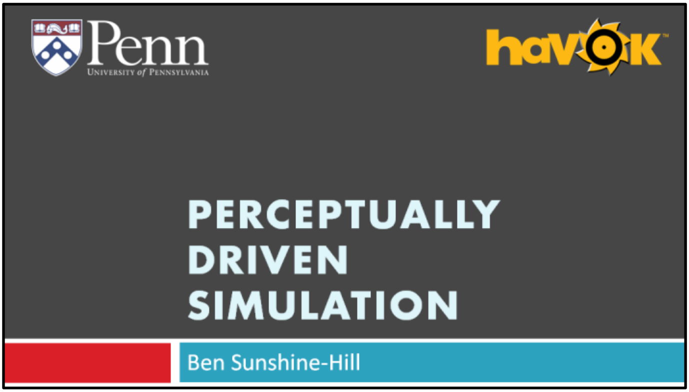


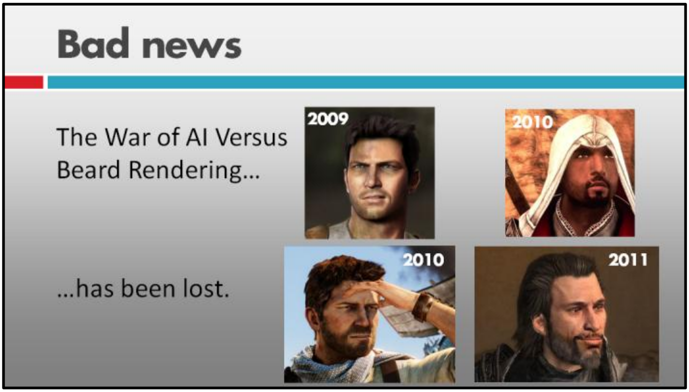
Penn
UNIVERSITY of PENNSYLVANIA

havok

PERCEPTUALLY DRIVEN SIMULATION

Ben Sunshine-Hill

Hi, I'm Ben Sunshine-Hill. I'm a software developer at Havok. I'm going to talk a little bit about some work I did at Penn, on "perceptually driven simulation".



Bad news

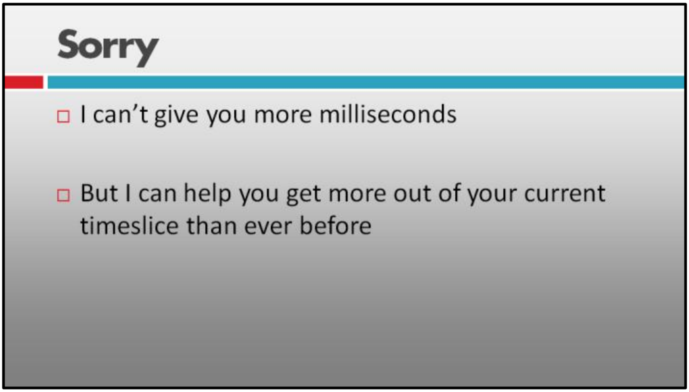
The War of AI Versus Beard Rendering…

…has been lost.

2009
2010
2010
2011

So a quick show of hands -- who here makes AI for a living?
Who wishes they had more CPU time to devote to running better AI?
Who thinks the games THEMSELVES might be MORE FUN if those processor-grubbing graphics programmers didn't steal every spare millisecond just to [X] render the main character's stubble more realistically?
[X] Well, bad news, kids. The war of AI

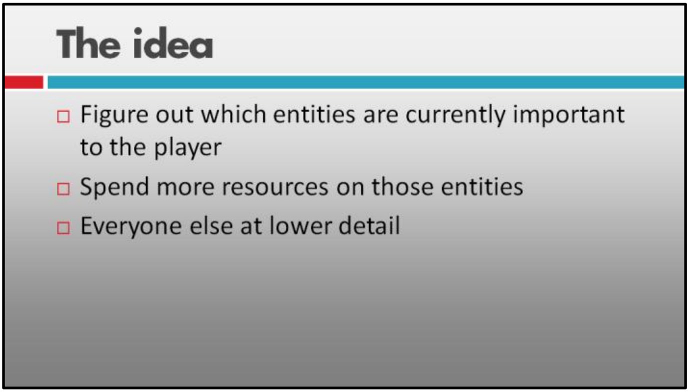
versus beard rendering has
been lost.


Sorry

- I can’t give you more milliseconds
- But I can help you get more out of your current timeslice than ever before

I’m Sorry. I [X] can’t give you more milliseconds.

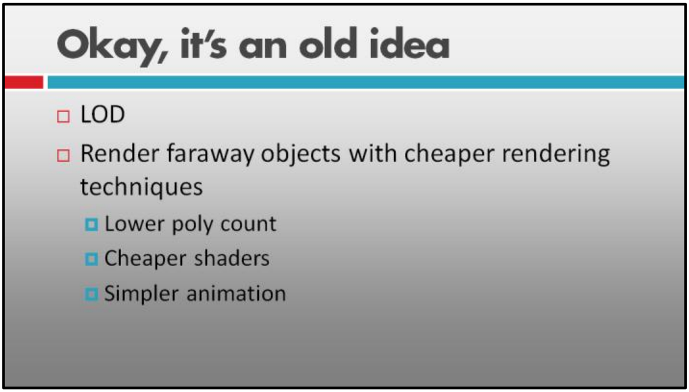
What I can do, is help you get [X] more computational power out of your current timeslice than ever before.



The idea

- Figure out which entities are currently important to the player
- Spend more resources on those entities
- Everyone else at lower detail

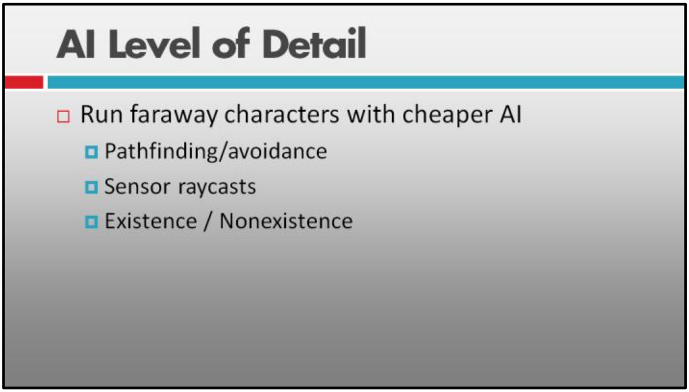
The idea is to [X] figure out which entities are currently important, and [X] spend more resources on those entities, giving them more detail, with all the [X] other entities simulated with cheaper techniques.



Okay, it's an old idea

- LOD
- Render faraway objects with cheaper rendering techniques
  - Lower poly count
  - Cheaper shaders
  - Simpler animation

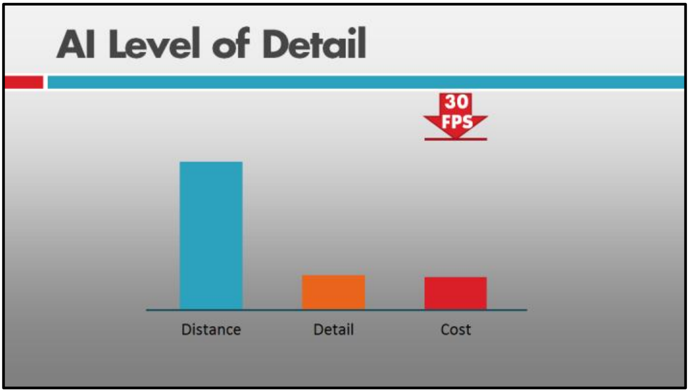
It's not a new idea, of course; this is just [X] LOD. The graphics guys [X] came up with this first, back when you couldn't really afford to render more than, like, eight triangles per frame.



AI Level of Detail

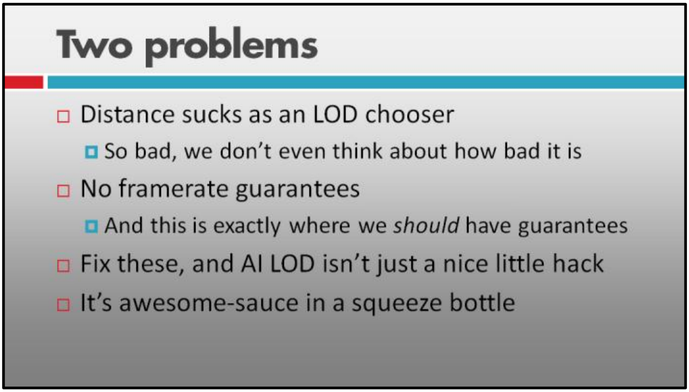
- Run faraway characters with cheaper AI
  - Pathfinding/avoidance
  - Sensor raycasts
  - Existence / Nonexistence

And it’s established in AI as well; we [X] run faraway characters with cheaper AI. Like, we’ll use [X] cheaper, stupider path following techniques, or [X] cheap out on determining what some entity can or can’t see, or like I talked about last year, [X] destroy faraway characters entirely, as they get outside the simulation bubble.



AI Level of Detail

So as an entity gets closer to us, we bump up the detail level, which of course means we’re spending more on that entity. And we try to pick this switchover distance so that in [X] general we don’t blow our framerate budget.



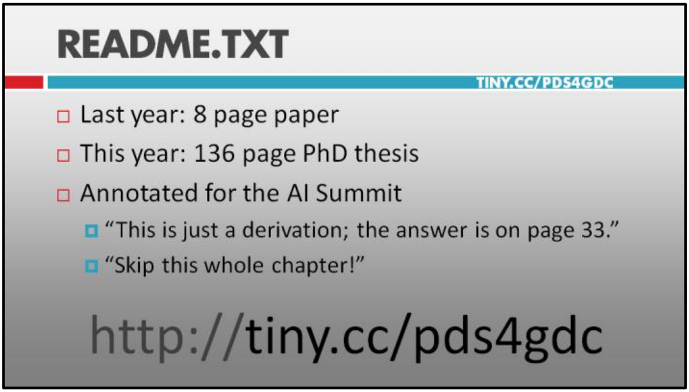
Two problems

- Distance sucks as an LOD chooser
  - So bad, we don't even think about how bad it is
- No framerate guarantees
  - And this is exactly where we *should* have guarantees
- Fix these, and AI LOD isn't just a nice little hack
- It's awesome-sauce in a squeeze bottle

But there's a couple of problems. First of all, distance [X] sucks as an LOD chooser. Seriously, it's [X] so bad we don't even notice how bad it is. And we don't actually get any [X] guarantee that we're doing enough detail reduction, or that we're not doing [X] too much, [X] and this is exactly what LOD should be giving us! That's what it's for!

…

If we can fix these problems, [X] AI level of detail isn't just a nice little hac.
[X]
…
It's something more.

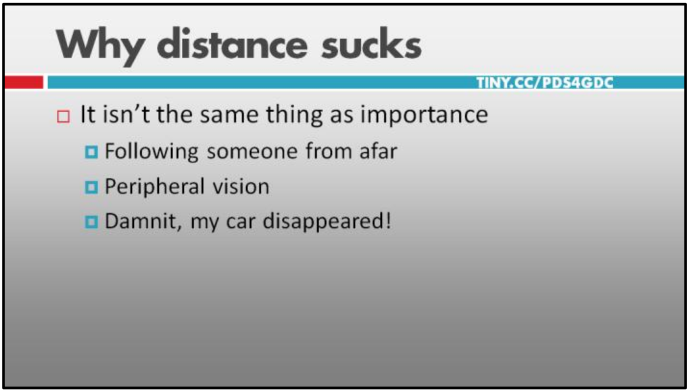

README.TXT

TINY.CC/PDS4GDC

- Last year: 8 page paper
- This year: 136 page PhD thesis
- Annotated for the AI Summit
  - "This is just a derivation; the answer is on page 33."
  - "Skip this whole chapter!"

http://tiny.cc/pds4gdc

Now, last year when I was here, talking about alibi generation, I threw up a url for an [X] 8-page paper which described all the math and stuff, so I didn't have to go into complete detail during the talk. This year, I'm afraid, it's a [X] 136-page PhD thesis. But fear not! Just for you guys, I've put together an [X] annotated version which tells you which parts you do and do not need to actually READ, with gems like [X] "this is just a derivation”, and [X] “skip this whole chapter”. I’ll [X] stick the URL up there for you.



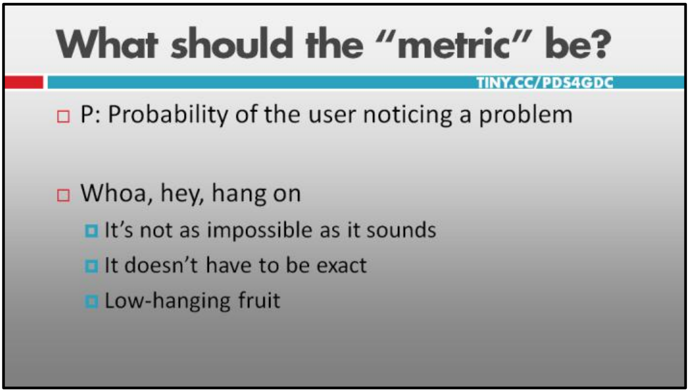
Why distance sucks

TINY.CC/PDS4GDC

- It isn't the same thing as importance
  - Following someone from afar
  - Peripheral vision
  - Damnit, my car disappeared!

So I said some pretty mean things about using distance for picking level of detail. Why? Well, basically, [X] it's not the same thing as importance. It's not WHAT. WE. WANT. If you've played Assassins Creed, you know that the guy you're most interested in is often [X] way out in front of you, and some random guy right next to you, you [X] barely even notice. And if you've played GTA, of course you [X] KNOW what happens if you forget to

park your car in one of
those “Don’t delete this
car” parking spaces.



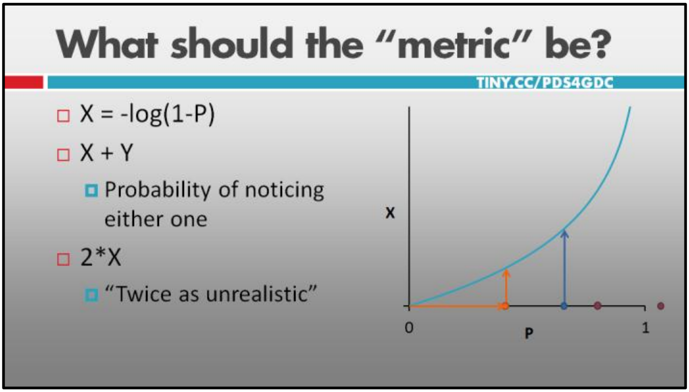
What should the "metric" be?

TINY.CC/PDS4GDC

- P: Probability of the user noticing a problem
- Whoa, hey, hang on
  - It's not as impossible as it sounds
  - It doesn't have to be exact
  - Low-hanging fruit

So if not distance, what's the actual NUMBER there? Because we NEED a number. What's the metric of importance?
I say, it's [X] the probability of the user noticing a problem! The probability that the actual player, sitting in front of the game, will go "hey, that's not right"!

Now [X] hold on a second, I see you guys shaking your heads. [X] It's not as impossible as it sounds. We're [X] not actually trying to compute an exact probability. We're going for the [X] low hanging-fruit here; we can do a pretty bad job of calculating it, and it'll still be much better than what we've got now.

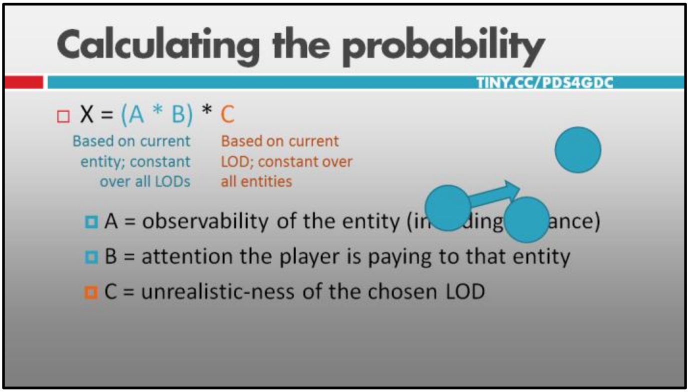


What should the "metric" be?

TINY.CC/PDS4GDC

- X = -log(1-P)
- X + Y
  - Probability of noticing either one
- 2*X
  - "Twice as unrealistic"

Incidentally, for the sake of the math, what we'll measure is not actually P, the probability, but [X] X, this log-scaled thing, and the graph of it [X] looks like this. And the nice thing about X is that we can do things like [X] add it. Of course we generally can't [X] add probabilities, the sum might not even be a valid probability number. But if we [X] add these log-scaled numbers for two low-detail entities, what we get out of it is the probability of noticing either one. And likewise, now we can say things like [X] 2 times X! That’s now the same thing as saying [X] “twice as unrealistic”; it’s the probability of noticing it if we were to do it twice.


Calculating the probability

TINY.CC/PDS4GDC

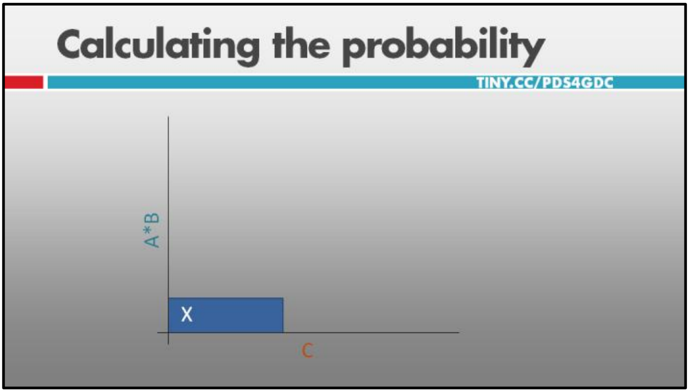
- X = (A * B) * C

  Based on current entity; constant over all LODs

  Based on current LOD; constant over all entities

  - A = observability of the entity (in ding ance)
  - B = attention the player is paying to that entity
  - C = unrealistic-ness of the chosen LOD

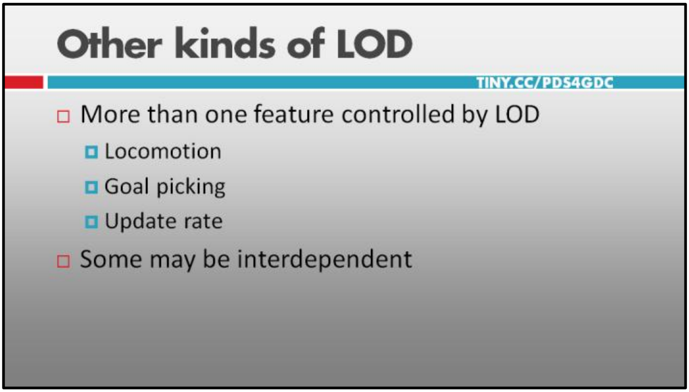
So for something like the probability of noticing that we're not doing [X] real collision avoidance for a character, that they're walking right through people, we might [X] calculate that as the product of these three numbers, where [X] A is how visible and observable the entity is (and this is where distance still figures in, in some small way), [X] B is how much attention the player's paying to it, and [X] C is some constant expressing just how crappy it is to have people walking through people instead of around them. So [X] A and B are a measure of importance of that entity, no matter how we're simulating it, and [X] C is for that LOD and is the same for any entity with that LOD, and their product is the answer.



Calculating the probability

TINY.CC/PDS4GDC

So whenever the [X] A and B go up, the probability of the user noticing our cheap-ass tricks goes up, [X] until we pick an LOD with a lower multiplier, so the total probability stays low.

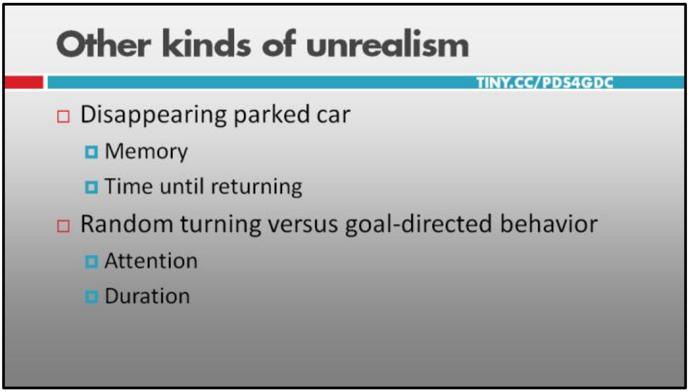


Other kinds of LOD

TINY.CC/PDS4GDC

- More than one feature controlled by LOD
  - Locomotion
  - Goal picking
  - Update rate
- Some may be interdependent

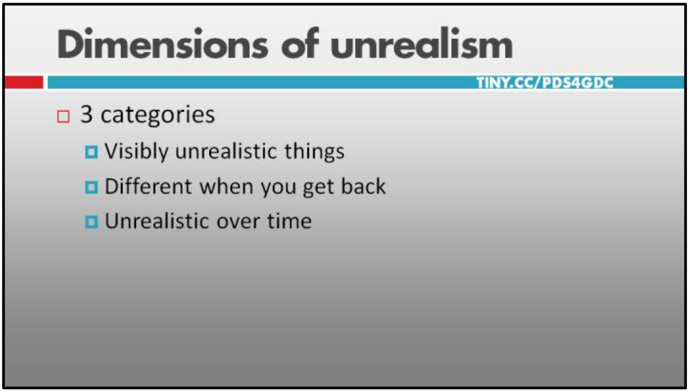
But there’s not just the one rectangle, because a single entity has [X] more than one kind of LOD; we can pick how it [X] does locomotion, and how it does [X] goal picking, and [X] how often it updates, all separately, and there might be some levels that [X] require other levels, like you can’t do high-quality pathfinding if the guy doesn’t have a goal.


Other kinds of unrealism

TINY.CC/PDS4GDC

- Disappearing parked car
  - Memory
  - Time until returning
- Random turning versus goal-directed behavior
  - Attention
  - Duration

And not everything fits the "observation times attention" model. I mean, the [X] disappearing parked car I mentioned isn't something we see out of the corner of our eye, it's unrealistic even if it happens offscreen. The probability for THAT is based on [X] how much we remember the car, and [X] how much we'll STILL remember it if and when we get back. Or if the guy in front of us is actually doing some [X] goal driven thing or is just turning randomly, and that depends on [X] how much attention we’re paying, and [X] how long we’ve been watching him. The paper at that [X] link up there goes into simple models for all of these factors.

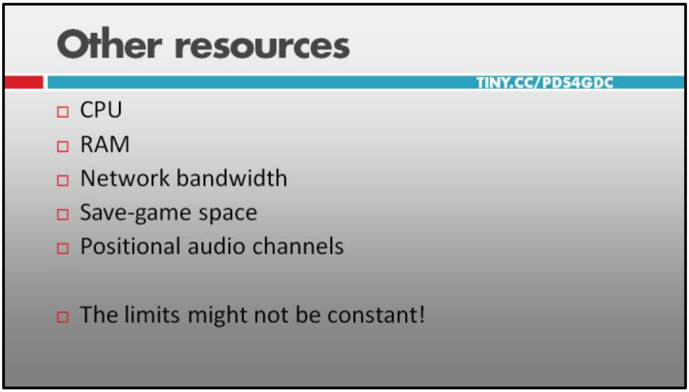


Dimensions of unrealism

TINY.CC/PDS4GDC

- 3 categories
  - Visibly unrealistic things
  - Different when you get back
  - Unrealistic over time

And so we've identified these three categories, these three "dimensions of unrealism" – You might disagree with how we broke them up, this stuff is definitely open to interpretation – and most of the ways we ratchet down AI detail, and there's a lot of them, cause one or more of these three, in varying amounts. And an entity, depending on his relationship to the player, is prone to each of these in varying amounts – An entity who just came around the corner and he’s right in front of us, we might not notice if he’s [X] wandering randomly, but we’ll sure notice if he has [X] foot-skate. So he has this [X] multidimensional importance based on those different factors I mentioned, and his various LODs, added together make a [X] corresponding vector of simulation quality. And the cool thing about that log-scaled probability is, all we have to do is find the [X] dot product of those two vectors, and the result is the [X] total probability for that entity.



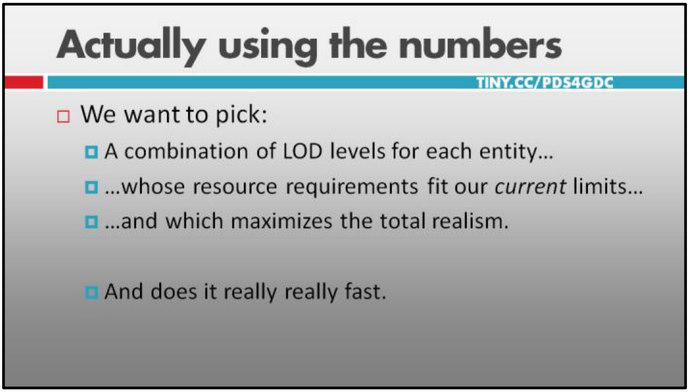
Other resources

TINY.CC/PDS4GDC

- CPU
- RAM
- Network bandwidth
- Save-game space
- Positional audio channels

- The limits might not be constant!

Oh, and just to make things harder, maybe we don’t want to keep just [X] CPU under control. We might want to make sure we don’t blow our [X] RAM budget too. And there are [X] other things that maybe we care about limiting, depending on what sort of a game this is and what our requirements are, and [X] the limits might go up or down when other things happen in our game. So, like, some background recalculation starts, and all of a sudden we have to spend less CPU, and we need to adjust for that.



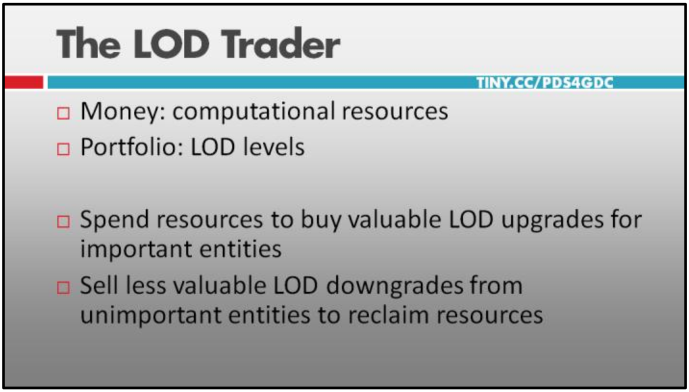
Actually using the numbers

TINY.CC/PDS4GDC

- We want to pick:
  - A combination of LOD levels for each entity…
  - …whose resource requirements fit our *current* limits…
  - …and which maximizes the total realism.

  - And does it really really fast.

So to sum up: [X] we want to pick: a [X] combination of levels for each entity so that the [X] resources fit all our CURRENT limits, and we want as [X] much realism, as low a probability of the player noticing a problem, as possible. Oh, [X] and we wanna do it really fast.



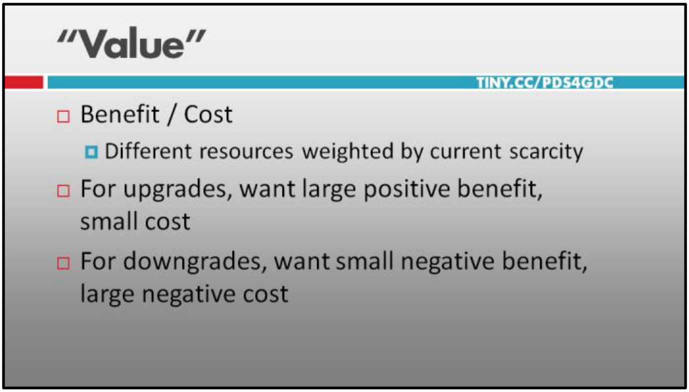
The LOD Trader

TINY.CC/PDS4GDC

- Money: computational resources
- Portfolio: LOD levels

- Spend resources to buy valuable LOD upgrades for important entities
- Sell less valuable LOD downgrades from unimportant entities to reclaim resources

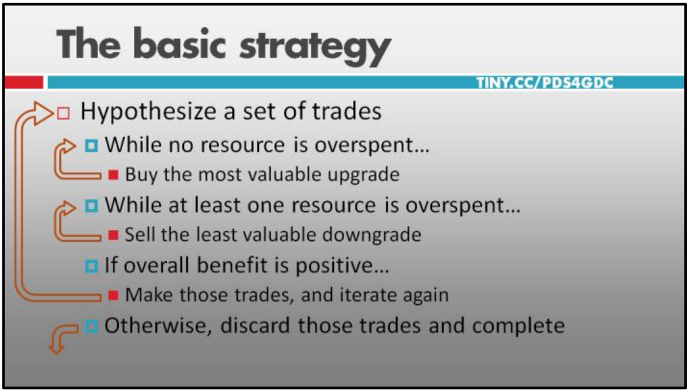
Which brings us to the LOD trader. I call it the LOD trader because it’s kind of like a stock trader; it has [X] computational resources as its money, and a [X] portfolio of how all the entities are being simulated. It can [X] spend resources to increase the LOD of valuable entities, but if it runs out of money it needs to [X] sell off less valuable LODs to get back under budget.


“Value”

TINY.CC/PDS4GDC

- Benefit / Cost
  - Different resources weighted by current scarcity
- For upgrades, want large positive benefit, small cost
- For downgrades, want small negative benefit, large negative cost

So what do I mean by “value” of an LOD change? Basically [X] the benefit divided by the cost. There’s different resources, remember, and we [X] weight them by how scarce they are currently. For upgrades, we want [X] high benefit, and only spending a little bit more, and for downgrades, we want [X] to only lose a bit of realism, and get a lot of resources back.

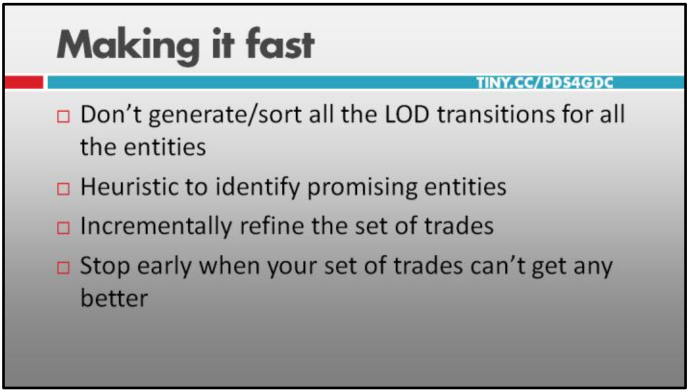


The basic strategy

TINY.CC/PDS4GDC

- Hypothesize a set of trades
  - While no resource is overspent…
    - Buy the most valuable upgrade
  - While at least one resource is overspent…
    - Sell the least valuable downgrade
  - If overall benefit is positive…
    - Make those trades, and iterate again
  - Otherwise, discard those trades and complete

The basic strategy is, [X] we come up with a hypothetical set of trades. As [X] long as we have resources left, we [X] buy the most valuable upgrade, and then [X] to pay for our excesses, we [X] sell the least valuable downgrade. And [X] then we see if this set of trades made things better or worse, and if it's [X] better we actually make the trades and then loop back, and [X] otherwise we discard the trades and we're done for this run-through.

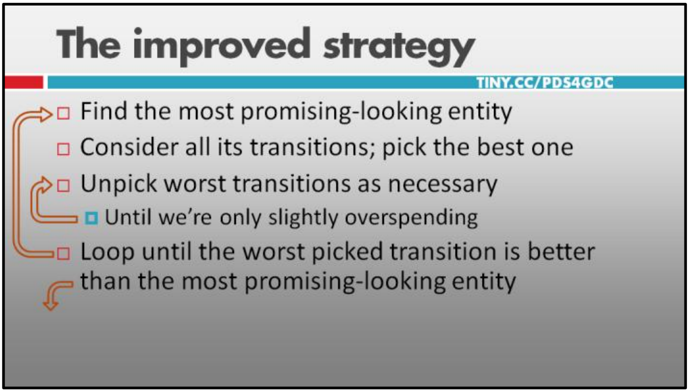


Making it fast

TINY.CC/PDS4GDC

- Don’t generate/sort all the LOD transitions for all the entities
- Heuristic to identify promising entities
- Incrementally refine the set of trades
- Stop early when your set of trades can’t get any better

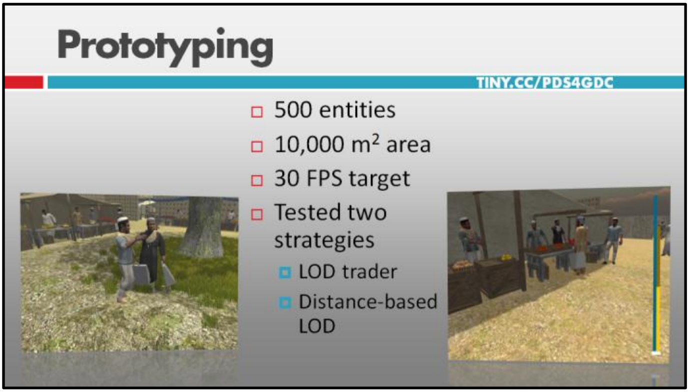
Now we want this to be really fast, so we [X] don’t actually generate all the possible transitions for all the entities, that’d be millions of them. We [X] use a heuristic to find entities which probably have valuable upgrades or downgrades, and we [X] refine our set of trades as we look at the most promising entities, and [X] at a certain point we find that we don’t need to look at any other entities so we can stop.


The improved strategy

TINY.CC/PDS4GDC

- Find the most promising-looking entity
- Consider all its transitions; pick the best one
- Unpick worst transitions as necessary
  - Until we're only slightly overspending
- Loop until the worst picked transition is better than the most promising-looking entity

In this improved algorithm, we [X] pop off the next promising entity, we [X] look at its transitions and pick the best one, and then we [X] undo transitions we picked before, so that we [X] don't overspend by too much. And we [X] keep looking at entities until the ones that are left can't possibly have better transitions than the ones we've chosen already, which in practice happens quite quickly.

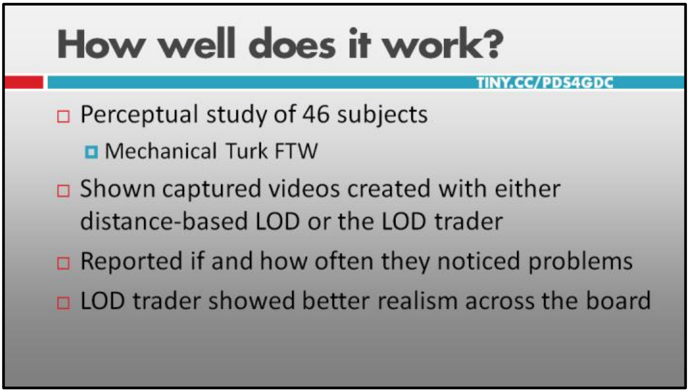


Prototyping

TINY.CC/PDS4GDC

- 500 entities
- 10,000 m^2 area
- 30 FPS target
- Tested two strategies
  - LOD trader
  - Distance-based LOD

We prototyped these methods out in a simulation of an outdoor marketplace, with [X] about five hundred people, roaming around [X] ten thousand square meters of level, with the target being [X] 30 frames per second. So it wasn't huge, but it was large enough that LOD makes a difference. We tested out [X] two LOD picking strategies: The [X] LOD trader, as well as [X] conventional distance-based LOD picking.

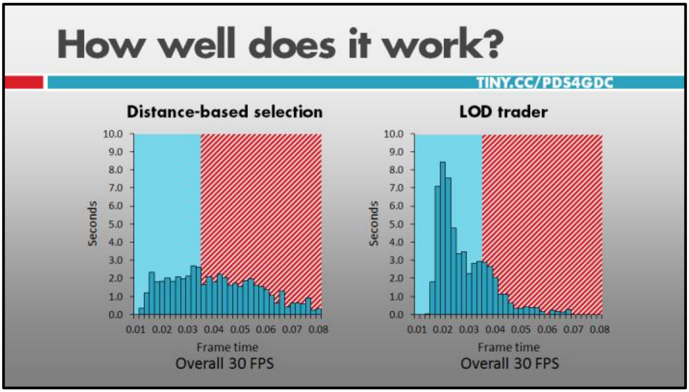


How well does it work?

TINY.CC/PDS4GDC

- Perceptual study of 46 subjects
  - Mechanical Turk FTW
- Shown captured videos created with either distance-based LOD or the LOD trader
- Reported if and how often they noticed problems
- LOD trader showed better realism across the board

So how well does it work? [X] well, we tested it out on a bunch of people [X] through Amazon's Mechanical Turk service (which is really great for these sorts of experiments, by the way), [X] showing them videos from the simulation with either LOD picking method – it would have been better if they could play the game, but we didn't want to have to worry about system requirements – [X] and they reported how often they noticed low-detail events in the simulation. And what we found [X] was that the LOD trader did better across the board than distance-based picking.

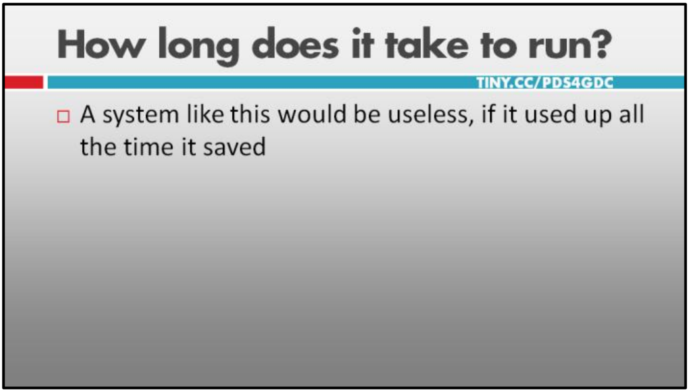


How well does it work?

TINY.CC/PDS4GDC

And it wasn't just realism that got better. [X] Our distance based selection was fine at maintaining an AVERAGE framerate, but in crowded areas the framerate really suffered, and in sparse areas it still ratcheted down realism for faraway agents, even when it didn't need to. The [X] LOD trader was a lot better at staying under the minimum frames per second, and in sparse areas, basically everyone was being simulated at highest quality,

because there was ROOM
for that.

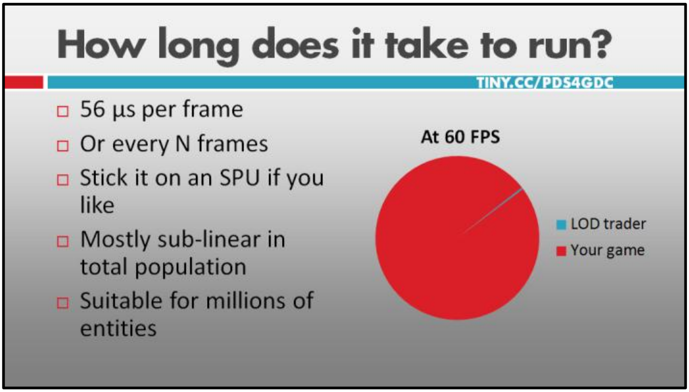


How long does it take to run?

TINY.CC/PDS4GDC

- A system like this would be useless, if it used up all the time it saved

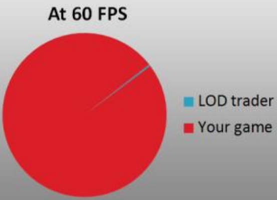
So, but what are you paying for all this? [X] We don’t want all this sorting and trading to use up the performance gains.



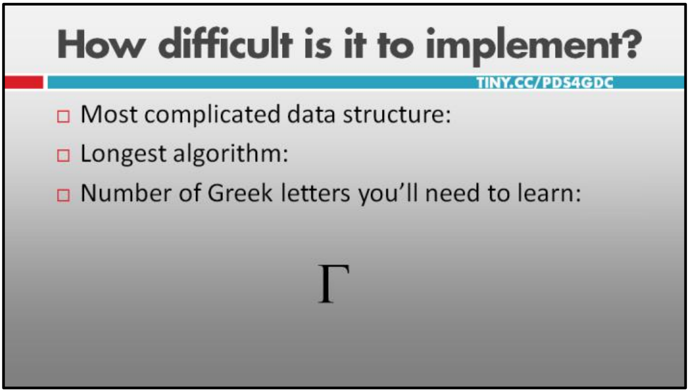
How long does it take to run?

TINY.CC/PDS4GDC

- 56 µs per frame
- Or every N frames
- Stick it on an SPU if you like
- Mostly sub-linear in total population
- Suitable for millions of entities



Well, in our simulation? [X] 56 microseconds. That's MICROseconds. As in, [X] less than one percent of our CPU budget if we're doing it sixty times a second. [X] And really, that's a silly thing to do– if we need it even cheaper, we can just run it every N frames. And we can even [X] stick it on an SPU if we want, it's great for that because it doesn't do much communication with the rest of the game. This is for five hundred agents, but we found that [X] the time grows sublinearly in the total WORLD population, the faraway entities are controlled pretty much for free. It’ll scale [X] well into the millions of entities.

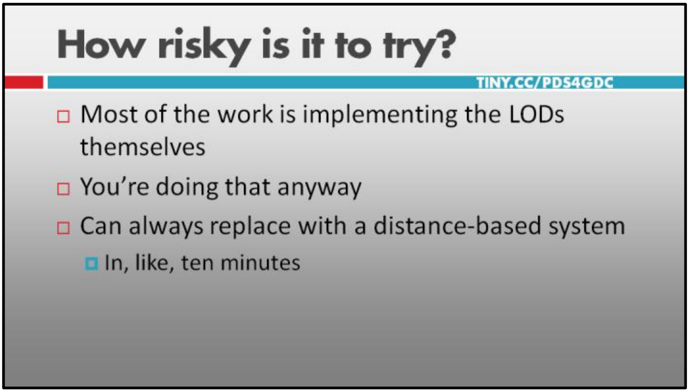

How difficult is it to implement?

TINY.CC/PDS4GDC

- Most complicated data structure:
- Longest algorithm:
- Number of Greek letters you'll need to learn:

Γ

Another reasonable question to ask about a doctoral thesis work is, how difficult would it be to actually make this? Because let me tell you, the alibi generation stuff I talked about last year had some really complex math backing it up. Well, good news this year. [X] The most complicated data structure you'll need? [X] A standard priority queue. The [X] longest algorithm involved is only [X] sixteen lines of pseudocode. And the [X] number of new Greek letters you’ll need to memorize? [X] One. … [X] This is an upper-case gamma. There you go.

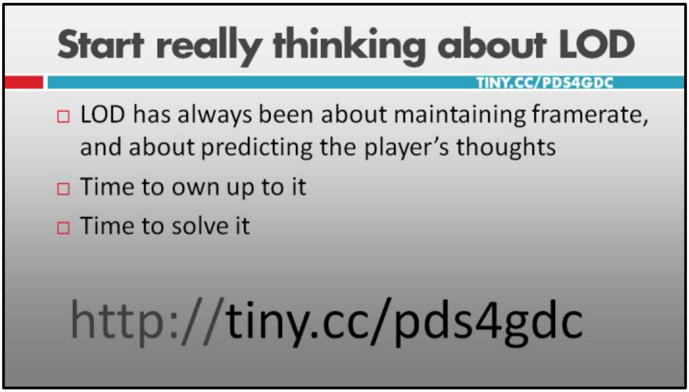


How risky is it to try?

TINY.CC/PDS4GDC

- Most of the work is implementing the LODs themselves
- You're doing that anyway
- Can always replace with a distance-based system
  - In, like, ten minutes

And what you really want to know is, how risky is it to try this out? What if it's useless and you've wasted your time implementing it? Well, most of the [X] work is implementing the LODs themselves, [X] and you're doing that anyway, you can always [X] replace the LOD trader with a distance based system if it turns out not to meet your needs. When we did that for testing, it took us like [X] ten minutes of coding to switch over.



Start really thinking about LOD

TINY.CC/PDS4GDC

- LOD has always been about maintaining framerate, and about predicting the player's thoughts
- Time to own up to it
- Time to solve it

http://tiny.cc/pds4gdc

You don't have to try out this LOD trader stuff, but if you take away one lesson, let it be this: [X] LOD has ALWAYS been about maintaining framerate, and about predicting what the player will and won't notice. Developers have largely ignored this fact, because it's scary to think of it as a global optimization problem we need to solve, and it's definitely scary to think about trying to predict the player's thoughts. But this is still what we're trying to do. It’s [X] time to own up to the fact that this is the objective. It’s time for you to [X] solve it.

[X] Thanks, everyone. We’ll take any questions you might have.