**[Replication in networked games: Space/time consistency (Part 3)](https://0fps.net/2014/02/26/replication-in-networked-games-spacetime-consistency-part-3/)**

Posted on [February 26, 2014](https://0fps.net/2014/02/26/replication-in-networked-games-spacetime-consistency-part-3/)by [mikolalysenko](https://0fps.net/author/mikolalysenko/" \o "View all posts by mikolalysenko)

[Last time](https://0fps.net/2014/02/17/replication-in-networked-games-latency-part-2/) in this series, I talked about latency and consistency models.  I wanted to say more about the last of these, [local perception filtering](http://www.scribd.com/doc/207432762/Sharkey-Et-Al-1998-A-Local-Perception-Filter-for-Distributed-Virtual), but ended up running way past the word count I was shooting for. So I decided to split the post and turn that discussion into today’s topic.

Causality

Sharkey and Ryan justified local perception filters based on the limitations of human perception. In this post I will take a more physical approach derived from the [causal theory of special relativity](http://en.wikipedia.org/wiki/Causal_structure). For the sake of simplicity, we will restrict ourselves to games which meet the following criteria:

1. The game exists within a [flat space-time](http://en.wikipedia.org/wiki/Minkowski_space).
2. All objects in the game are [point-particles](http://en.wikipedia.org/wiki/Point_particle).
3. All interactions are [local](http://en.wikipedia.org/wiki/Principle_of_locality).
4. There is an [upper bound on the speed of objects](http://en.wikipedia.org/wiki/Speed_of_light).

Many games meet these requirements. For example, in [Realm of the Mad God](http://store.steampowered.com/app/200210/), gameplay takes place in a flat 2D space. Objects like players or projectiles behave as point-masses moving about their center. Interactions only occur when two objects come within a finite distance of each other. And finally, no projectile, player or monster can move arbitrarily fast. The situation is pretty similar for most shooters and MMOs. But not all games fit this mold. As non-examples, consider any game with rigid body dynamics, non-local constraints or instant-hit weapons. While some of the results in this article may generalize to these games, doing so would require a more careful analysis.

Physical concepts

The basis for space-time consistency is the concept of [causal precedence](http://en.wikipedia.org/wiki/Causal_structure). But to say how this works, we need to recall some concepts from physics.

Let d>0 be the dimension of space and let \mathbb{R}^{d,1} denote a d+1 dimensional real vector space. In the [standard basis](http://en.wikipedia.org/wiki/Standard_basis) we can split any vector (x,t) \in \mathbb{R}^{d,1} into a spatial component x \in \mathbb{R}^d and a [time coordinate](http://en.wikipedia.org/wiki/Coordinate_time) t \in \mathbb{R}. Now let c be the maximum velocity any object can travel (aka the [speed of light in the game](http://en.wikipedia.org/wiki/Speed_of_light)). Then define an inner product on \mathbb{R}^{d,1},

\langle (x,t), (y,s) \rangle = \langle x,y \rangle - c^2 t s

Using this bilinear form, vectors (x,t) \in \mathbb{R}^{d,1} can be classified into 3 categories:

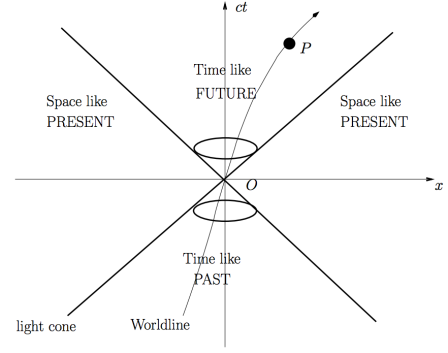
1. [*Time-like vectors*](http://en.wikipedia.org/wiki/Spacetime#Time-like_interval): \langle (x,t), (x,t) \rangle < 0
2. [*Null vectors*](http://en.wikipedia.org/wiki/Spacetime#Light-like_interval): \langle (x,t), (x,t) \rangle = 0
3. [*Space-like vectors*](http://en.wikipedia.org/wiki/Spacetime#Space-like_interval): \langle (x,t), (x,t) \rangle > 0

As a special consideration, we will say that a vector is ***causal***if it is either time-like or null. We will suppose that [time proceeds in the direction](http://en.wikipedia.org/wiki/Arrow_of_time) (0, +1). In this convention, causal vectors (x,t) \in \mathbb{R}^{d,1} are further classified into 3 subtypes:

1. *Future-directed*: t > 0
2. *Zero*: t = 0
3. *Past-directed*: < 0

In the same way that a [Euclidean](http://en.wikipedia.org/wiki/Euclidean_space) is constructed from a normed [vector space](http://en.wikipedia.org/wiki/Vector_space), space-time \mathbb{E}^{d,1} is a [psuedo-Euclidean space](http://en.wikipedia.org/wiki/Pseudo-Euclidean_space) associated to the index-1 vector space \mathbb{R}^{d,1} and its inner product. We will assume that every [event](http://www.stanford.edu/class/cs240/readings/lamport.pdf) (eg collision, player input, etc.) is associated to a unique point in space-time and when it is unambiguous, we will identify the space-time point of the event with itself. Objects in the game are points and their motions sweep out [world lines](http://en.wikipedia.org/wiki/World_line) (or [trajectories](http://en.wikipedia.org/wiki/Trajectory)) in space-time.

The relationship between these concepts can be visualized using a [Minkowski diagram](http://en.wikipedia.org/wiki/Minkowski_diagram):

[](https://0fps.files.wordpress.com/2014/02/screen-shot-2014-02-25-at-1-52-39-pm1.png)

A Minkowski diagram showing the relationship between time-like, space-like and null vectors. Image taken from [Roger Horsely](http://www2.ph.ed.ac.uk/~rhorsley/)‘s [lecture notes on classical mechanics](http://www2.ph.ed.ac.uk/~rhorsley/SI11-12_socm/lec18.pdf). (c) Roger Horsley 2011-2012.

If you prefer a more interactive environment, Kristian Evensen made a browser based Minkowski diagram demo:

K. Evensen. (2009) “[An interactive Minkowski diagram](http://www.trell.org/div/minkowski.html)“

Space-time consistency

With the above physical language, we are now ready to define the space-time consistency as it applies to video games.

*Given a pair of events, p, q \in \mathbb{E}^{d,1} we say that p****causally precedes****q (written p \preceq q) if p-q is future-directed causal or zero.*

Causal precedence is a [partial order](http://en.wikipedia.org/wiki/Partially_ordered_set) on the points of space-time, and it was observed by Zeeman to [uniquely determine the topology and symmetry of space-time](http://download.wpsoftware.net/causality-lorentz-group-zeeman.pdf). (Note: These results were later [extended by Stephen Hawking et al. to an arbitrary curved space-time](http://authors.library.caltech.edu/11027/1/HAWjmp76.pdf), though based on our first assumption we will not need to consider such generalizations.) Causal precedence determines the following consistency model:

*An ordered sequence of events p_0, p_1, ... p_n\in \mathbb{E}^{d,1} is****space-time consistent****if for all 0 \leq i, j \leq n,  p_i \preceq p_j \implies i \leq j.*

Space-time consistency is a special case of [causal consistency](http://en.wikipedia.org/wiki/Causal_consistency). Relativistic causal precedence is stricter than [causality consistency,](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.3356&rep=rep1&type=pdf) because it does not account for game specific constraints on interactions between objects. For example, a special effect might not influence any game objects, yet in a relativistic sense it causally precedes all events within its [future light cone](http://en.wikipedia.org/wiki/Light_cone). Still space-time consistency is more flexible than strict temporal consistency, and as we shall see this can be exploited to reduce latency.

Cone of uncertainty

As a first application of space-time consistency, we derive a set of sufficient conditions for [dead-reckoning](http://en.wikipedia.org/wiki/Dead_reckoning) to correctly predict a remote event. The basis for this analysis is the geometric concept of a [light cone](http://en.wikipedia.org/wiki/Light_cone), which we now define:

Any [closed regular set](http://en.wikipedia.org/wiki/Solid_modeling#Mathematical_foundations) S \subseteq \mathbb{E}^{d,1} determines a pair of closed regular sets called its *causal future* and *causal past*, respectively:

* *Causal future*: J^+ (S)= \left \{ p \in \mathbb{E}^{d,1} : \left ( \exists q \in S : q \preceq p \right ) \right \}
* *Causal past*: J^- (S)= \left \{ p \in \mathbb{E}^{d,1} : \left ( \exists q \in S : p \preceq q \right ) \right \}

According to our assumptions about the direction of time, if an event had any causal influence on an event in S, then it must be contained in J^-(S). Conversely, events in S can only influence future events in J^+(S). When the set S is a singleton, then J^+(S) is called the *future light cone* of S, and J^-(S) is the *past light cone*, and the set J^+(S) \cup J^-(S) is the *light cone* of S.

The causal future and past are [idempotent](http://en.wikipedia.org/wiki/Idempotence) operators,

J^+ (J^+(S)) = J^+ (S),

J^- (J^-(S)) = J^- (S).

Sets which are closed under causal past/future are called *closed causal sets*. If S is a closed causal set, then so is its [regularized complement](http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/model/reg-bool-op.html),

J^- \left ( \overline{J^+ (S)^c } \right ) = \overline{ J^+ (S)^c },

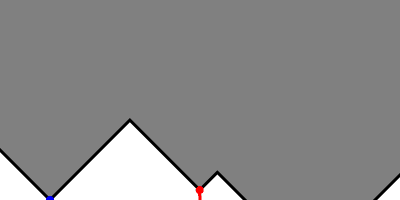
J^+ \left ( \overline{J^- (S)^c } \right ) = \overline{ J^- (S)^c }.

Now, to connect this back to dead-reckoning, let us suppose that there are two types of objects within the game:

1. Active entities: Whose motion is controlled by non-deterministic inputs from a remote machine
2. Passive entities: Whose trajectory is determined completely by its interactions with other entities.

For every active entity, we track an event r_i = (x_i, t_i) which is the point in space-time at which it most recently received an input from its remote controller. Let R = \{ r_0, r_1, ... \} be the set of all these events. We call the causal future J^+(R) the [cone of uncertainty](http://en.wikipedia.org/wiki/Cone_of_Uncertainty). Events outside the cone of uncertainty are causally determined by past observations, since as we stated above J^-( \overline{J^+(R)^C} ) = \overline{J^+(R)^C}. Consequently, these events can be predicted by dead-reckoning. On the other hand, events inside J^+(R) could possibly be affected by the actions of remote players and therefore they cannot be rendered without using some type of optimistic prediction.

Here is a Minkowski diagram showing the how the cone of uncertainty evolves in a networked game as new remote events are processed:

[](https://0fps.files.wordpress.com/2014/02/cone-of-uncertainty1.gif)

The cone of uncertainty illustrated. In this Minkowski diagram, the vertical axis represents time. The world lines for each active object are drawn as colored paths. The grey region represents the cone of uncertainty, and is updated as new remote events are processed.

Cauchy surfaces

In this section we consider the problem of turning a (d+1)-dimensional collection of world lines in space-time into a d-dimensional picture of the state of the world. The geometry of this process is encoded by a [Cauchy surface](http://en.wikipedia.org/wiki/Cauchy_surface). Intuitively, a Cauchy surface captures an instant in space-time as it is perceived by some observer. The rendered state of objects are determined by the points at which their world lines intersect this surface. That these intersections are well-defined motivates the following definition:

*A hypersurface S \subseteq \mathbb{E}^{d,1} is a****Cauchy surface****if every time-like curve extends to one which intersects S exactly once.*

This is not the only way to define a Cauchy surface. Equivalently,

***Proposition 1:****Any Cauchy surface S partitions a flat space-time into 3 regions:*

* *The interior of the causal future: \text{int}(J^+(S))*
* *The interior of the causal past: \text{int}(J^-(S))*
* *And S itself*

*If S is a maximal set with these properties, then S is a Cauchy surface.*

In a flat space-time, Cauchy surfaces can be parameterized by spatial coordinates. Let S be a Cauchy surface and \phi_S : \mathbb{E}^d \to \mathbb{E},

\phi_S(x) = \min_{ (x,t) \in S } t.

Then,

S = \{ (x, \phi_S(x)) \in \mathbb{E}^{d,1} : x \in \mathbb{E}^d \}.

The inverse question of when a function \phi : \mathbb{E}^d \to \mathbb{E} determines a Cauchy surface is answered by the following theorem:

***Theorem 1:****A function \phi : \mathbb{E}^d \to \mathbb{E} determines a Cauchy surface S = \{ (x, \phi(x) ) : x \in \mathbb{E}^d \} if and only if the subgradient of \phi is bounded by \frac{1}{c}.*

**Proof**: To show that this condition is sufficient, it is enough to prove that any time parameterized time-like curve x : \mathbb{E} \to \mathbb{E}^d intersects the surface only once. To show that the curve crosses at least once, recall proposition 1 implies that \phi partitions \mathbb{E}^{d,1} into 3 regions, \{ (x, t) : t < \phi(x) \}, \{ (x, \phi(x) \} and \{ (x,t) : \phi(x) < t \}, and that the curve begins in the causal past of S and ends in the causal future, so by the intermediate value theorem the must cross S. Now let (x,t) be a point of intersection between the curve and the surface.  Then the entire future of the curve is contained in the open cone \text{int} (J^+( \{ (x,t) \} )). Similarly, because \nabla_{\dot{x}(t)} \phi (x(t)) < \frac{1}{c}, no other points on S intersect \text{int}( J^+ ( \{ (x,t) \}) ). Ditto for all the points in the causal past, and so any intersection must be unique. To show necessity, take any Cauchy surface S and construct the field \phi_S as shown above. Suppose that there is some point x \in \mathbb{E}^d and unit vector v \in \mathbb{R}^d where |\nabla_{v} \phi_S(x)| > \frac{1}{c}. Without loss of generality, assume \phi_S(x) = 0. Then there exists some \epsilon > \delta > 0 where \phi_S (x + \epsilon v) > \frac{\epsilon}{c - \delta}. Construct the time-like curve q(t) = \left \{ \begin{array}{cc} x & \text{if } t < \delta \\ x + (c - \delta) v t & \text{otherwise} \\ \end{array} \right.. By the intermediate value theorem, q crosses S in two locations and so S is not Cauchy. **QED**

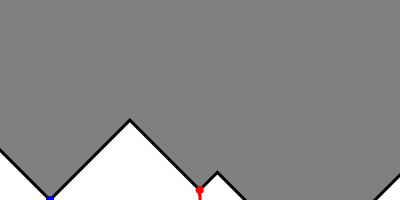
Consistency revisited

Rendering a game amounts to selecting a Cauchy surface and intersecting it with the world lines of all objects on the screen. The spatial coordinates of these intersections are then used to draw objects to the screen. From this perspective one can interpret a consistency model as determining a Cauchy surface. We now apply this insight to the [three consistency models which were discussed last time](https://0fps.net/2014/02/17/replication-in-networked-games-latency-part-2/).

Starting with strict consistency, define t_{\min} = \min_{(x_i, t_i) \in R} t_i be the time of the oldest-most recent input from a remote player.  Then define a constant Cauchy surface,

\phi_{\text{strict}}(x) = t_{\min} .

As we did with the cone of uncertainty, we can visualize the evolution of this Cauchy surface with a Minkowski diagram:

[](https://0fps.files.wordpress.com/2014/02/strict-cauchy1.gif)

A visualization of the Cauchy surface for strict consistency. The orange line represents the Cauchy surface viewed by the local (red) player. Note that this is some time behind the most recently acknowledged input.

The same analysis applies to optimistic consistency. Let t_{\text{current}} be the time that the most recent local input was processed. Then the optimistic Cauchy surface is,

\phi_{\text{optimistic}}(x) = t_{\text{current}} .

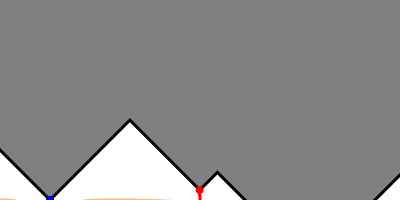
Which gives the following Minkowski diagram:

[](https://0fps.files.wordpress.com/2014/02/optimistic-cauchy1.gif)

An optimistic Cauchy surface (shown in orange). The surface tracks the local red player closely, giving more responsive inputs. Unfortunately, the optimistic Cauchy surface intersects the cone of uncertainty and so it requires local prediction to extrapolate the position of remote objects.

Unlike in the case of the strict consistency, the optimistic causal surface intersects the cone of uncertainty. As a result, it requires prediction to extrapolate the state of remote entities.

Finally, here is a Minkowski diagram showing the Cauchy surface of a local perception filter:

[](https://0fps.files.wordpress.com/2014/02/lpf-cauchy1.gif)

The Cauchy surface for a local perception filter. Note that this surface closely follows the local player, yet does not intersect the cone of uncertainty.

Time dilation

Local perception filters make finer geometric tradeoffs between local responsiveness and visual consistency. They achieve this by using a curved Cauchy surface. This has a number of advantages in terms of improving responsiveness, but introduces [time dilation](http://en.wikipedia.org/wiki/Time_dilation) as a side effect. In a game, this time dilation will be perceived as a virtual acceleration applied to remote objects. To explain this concept, we need to make a distinction between local time and coordinate time. *Coordinate time*, t, is the time component of the coordinates of an event in Minkowski space-time. *Local time*, \tau, is the time that an event is rendered for a local user.

Now suppose that we have a Cauchy surface, \phi(x,\tau) varying as function of local time. We want to compute the time dilation experienced by an object with a coordinate time parameterized world line, q : \mathbb{E} \to \mathbb{E}^d. In this situation, define the mapping \hat{t} : \mathbb{E} \to \mathbb{E}, that relates local time to the particle’s coordinate time,

\hat{t}(\tau) = \phi(q( \hat{t}(\tau) ), \tau).

The *time dilation* observed in the particle is the ratio of change in coordinate time to change in local time, or by the chain rule,

\frac{d \hat{t}}{d \tau} = \dot{\phi} ( q, \tau) + \nabla \phi (q, \tau) \cdot \dot{q}(\hat{t}).

In general, we would like $\frac{d \hat{t}}{d \tau}$ to be as close to 1 as possible. If the time dilation factor ever becomes \leq 0, then objects will stop or possibly move backwards in time. This condition produces jitter, and it is important that we avoid it in all circumstances. Fortunately, it is not hard to derive a set of sufficient conditions to ensure that this is the case for all time-like paths:

***Theorem 2:****If \dot{\phi} > 0 and | \nabla \phi | < \frac{1}{c}, then for any time-like particle the time dilation is strictly positive.*

The first condition is natural, since we can assume that the Cauchy surface is strictly moving forward in time.  Similarly, according to Theorem 1 the second condition is the same as the requirement that \phi determines a Cauchy surface. If these conditions are violated, then it is possible for objects to jitter or even reverse direction. This is directly analogous to the situation in physics, where [if an object travels faster than light it can move backwards in time](http://en.wikipedia.org/wiki/Time_travel#Time_travel_to_the_past_in_physics). The easy fix in a game is to just modify the rules to ensure that this does not ever happen.

Intersecting world lines

Another tricky issue is finding the intersection of the world lines with the Cauchy surface. For special choices of world lines and Cauchy surfaces, it is possible to compute these intersections in closed form. For an example, in the original local perception filter paper Sharkey and Ryan carry out this analysis under the assumption that the world line for each particle is a polynomial curve. In general though it is necessary to use a numerical method to solve for the local time of each object. Fortunately, the requirements of space-time consistency ensure that this is not very expensive. Given a world line q as before, we observe the following [monotonicity](http://en.wikipedia.org/wiki/Monotonic_function) properties:

***Theorem 3:****For any Cauchy surface \phi and time-like curve q:*

* *If \hat{t} > \phi( q(\hat{t}), \tau), then for all u > \hat{t}, u > \phi(q(u), \tau).*
* *If \hat{t} < \phi( q(\hat{t}), \tau), then for all u < \hat{t}, u < \phi(q(u), \tau)*
* *There is only one \hat{t} where \hat{t} = \phi(q(\hat{t}), \tau)*

As a result, we can use [bisection](http://en.wikipedia.org/wiki/Bisection_method) to find \hat{t} to O(n) bits of precision in O(n) queries of the Cauchy surface. In pseudocode, we have the following algorithm:

findWorldLineIntersection(phi, q, t0, t1, n):

for i=0 to n:

t = (t0 + t1) / 2

if t > phi(q(t)):

t1 = t

else:

t0 = t

return t0

This is exponential order convergence, and is about as fast as one can hope to achieve. Here phi is a function encoding the Cauchy surface as described above, q is coordinate time parameterized curve representing the world line of the particle, t0 and t1 are upper and lower bounds on the intersection region and n is the number of iterations to perform. Higher values of n will give more accurate results, and depending on the initial choice of t0 and t1, for floating point precision no more than 20 or so iterations should be necessary.

Limitations of local perception filters

While local perception filters give faster response times than strict surfaces, they do not always achieve as low a latency as is possible in optimistic prediction. The reason for this is that we do not allow the local perception filter to cross into the cone of uncertainty. If the local player passes into this region, then they will necessarily experience input lag.

We can quantify the situations where this happens. As a first observation, we note that the boundary of the cone of uncertainty is a Cauchy surface and so it can be described parametrically. That is define a function h : \mathbb{E}^d \to \mathbb{E} where,

h(x) = \min_{(x_i, t_i) \in R} \frac{|x - x_i|}{c}+t_i.

We will call h the *horizon*. If the last locally processed input was at (x_0, t_0) \in \mathbb{E}^{d,1}, then the minimum input delay of the local perception filter is,

\Delta = t_0 - h(x_0).

The magnitude of \Delta is a function of the last acknowledged position and ping of all remote players. If \Delta = 0, then we can respond to local inputs immediately. This leads to the following theorem which gives necessary conditions for a local input to be processed immediately:

***Theorem 4:****If the round trip ping of each remote player is \Delta_i and their last acknowledged position is x_i, then in order for a local perception filter to process a local input without lag, we must require that:*

*c \Delta_i < |x_i - x_0|*

Under this interpretation, each remote player sweeps out a sphere of influence,

S_i = \left \{ x \in \mathbb{E}^d : c \Delta_i \leq |x - x_i| \right \}

And if the local player passes into a sphere of influence they will experience lag which is proportional to their distance to the remote player. The size of these spheres of influence is determined by the ping of each remote player and the speed of light. As a result, players with a higher ping will inflict more lag over a larger region of space. Similarly, increasing the speed of light expands the spheres of influence for all players and can thus cause the local player to experience more input latency. So, the moral of this theorem is that if a game has a lot of close quarters combat or fast moving projectiles, then local perception filters might not be much help. On the other hand, if you can ensure that players are separated by at least c \Delta distance from each other, then local perception filters completely eliminate input lag.

Software engineering issues

In general, it can be difficult to bolt networked replication on top of an existing game. The same is true of local perception filters. At minimum, a game engine must have the following features:

* Decoupled rendering
* Deterministic updates
* Persistence

Each of these capabilities requires architectural modifications. These changes are easy to implement if incorporated early in the design process, and so if networking is a priority then it pays to deal with them up front.

Decoupled rendering

Though this is standard practice today, it is especially important in a network game that the updates are not tied to rendered frames. Not only does this allow for fast and fluid animations regardless of the update rate, but it also makes games easier to debug by making time rate dependent behaviors deterministic. Glenn Fiedler popularized this idea in his writing on networked games:

G. Fiedler. (2006) “[Fix your time step!](http://gafferongames.com/game-physics/fix-your-timestep/)“

That article covers the basics pretty well, though it glosses over the somewhat some subtle issues like input handling. In a fixed time step game it is more natural to use [polling](http://en.wikipedia.org/wiki/Polling_(computer_science)) to handle inputs instead of [asynchronous interrupts (ie events)](http://en.wikipedia.org/wiki/Interrupt). In an environment like the DOM which defaults to the latter, it is necessary to enqueue events and process them within the main loop.

In some applications, like virtual reality, it may be important to respond to certain user inputs immediately. For example, the current position of the user’s head must be taken into account before rendering the latest frame or else the player will experience motion sickness. In this situation it is important that these inputs have a limited effect on the state of the game, or that their effects on the state can processed at a lower latency.

Adding local perception filters does not change the update step, but it does modify how the game is rendered.  In psuedocode, this is what the drawFrame procedure might look like:

drawFrame(localTime):

phi = constructLocalPerceptionFilter(localTime)

x[] = rendered positions of all objects

for each object i:

x[i] = q[i](findWorldLineIntersection(phi,q[i],t0,t1,20))

drawObjects(x)

Deterministic updates

In order for a server to handle inputs from remote clients in any order, it is necessary that game updates are deterministic. That is, we require that given some list of events and a state, there is a function next that completely determines the successor state:

\text{update} : \text{state} \times \text{events} \to \text{state}.

Deterministic updates also simplify testing a game, since it is possible to record and play back a sequence of events exactly. The cost though is that all system state including the value of each random number generator must be passed as input to the update function.

Persistence

Lastly, local perception filters need to maintain the history of the game. This representation should support efficient queries of the world lines of each object at arbitrary points in coordinate time and branching updates (since remote events can be processed out of order). As a sketch of how this can be implemented, refer to the following article by Forrest Smith:

F. Smith. (2013) “[The Tech of Planetary Annihilation: ChronoCam](http://forrestthewoods.com/the-tech-of-planetary-annihilation-chronocam/)“

At a high level, the ChronoCam system used by [Planetary Annihilation](https://www.kickstarter.com/projects/659943965/planetary-annihilation-a-next-generation-rts) gives a streaming persistent history all world lines in the game. In addition to being a necessary component in realizing local perception filters, Forrest Smith observes that maintaining a persistent history gives the following incidental benefits:

* Robust demo recordings – Since only positions are saved, demos are all upwards compatible and seeking/playback is very efficient
* Bandwidth savings – Fewer position updates can be sent to compensate for low capacity connections and paths can be simplified to save space
* Cheating prevention – The state of hidden units does not need to be replicated

The general problem of storing and maintaining the history of a system is known in data structures as “[persistence](http://en.wikipedia.org/wiki/Persistent_data_structure)“.  In general, any data structure in the [pointer-machine model](http://en.wikipedia.org/wiki/Pointer_machine) with bounded in-degree can be made into a persistent version of the same data structure with only O(1) overhead. This transformation is described in the following paper:

J.R. Driscoll, N. Sarnak, D.D. Sleator, R.E. Tarjan. (1989) “[Making data structures persistent](http://www.cs.cmu.edu/~sleator/papers/Persistence.htm)” JCSS

While the DSST transformation is automatic, in practice it requires a certain amount of human finesse to apply correctly. One of the main sticking points is that bounded in-degree rules out certain classes of objects like iterators. Still, it is a useful idea and understanding it gives the basic tools necessary to implement persistent data structures.

Functional programming

Finally, I will conclude this section by observing that we can get all of the above features automatically if we use [functional programming](http://en.wikipedia.org/wiki/Purely_functional). For example, “deterministic” in the sense we are using it is just another word for functional purity. Similarly, data structure persistence is a weaker requirement than [total immutability](http://www.cs.cmu.edu/~rwh/theses/okasaki.pdf). Thus it stands to reason that if we took the more aggressive position and functionalized everything that we would get all the above features completely by accident. While I’m not sure how practical it is to write a game this way, it seems plausible that if functional programming becomes more popular, then local perception filters may see wider use within the game programming community.

A simplified shooter

To demonstrate these principles I made simple interactive demonstration. You can try it out here:

[Demo!](http://mikolalysenko.github.io/local-perception-filter-demo/)

One thing which I did not cover in detail was how to choose the Cauchy surface for the local perception filter in the first place. The main reason I didn’t bring this up was that the word count on this post had already spiraled out of control, and so I had to cut some material just to get it finished. I may revisit this topic in a later post and compare some different options.

In the demo, the local perception filter is implemented using the following function of the horizon:

\phi(x) = \min(t_0, h(x))

Where t_0 is the time of the local player. Some other options would be to use a smoother surface or some offset of the true horizon. At any rate, this surface seems to give acceptable results but I fully expect that it could be improved drastically with more careful consideration.

Conclusion

This post elaborates on the ideas sketched out by Sharkey and Ryan. In particular, we handle the problem of intersecting world lines with the Cauchy surface in general and give precise conditions on the geometry of the Cauchy surface for it to be jitter free. I think that this is the first time that anyone has proposed using a binary search to find the coordinate time for each world line intersection, though it is kind of an obvious idea in hindsight so I would not be surprised if it is already known. Additionally the precise interpretation of special relativity as applied to video games in this post is new, though the conceptual origin is again in Sharkey and Ryan’s paper.

In closing, local perception filters are a promising approach to latency hiding in networked games. Though they cannot eliminate lag in all cases, in games where most interactions are mediated by projectiles they can drastically reduce it. Understanding space-time consistency and local perception filtering is also helpful in general, and gives insight into the problems in networked games.

Next time

In the next post I want to move past latency and analyze the problem of bandwidth. It will probably be less mathematical than this post.