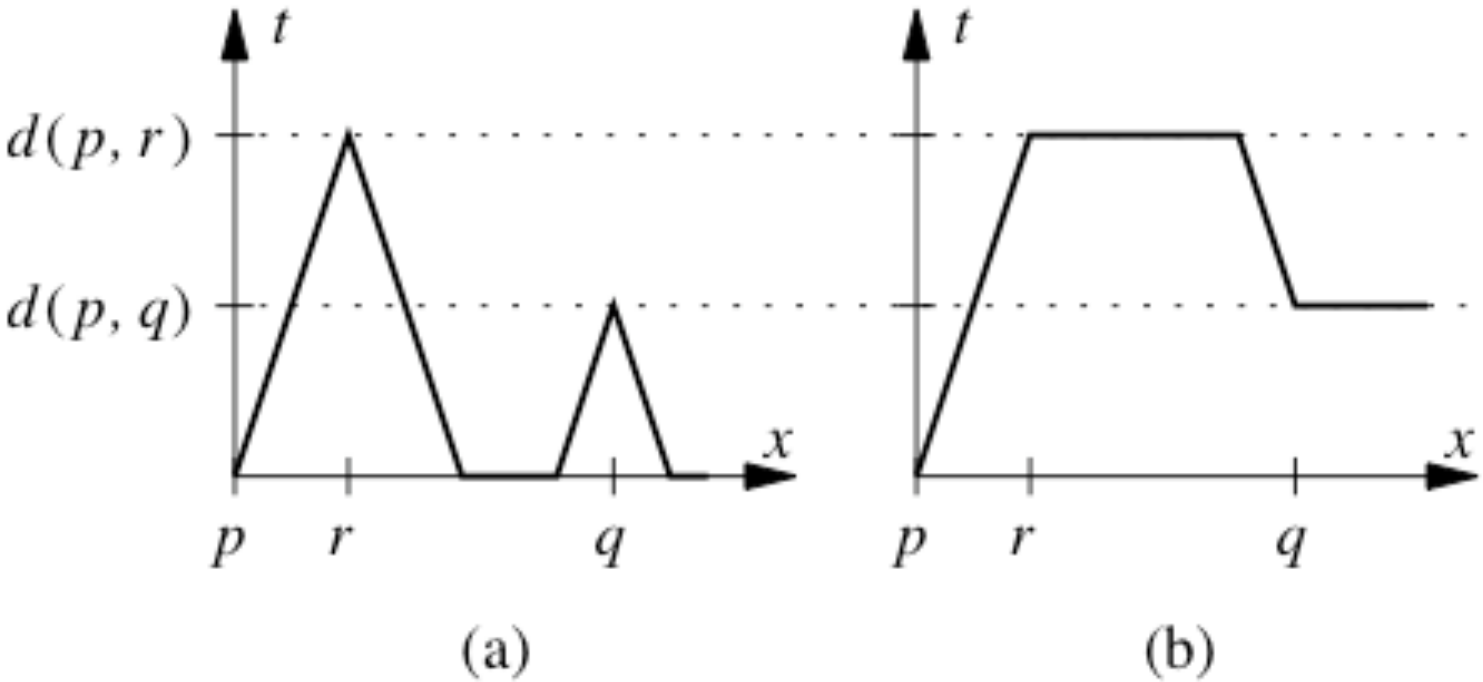
- Problem: when e reaches r, p's view of e is not there yet, but e will speed up anyway in p's view.
- Solution:
  - Introduce a shadow r' that "buffers" p's view of the speedup

$$\delta(r, r') = \delta_e(d(p, r)). \quad d(p, e) = \begin{cases} 0, & \text{if } \delta(p, e) = 0, \\ d(p, r), & \text{if } \delta(r', e) = 0. \end{cases}$$

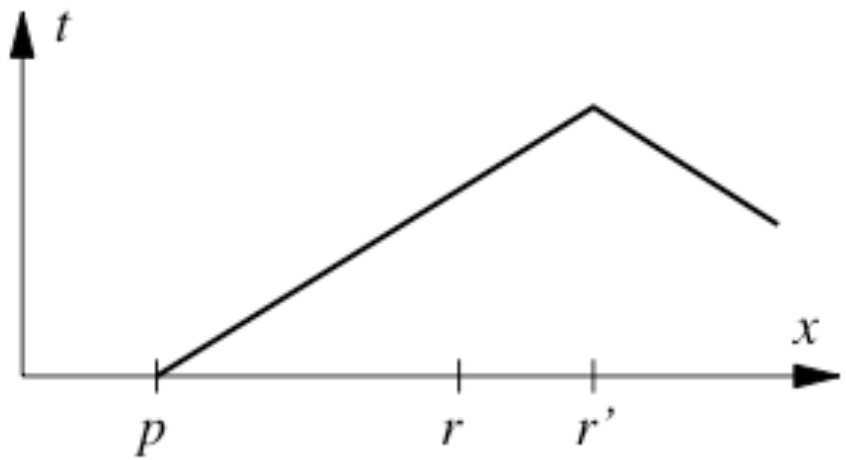


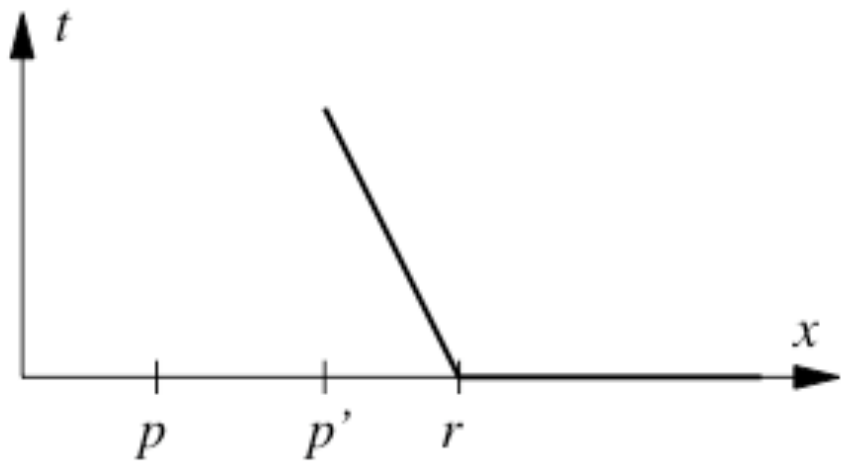
- Top: won’t speed up until r’
- Bottom: starts showing e at p’

- With multiple players 2 approaches:
  - A. Minimize e's not at local time (closest to local time)
  - B. Minimize delay changes (more smooth)

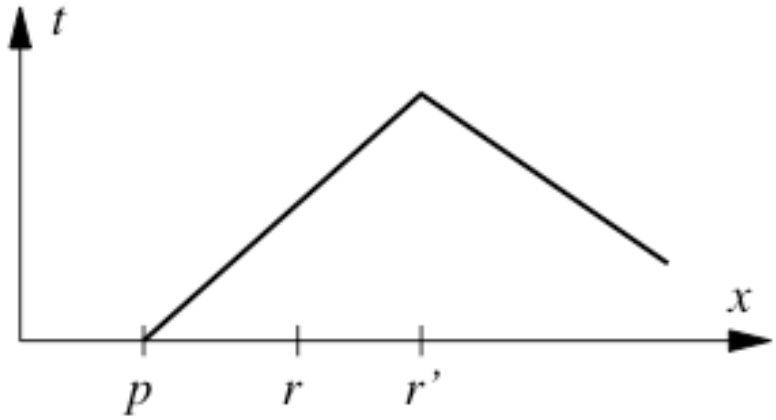


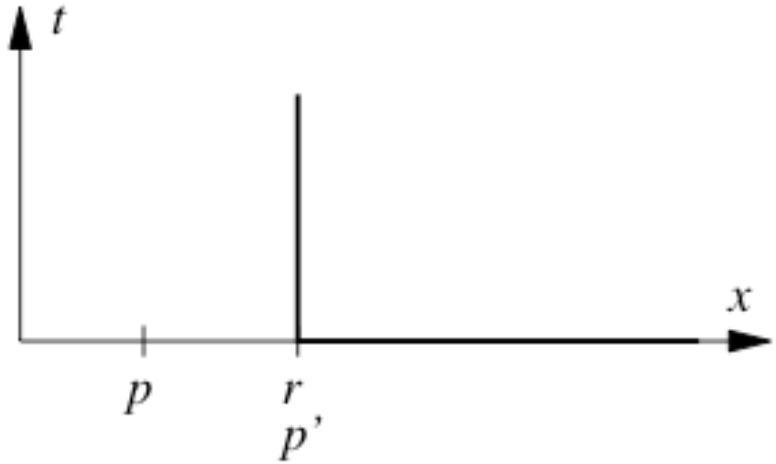
- Problems
  - When they reach “critical proximity” - get too close - no longer works. (graph: still okay)



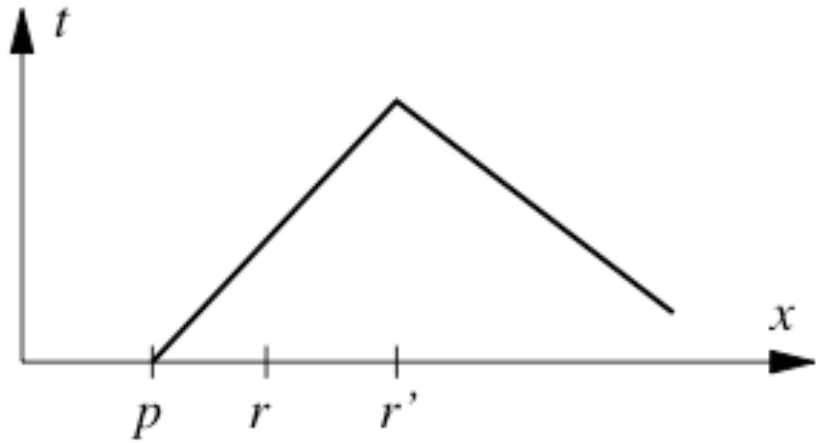


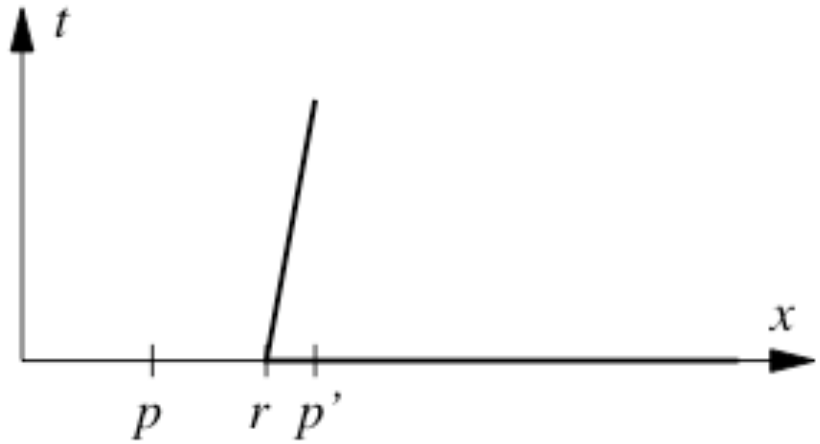
- Problems
  - When they reach “critical proximity” - get too close - no longer works(graph: at critical proximity)





- Problems
  - When they reach “critical proximity” - get too close - no longer works (graph: past)





Perception Filters

- Other Limitations ?

Perception Filters

- Other Limitations ?
  - A local player CANNOT DIRECTLY INTERACT with a remote player
  - Computational requirements
  - Jitter can mess things up easily (won't look smooth)
  - Assumption that we know exact delays

Bullet Time

- Use the same machinery: when a player is using bullet time it is similar to a large delay.
- In Bullet Time mode, entities close to player slow down.