Realizing Bullet Time Effect in Multiplayer Games with Local Perception Filters

Jouni Smed*
Turku Centre for Computer Science (TUCS) and Department of Information Technology, University of Turku, Finland
jouni.smed@cs.utu.fi

Henrik Niinisalo
Department of Information Technology, University of Turku, Finland

Harri Hakonen
Department of Information Technology, University of Turku, Finland
harri.hakonen@cs.utu.fi

ABSTRACT

Local perception filters exploit the limitations of human perception to reduce the effects of network latency in multiplayer computer games. Because they allow temporal distortions in the rendered view, they can be modified to realize bullet time effect, where a player can get more reaction time by slowing down the surrounding game world. In this paper, we examine the concepts behind local perception filters and extend them to cover artificially increased delays. The presented methods are implemented in a testbench program, which is used to study the usability and limitations of the approach.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Synchronous Interaction*; K.8.0 [**Personal Computing**]: General—*Games*

General Terms

Algorithms

Keywords

Computer games, networking, multiplayer, latency, bullet time, virtual environments

1. INTRODUCTION

"Bullet time" is a visual effect which combines slow motion with dynamic camera movement. Bullet time effect was introduced in the film *The Matrix* [9], and it soon found its way into computer games like *Max Payne* [3]. In computer games, bullet time allows the player to slow down the surrounding game world thus enabling to the player to have more time to make decisions.

Whereas the bullet time effect is quite easy to implement in a single player game, simply by slowing down the rendering, in multiplayer games the bullet time effect is not used—or it is implemented as speeding up the player rather than slowing down the environment. For instance, force speed in *Jedi Knight II: Jedi Outcast* [2] implements the bullet time effect differently in the single player mode than in the multiplayer mode. The reason for this is obvious: If one player could slow down the time of its surroundings, it would be awkward for the other players within the influence area because, rather than enhancing the game play of the player using the bullet time, it would only hinder the game play of the other human players.

In this paper, we propose a method realizing the bullet time effect in a multiplayer game. It is based on the idea of *local perception filters* introduced in [5], which is a method used to hide communication delays in networked virtual environments. It exploits the human perceptual limitations by rendering entities at slightly out-of-date locations based on the underlying communication delays. The idea is to make these temporal distortions of the game world as unnoticeable as possible. However, it can be accommodated to include also artificial temporal distortions caused by the bullet time effect, as we shall see in this paper.

We begin the paper with an introduction to local perception filters in Section 2. For a broader review of the problems in networked computer games, see [7, 8]. We discuss the problems that have to be solved in the implementation and general limitations of the approach. Next, we introduce how the local perception filter approach can be extended to include bullet time effect in Section 3. Section 4 describes the MMD system, which is used as a testbench to try out and improve the described ideas. Final remarks appear in Section 5.

2. LOCAL PERCEPTION FILTERS

The entities of a game world can be separated into two classes:

- *Active entities* are indeterministic entities (e.g., con-

*Corresponding author. Address: Lemminkäisenkatu 14 A, FI-20520, Turku, Finland. Phone: +358-2-3338673.

SIGCOMM'04 Workshops, Aug. 30+Sept. 3, 2004, Portland, Oregon, USA.

trolled by human players) whose behaviour cannot be predicted.

- *Passive entities* are deterministic entities, whose behaviour follow, for example, the laws of physics (e.g., projectiles) or which are otherwise predictable (e.g., buildings).

Interaction means, theoretically speaking, that the interacting entities must communicate with each other to resolve the outcome. If the communication delay between entities is negligible (e.g., they reside in the same computer), the interaction seems credible. On the other hand, networking incurs communication delays which can hinder the interaction between the active entities.

Because in computer games the active entities usually correspond to the avatars of the players, we shall henceforth refer to active entities as *players* and passive entities simply as *entities*. Based on the communication delay, we divide the players to *local players* (e.g., sharing the same computer) and *remote players* (e.g., players connected by a network).

Local perception filters address this problem of delays by discerning the actual situation from the rendered situation. The rendered situation, which is perceived by the player, need not to coincide with the current actual situation but it can comprise some out-of-date information. The amount of this temporal distortion is easy to determine for active entities: Local players are rendered using up-to-date state information, whilst a remote player with a communication delay of d seconds is rendered using the known, d seconds old state information. In this sense, local perception filters differ from dead reckoning, where the communication delay is compensated by predicting the current state of a remote player from the out-of-date information.

The temporal distortion of passive entities can change dynamically. The nearer an entity is to a local player, the closer it has be to rendered to its current state, because it is possible that the player is going to interact with it. Conversely, an entity nearing on a remote player must be rendered closer to that remote player's time, because if there is an interaction between the remote player and the entity, the outcome is rendered after the communication delay. In other words, the rendered remote interactions, albeit occurring in real-time, have happened in the past, and only when the local player itself participates in the interaction, it must happen in the present time.

Figure 1 gives an example, where the player controlling the white ship shoots a bullet (i.e., a passive entity) towards the grey ship controlled by a remote player. The players' views are not entirely consistent with each other: In the beginning the white ship renders the bullet to the actual position but as it closes on the grey ship it begins to lack behind the actual position. Conversely, when the grey ship first learns about the bullet, it has already travelled some distance. For example, let us assume that the communication delay between the ships is 0.5 seconds and the bullet travels in 2.0 seconds from the white ship to the grey ship. When the white ship fires, it sees the bullet immediately but after that the rendered bullet starts to drag behind the actual position. After 2.0 seconds the bullet has arrived to the grey ship, but it is rendered like it has travelled only 1.5 seconds. It takes 0.5 seconds for the grey ship's reaction to convey to the white ship, and once that message arrives, after 2.5 seconds, the bullet is rendered near the grey ship and

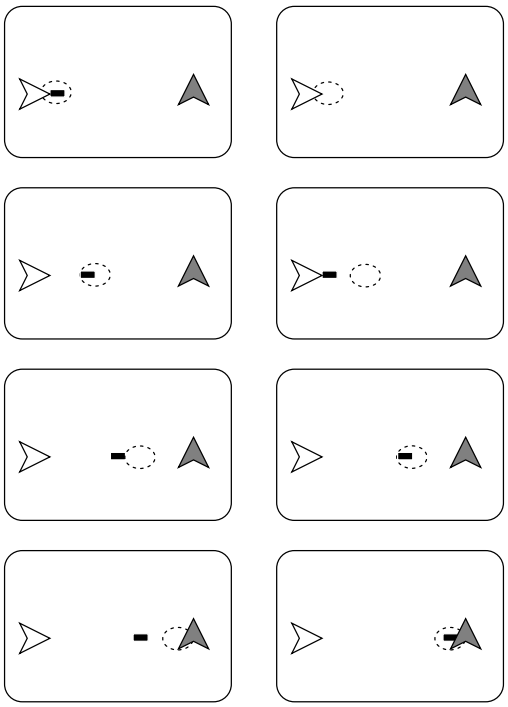

Figure 1: An example of local perception filters with two stationary players (white and grey ship) and one moving entity (a bullet shot by the white ship). On the left side, from top to bottom, are the rendered views from the white ship's perspective; on the right side are the corresponding views from the grey ship's perspective. Dashed ovals indicate the actual position of the bullet and black rectangles its rendered position. As the bullet closes on the grey ship, the white ship perceives it to slow down, whilst the grey ship perceives it to gain speed.

reaction occurs at an appropriate moment. From the grey ship's perspective the chain of events is different: When it learns about the bullet, it has already travelled 0.5 seconds, but it is rendered coming from the white ship. The rendered bullet must now catch up the actual bullet so that at the moment of 2.0 seconds both the rendered and actual bullet arrive to the grey ship, which can then react and send the reaction to the white ship.

Each player has its own perception of the game world, where all entities, in addition to spatial co-ordinates (x, y, z), are associated with a time delay (t), thus forming a $3\frac{1}{2}$-dimensional co-ordinate system. The local player is at the current time $t = 0$, and remote players are assigned t values according to their communication delays. Once we have assigned these values, we can define a causal surface [4] or a *temporal contour* [6] over the game world. The temporal contour defines suitable t values for each spatial point. Figure 2 illustrates one possible temporal contour for the white ship of the previous example. When the bullet leaves the white ship, $t = 0$, but the t value increases as the it closes on the grey ship, until they both have the same t value.

The changes in the movement of an entity caused by the temporal contour should be minimal and smooth. Moreover, all interactions between players and entities should

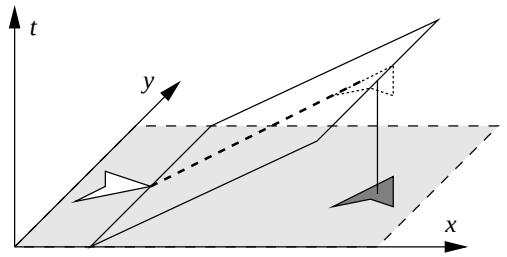

Figure 2: The $2\frac{1}{2}$-dimensional temporal contour from the white ship's perspective. The bullet travels "uphill" the contour until it reaches the t value of the grey ship.

appear to be realistic and consistent (e.g., preserve causality of events). The requirements for temporal contours, as outlined in [5], can be summarized into three rules:

1. Player should be able to interact in real-time with the nearby entities.
2. Player should be able to view remote interactions in real-time, although they can be out-of-date.
3. Temporal distortions in the player's perception should be as unnoticeable as possible.

The most important limitation of local perception filters, which follows from the first rule, is that a player cannot interact *directly* with a remote player. The players can engage into an exchange of passive entities (e.g., bullets, arrows, missiles or insults) but they cannot get into a mêlée with each other.

Another problem is the computational requirements, because all players have their own temporal contours, which must be updated dynamically whenever a remote player's position or communication delay changes. Ideally, the temporal contour should be smooth so that the changes remain unnoticeable. Although this seems to imply that the temporal contours must minimize the overall change, as the ones defined in [5], we have opted for linear functions because they are much more simple, easier to compute, and they are still reasonably unnoticeable.

In the following subsections we study first how to define linear temporal contours in the case of two players, and then extend the discussion to cover multiple players. After that, we point out some problems and limitations of local perception filters that should be considered in the implementation.

2.1 Two Players

Let us first look at a case where we have only two players, p and r, and one entity e. The players and the entity have a spatial location, and the players are associated with a communication delay, which is due to the network latency and cannot be reduced. If i and j are players or entities, let $\delta(i,j)$ denote the spatial distance between them and $d(i,j)$ the delay from the perspective of i. The communication delay between players does not have to be the same in both directions but we can allow $d(i,j) \neq d(j,i)$.

In the case of two players, the delay function d for the entity e must have the following properties

$$d(p,e) = \begin{cases} 0, & \text{if } \delta(p,e) = 0, \\ d(p,r), & \text{if } \delta(r,e) = 0. \end{cases} \quad (1)$$

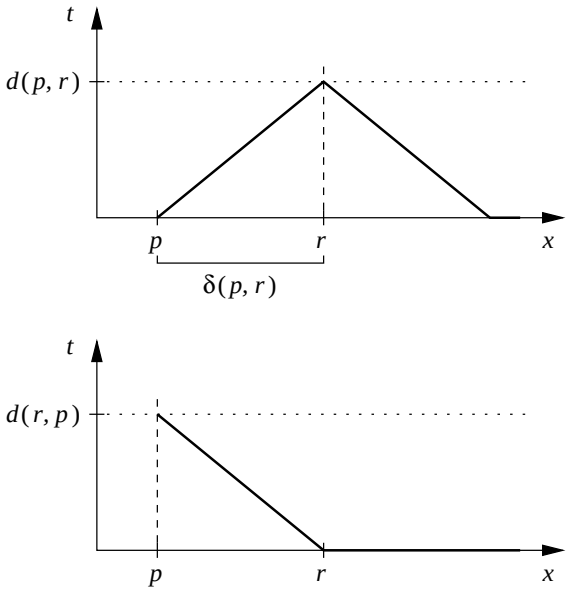

Figure 4: Player p shoots player r in a one-dimensional world. Above the temporal contour from the perspective of player p, and below from the perspective of player r. The corresponding values in t-axis illustrate the delay (i.e., the temporal difference) between the actual and rendered position at each actual spatial point in x-axis.

Simply put, if e and p are at the same position, the delay to p is zero, and if e and r are at the same position, the delay from p is the same as the communication delay from p to r.

The rest of the function can be defined, for example, linearly as

$$d(p,e) = d(p,r) \cdot \max\left\{1 - \frac{\delta(r,e)}{\delta(p,r)},\ 0\right\}, \quad (2)$$

which is illustrated in Figure 3. The delay function defines now a symmetrical temporal contour around r, which the entities must follow when they are rendered. This is not the only possibility, and the delay function can even be asymmetric (i.e., the slope does not have to be the same to all directions).

Let us take an example, which is illustrated in Figure 4, where player p shoots a bullet e towards player r. If we look at the situation from perspective of player p, initially the distance to the bullet $\delta(p,e) = 0$ and the delay $d(p,e) = 0$. The delay increases as the bullet closes on r, until $d(p,e) = d(p,r)$ when $\delta(r,e) = 0$. Once the bullet has passed r, the delay reduces back to zero. Player p perceives the temporal contour so that the bullet moves slower when it is climbing "uphill" and faster when it is going "downhill". From the perspective of player r, the bullet has initially delay $d(r,e) = d(r,p)$, which reduces to $d(r,e) = 0$ when $\delta(r,e) = 0$. In other words, player r perceives the bullet moving faster than its actual speed until it has passed the player.

If we define the temporal contour observing the constraints of Equation 1, we may notice a slight visual flaw in the rendered outcome. Assume player p shoots a bullet e towards remote player r. The bullet slows down, and when $\delta(r,e) = 0$, the delay function has reached its maximum and

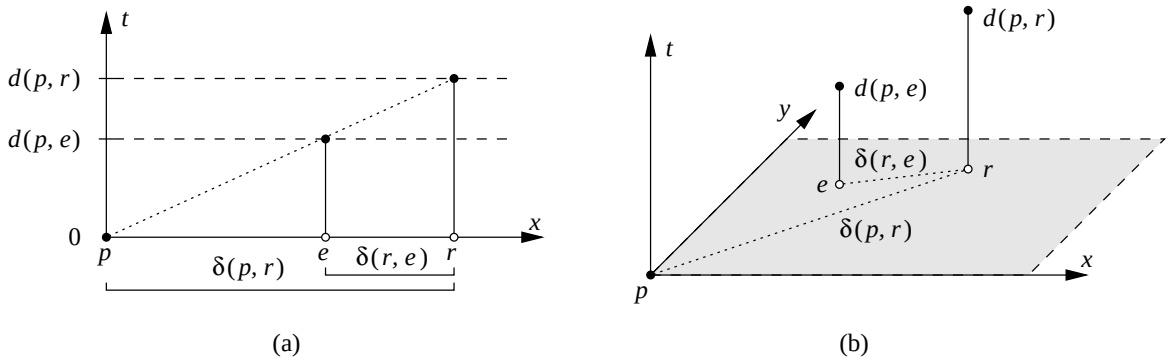

Figure 3: Examples of the linear delay function of Equation 2 defining the temporal contour in (a) one-dimensional game world and (b) two-dimensional game world.

$d(p,e) = d(p,r)$. However, when the actual bullet reaches r, the rendered bullet of p is still short of reaching r (see the bottom left frame of Figure 1). Because the temporal contour is already at its peak value, the bullet begins to speed up before it is rendered at r. This can look disruptive, because the change happens before the bullet is rendered to interact with the remote player. Intuitively, acceleration should occur only after the bullet has passed the remote player. From the perspective of player r, the rendering has a similar problem: Once r learns about the bullet, its rendered position is not next to p but some way forward along the trajectory. Simply put, the problem is that the delay function is defined using actual positions, whereas it should observe also the movement of the entity during the communication delay. This means that each individual entity requires a slight refinement of the temporal contour to reduce these perceptual disruptions.

To solve the problem, let us first introduce function $\delta_e(t)$, which represents the distance that the entity e travels in the time t. Obviously, the function is based on the velocity and acceleration information, but the given generalization suffices for our current use. Let us now define a *shadow* r' of player r that has the following property

$$\delta(r, r') = \delta_e(d(p, r)). \tag{3}$$

The shadow r' represents the position where the entity e actually resides, when player p is rendering it at the position of remote player r. Now, we can rewrite Equation 1 as

$$d(p,e) = \begin{cases} 0, & \text{if } \delta(p,e) = 0, \\ d(p,r), & \text{if } \delta(r',e) = 0. \end{cases} \tag{4}$$

Simply put, this means that we push the peak of temporal contour forward the distance of $\delta_e(d(p,r))$ to r', which is illustrated in Figure 5. The reason why we want to use the actual spatial positions is that they, unlike the rendered positions, are consistent among all players.

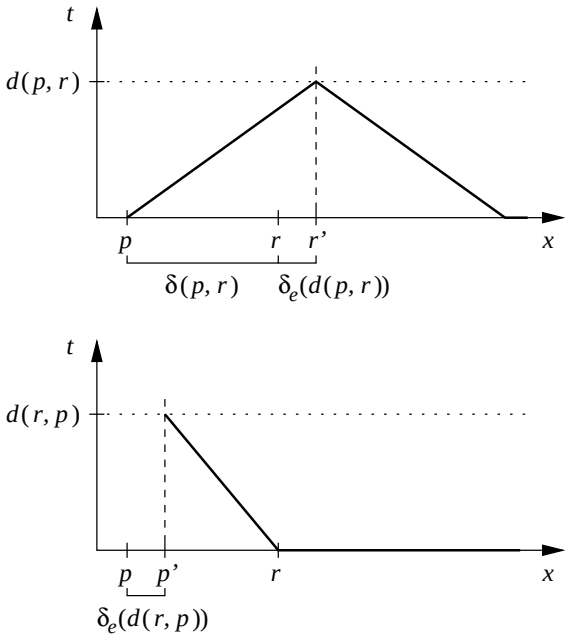

Figure 5: Temporal contours are corrected by the distance the entity travels in the communication delay. Above, the corrected temporal contour of player p, where the peak is pushed forward to r'. Below, the corrected temporal contour of player r, where the peak is pushed forward to p'.

2.2 Multiple Players

When we have multiple remote players, each of them has their own delay functions, and to get the temporal contour we must aggregate them. To realize the aggregation we can use the following approaches (see Figure 6):

1. Try to minimize the number of entities that are not in the local time (i.e., whose delay is not zero). This means that once an entity has passed a remote player, its delay returns back to zero. This approach aims at maintaining the situation as close to the actual situation as possible, and it suits best when there is a lot of interaction between the entities. The drawback is that an entity may bounce back and forth between the local and remote times, which can make its movements look jerky.

2. Try to minimize the number of delay changes. Once an entity has reached some delay level, it remains unchanged unless it begins to approach a new remote player. This helps to reduce bouncing between different times, which is prominent especially if there are several remote players along the path of the entity.

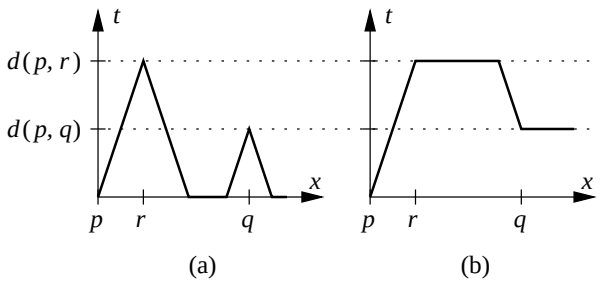

Figure 6: Two approaches to aggregate the temporal contour of player p, when there are two remote players r and q. (a) Minimize the number of entities that are not in local time. (b) Minimize the number of delay changes.

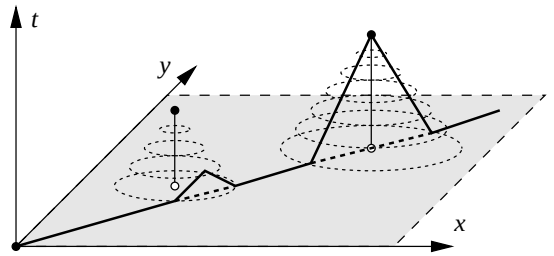

Figure 7: Two remote players with linear delay functions are aggregated to form the temporal contour. If the local player, who resides in the origin, launches an entity, it must follow the temporal contour.

The drawback is the that rendered view, in its entirety, does not remain as close to the actual situation as in the first approach.

Once we have formed the temporal contour, it is used similarly as in the case of two players (see Figure 7).

2.3 Problems and Limitations

The main limitation of local perception filters is that the players cannot interact directly with each other but all interaction must happen using passive entities. In fact, the closer the players get to each other the more noticeable the temporal distortion becomes. Finally, when they reach a *critical proximity*, even interaction using passive entities becomes impossible (see Figure 8). If we have players p and r and entity e, the critical proximity is exceeded when $\delta(p,r) < \delta_e(\max\{d(p,r), d(r,p)\})$, because now it is possible that a player learns about e too late.

Because remote players define the temporal contour, any sudden changes in their position or existence can cause drastic effects in the rendered view. For example, if a nearby remote player leaves the game world, it no longer affects the temporal contour and some entities may suddenly jump forward in time to match the updated temporal contour. Moreover, remote players can distort the rendered view just by being near the local player. Figure 9 illustrate a situation where player r has a high communication delay in comparison to players q and s. Now, if player r moves near player q, its delay function dominates the temporal contour

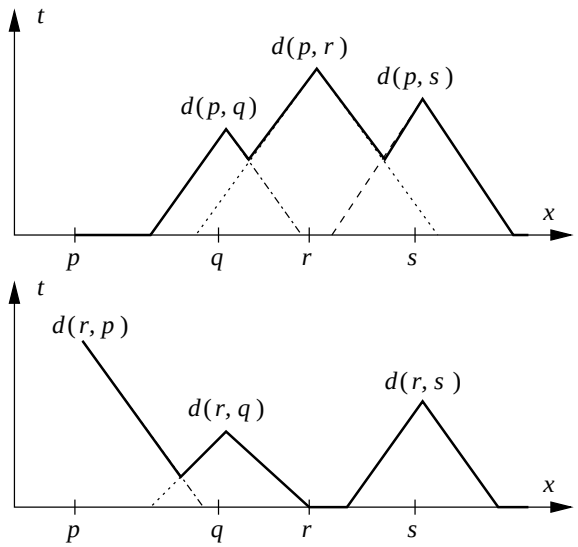

Figure 9: Four players have different communication delays. Above, the temporal contour of player p is dominated by remote player r, which alters significantly the temporal contour around remote players q and s. Below, the temporal contour of player r.

of player p in that area, and the temporal distortion around q increases.

The underlying assumption behind local perception filters is that we know the exact communication delays between the players. In reality, latency and the amount of traffic in a network tend to vary over time, which means that the height of the peaks of the temporal contour must reflect these changes. If this jitter becomes too high, the entities begin bounce back and forth in time instead of smooth temporal transitions. Naturally, we can damp this effect by changing temporal contour slowly over time.

3. ADDING BULLET TIME TO THE DELAYS

The idea of bullet time effect is that the player using bullet time has more time to react to the events of the surrounding game world. In other words, the delay between the bullet-timed player and the other players increases. Since local perception filters provide a way to realize this, it seems an obvious way to implement bullet time effect in a multiplayer game: In addition to the real-world communication delays, we have artificial, player-initiated delays—the bullet time—which are then used to form the temporal contours. The outcome is that entities approaching a bullet-timed player slow down and entities coming from a bullet-timed player speed up. Obviously, the game design should prevent the players from overusing bullet times by making it a limited resource which can be used only for a short period. Also, by incorporating the temporal distortions as an integral part of the game could lead to some interesting designs.

Let us denote the bullet time of player p with $b(p)$. As in the previous section, assume we have two players, p and r, and a bullet e shot by player p. Figure 10 illustrates the players' temporal contours when player p is using bullet time. From the perspective of player p, when the bullet reaches

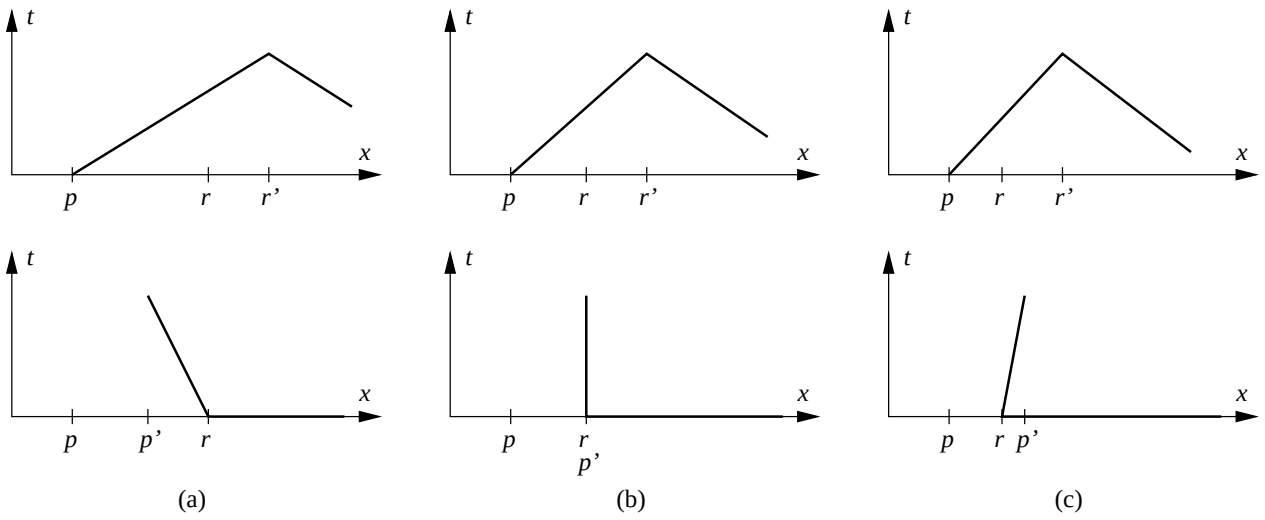

Figure 8: Critical proximity in temporal contours when player p shoots player r. Above, the perspective of player p; below, the perspective of player r. (a) The players have not reached the critical proximity. Player p sees the bullet moving slowly towards r, whilst r sees the bullet arriving fast. (b) The players have reached the critical proximity. Player r learns about the bullet exactly the same time as it passes r. Conversely, the temporal distortion of player p has not increased much. (c) The players have exceeded the critical proximity. Player r learns about the bullet only when it has already passed r, whilst the temporal distortion of player p still remains reasonable.

r the delay is $d(p,r) - b(p)$. As before, the delay function represents the temporal difference between the actual entity and rendered entity. However, whereas normally the delay values are positive (i.e., the actual position is ahead of the rendered position), bullet time can lead to negative delay values (i.e., the rendered position is ahead of the actual position). This becomes more obvious, when we consider the same situation from the perspective of player r. When the bullet reaches player r, the delay is $-b(p)$ because the bullet time, in effect, takes away time from player r. Naturally, collision detection and other reactions must be based on this rendered entity rather than the actual entity, which is still on the way to the target.

Like normal temporal contours, bullet-timed temporal contours also require refining to avoid visual disruptions. The *bullet time shadow* r'' of player r corrects the temporal contour based on the movement of e: For player p, r'' must have the property

$$\delta(r, r'') = \delta_e(d(p,r) - b(p)), \tag{5}$$

and for player r, r'' must have the property

$$\delta(r, r'') = \delta_e(b(p)). \tag{6}$$

In Figure 11 player r is using bullet time whilst being shot by player p. In this case, the bullet time $b(r)$ is added to the normal communication delay in the temporal contour of player p, which means that the delay is $d(p,r) + b(r)$ when the bullet reaches r. Conversely, player r has the delay $b(r)$ when the bullet reaches it. Again, to refine the temporal contours, we must calculate the bullet time shadow r''. For player p, r'' must have the property

$$\delta(r, r'') = \delta_e(d(p,r) + b(p)), \tag{7}$$

and for player r, r'' must have the property

$$\delta(r, r'') = \delta_e(b(r)). \tag{8}$$

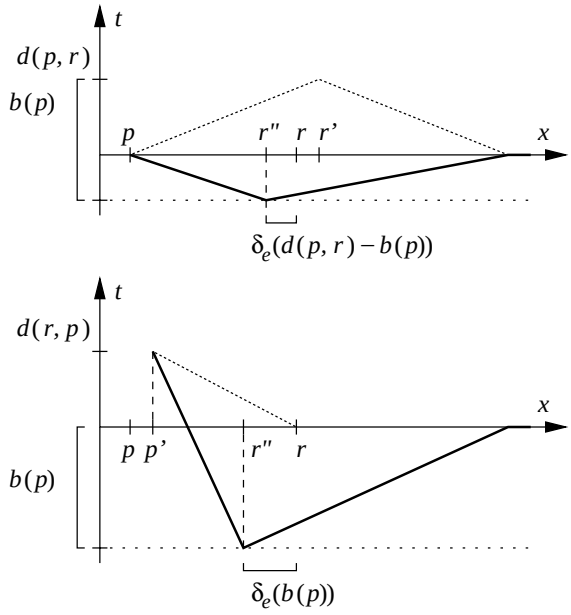

Figure 10: Player p shoots player r, and player p is using bullet time. Above, the temporal contour of player p; below, the temporal contour of player r.

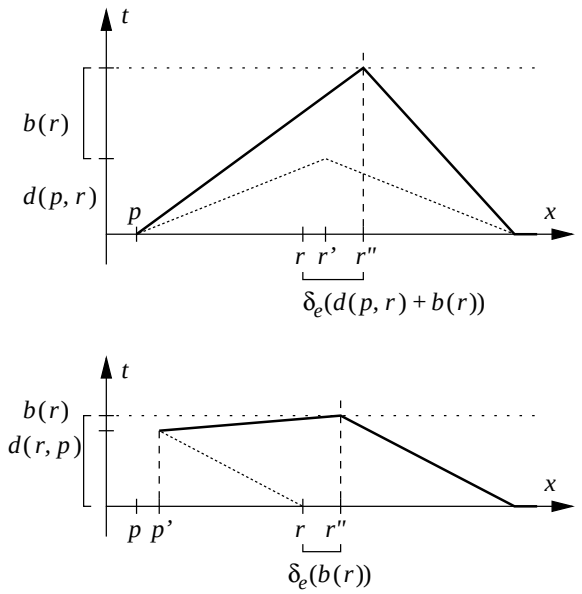

Figure 11: Player p shoots player r, and player r is using bullet time. Above, the temporal contour of player p; below, the temporal contour of player r.

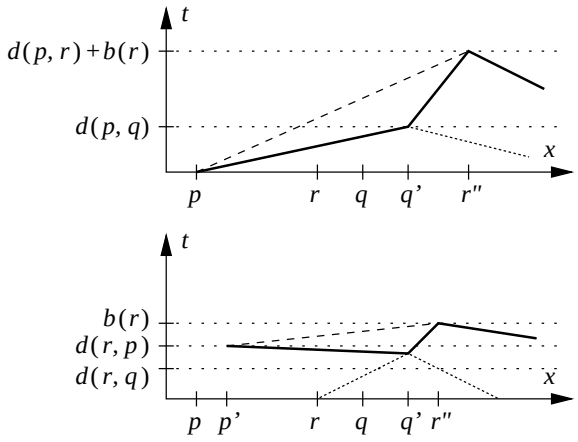

Figure 12: Player p shoots at r and q, and player r is using bullet time. Above, the temporal contour of player p; below, the temporal contour of player r.

Bullet-timed temporal contours can be generalized to multiple players the same way as normal temporal contours. However, bullet time can lead to problematic situations. Figure 12 illustrates such a case, where p shoots at r and q, when player r is using bullet time. Let us consider what happens when the bullet reaches shadow q'. At that moment player p should see the bullet rendered at the position of player q. However, before that event the bullet should have already passed player r—but that will happen only when the bullet reaches the bullet time shadow r''. This breaks the causality of events. A simple solution is to increase the delay around q, and thus give player q unearned bullet time, so that the causality is preserved.

Alternatively, we can calculate points along the entity's path where it is closest to each player. For each point, we calculate a delay interval which depends on the distance from the path. If the player is near to the path (and interaction is quite likely), the delay interval is small, and for faraway players the interval is wider. Now, instead of following the temporal contour exactly, the delay of the entity must remain within the intervals, which allows to preserve causality more fairly [1].

4. EXPERIMENTAL OBSERVATIONS

MaxMaze Demonstrator (MMD) is a testbench system for local perception filters[1] [1]. It uses linear delay functions and supports scenarios which have 2–3 players, which can move, shoot and use bullet time. MMD allows to analyse the situation frame-to-frame from the perspective of any player or from a view which combines all perspectives (see Figure 13). The information is available also in numerical format, which can viewed in the event log.

The theoretical results presented in the previous sections are mainly due to our experiments with MMD. In most cases anomalous behaviour in the testbench has revealed problems that the earlier, mostly theoretical work has not recognized. For example, the need to refine the temporal contour using the players' shadows became imminent when we observed frequent visual disruptions. Moreover, to work out the causal problems of multiple players using bullet time would have taken much longer without a tool where the solution methods could be analysed based on visual feedback. To summarize, because the local perception filter approach is about *visual perceptions*, it must be evaluated also visually.

[1]The MMD system is publicly available from the web page `http://staff.cs.utu.fi/staff/jouni.smed/mmd/`.

5. CONCLUDING REMARKS

Bullet time effect can be realized in multiplayer computer games by using local perception filters. In this paper, we described the theory behind local perception filters as well as extended the previous work by introducing refinements to the temporal contour. Bullet time alters the temporal contour by adding artificial, player-initiated delay to the normal communication delay. The results presented in this paper are inspired and studied by using a testbench program.

This paper is likely to raise more questions than give answers. Future work on extending the ideas to non-linear temporal contours is still required. Moreover, an evaluation on the subjective difference between different temporal contours would justify our claim on the usability of linear delay functions. And of course, we would like to see—and play!—multiplayer computer games utilizing the ideas presented in this paper.

6. ACKNOWLEDGEMENTS

The authors wish to express their gratitude to the late Dr Timo Kaukoranta for his initial input, inspiration, and insights.

7. REFERENCES

[1] H. Niinisalo. *MaxMaze Demonstrator Documentation.* Department of Information Technology, University of Turku, 2003.

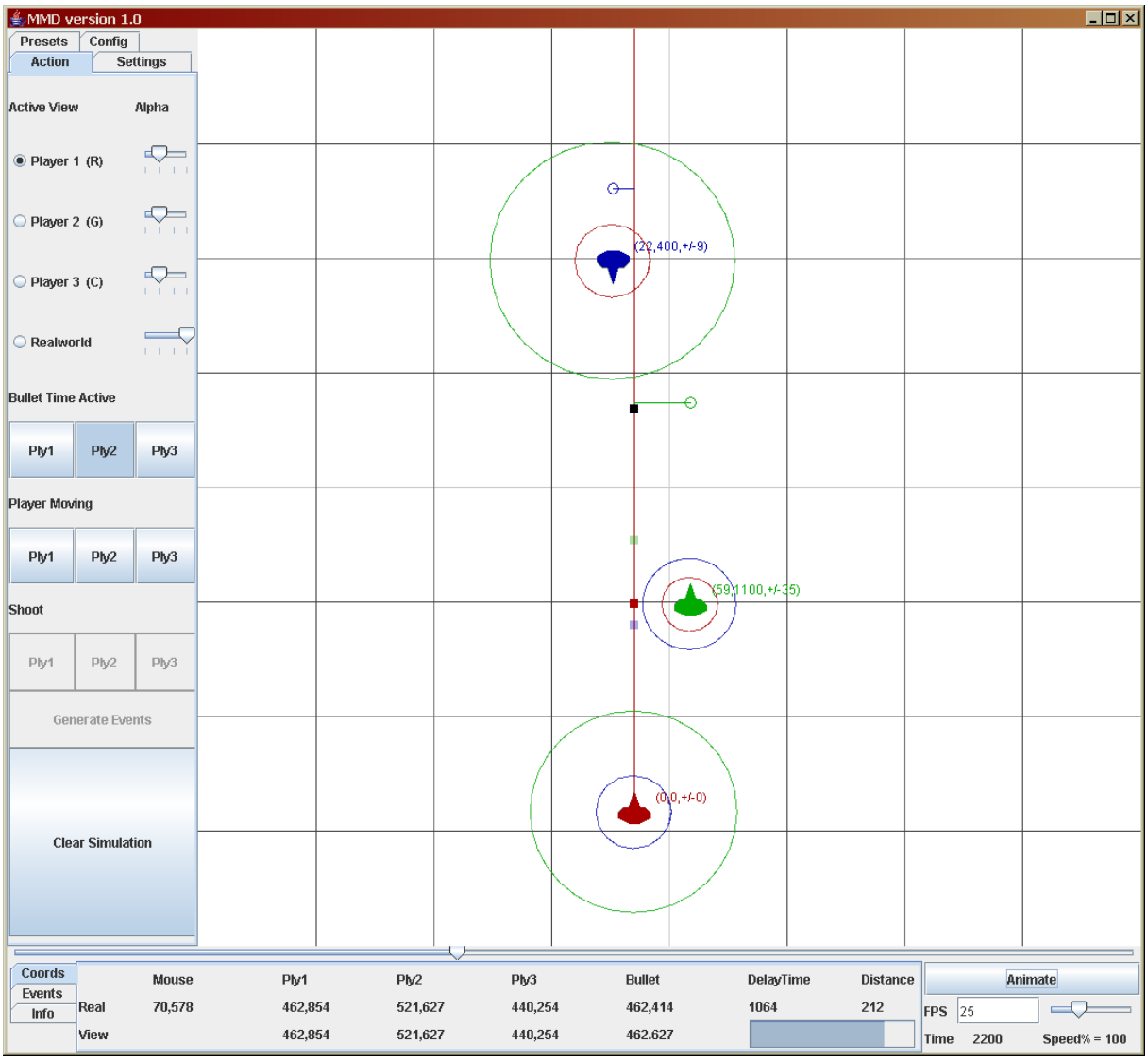

Figure 13: Screenshot of the MMD system. The player at the bottom has shot a bullet, and the view is from its perspective. The player at the middle has activated the bullet time, and the bullet is about to pass it. However, the actual bullet is well past the middle player and heading towards the player at the top.

[2] Raven Software. *Jedi Knight II: Jedi Outcast*. LucasArts, 2002.

[3] Remedy Entertainment. *Max Payne*. Gathering of Developers, 2001.

[4] M. D. Ryan and P. M. Sharkey. The causal surface and its effect on distribution transparency in a distributed virtual environment. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, volume 6, pages 75–80, Tokyo, Japan, Oct. 1999.

[5] P. M. Sharkey, M. D. Ryan, and D. J. Roberts. A local perception filter for distributed virtual environments. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 242–9, Atlanta, GA, Mar. 1998.

[6] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. Addison Wesley, Reading, MA, 1999.

[7] J. Smed, T. Kaukoranta, and H. Hakonen. Aspects of networking in multiplayer computer games. *The Electronic Library*, 20(2):87–97, 2002.

[8] J. Smed, T. Kaukoranta, and H. Hakonen. Networking and multiplayer computer games—the story so far. *International Journal of Intelligent Games & Simulation*, 2(2):101–10, 2003. Available at ⟨http://www.scit.wlv.ac.uk/∼cm1822/ijigs22.htm⟩.

[9] A. Wachowski and L. Wachowski. *The Matrix*. Warner Bros., 1999.