GDC 09

learn
network
inspire

www.GDConf.com

Game Developers Conference®
March 23-27, 2009 | Moscone Center, San Francisco

The PlayStation®3's SPUs in the Real World

A KILLZONE 2 Case Study

Michiel van der Leeuw

Technical Director - Guerrilla Games

TAKEAWAY

- Cool stuff you can do on SPUs
- Things that worked or didn't work for us
- Practical advice on approaching your SPUs
- Some food for thought

KILLZONE 2

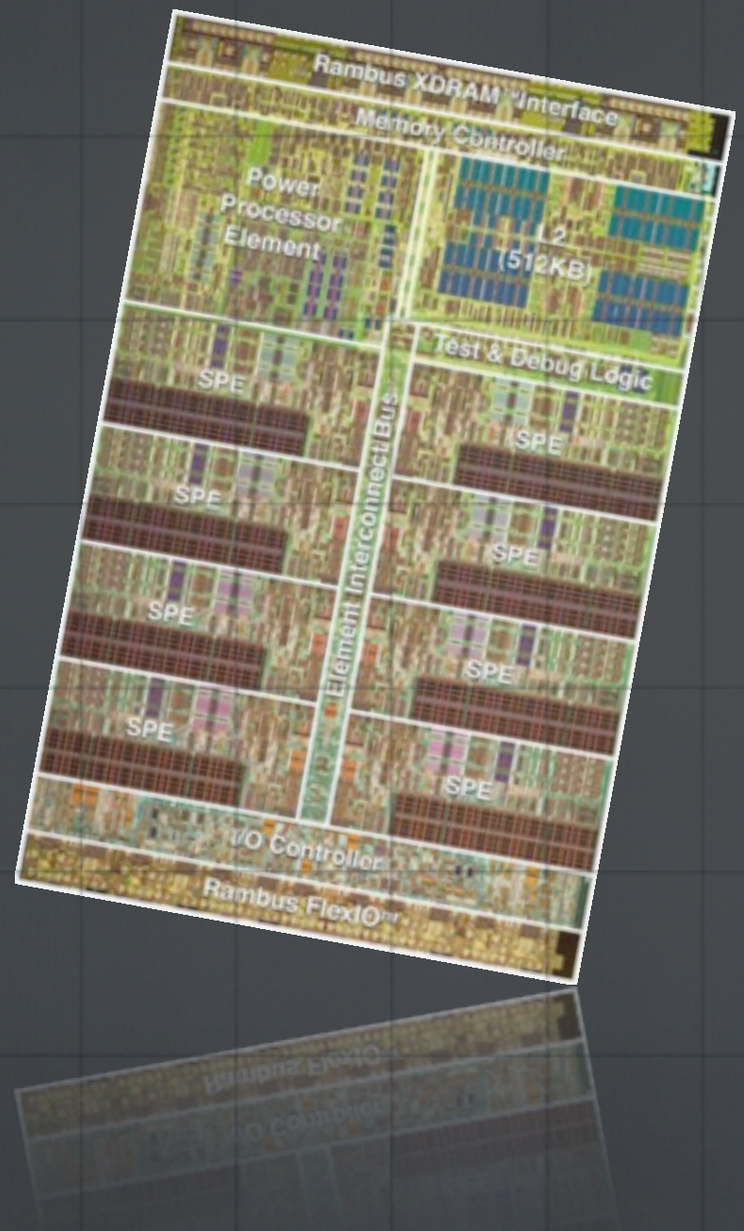
- Announced E3 2005
- Preproduction until end 2006
- Production until end 2008
- 18 months Full Production

- 140 Guerrilla Games Staff
- 50 Sony Staff

- 27 Programmers

SPUs OVERVIEW

- 6 x 3.2GHz of processing power
- Complete instruction set, general purpose
- 256K embedded memory per SPU
  - No instruction or data cache!
- Very fast DMA in/out
- Libraries for scheduling



KILLZONE 2 SPU USAGE

KILLZONE 2 SPU USAGE

- Animation
- AI Threat prediction
- AI Line of fire
- AI Obstacle avoidance
- Collision detection
- Physics
- Particle simulation
- Particle rendering
- Scene graph
- Display list building
- IBL Light probes
- Graphics post-processing
- Dynamic music system
- Skinning
- MP3 Decompression
- Edge Zlib
- etc.

44 Job types

Killzone 2 - SPU usage in cut scene

SPUs in GRAPHICS

DISPLAYLIST BUILDING

- Entire rendering engine is data-driven
  - No calls to `virtual void Draw()`
- All objects keep `MeshInstanceTree` up-to-date
  - Lightweight data structure
  - Nodes describe:
    - Mesh hierarchy
    - LOD selection rules
    - Visibility filtering (1st person shadow...)
  - Leafs describe:
    - Renderstate
    - Primitive info (vertex arrays, etc.)
- PPU hands view and projection matrix to SPU job

LIGHT PROBE SAMPLING

- Purpose: Make dynamic objects blend in with environment

- ~ 2500 static light probes per level
  - Created offline during lightmap rendering
  - Stored as 9x3 Spherical Harmonics in KD-tree

- When object requires Image-Based Lighting (IBL)
  - A job is added to sample lighting for that object
  - Finds four closest light probes in KD-tree
  - Interpolates linearly using inverse-distance weights
  - Rotates into view space
  - Create 8x8 spherical mapped texture for sampling

IBL Placement

IBL Placement

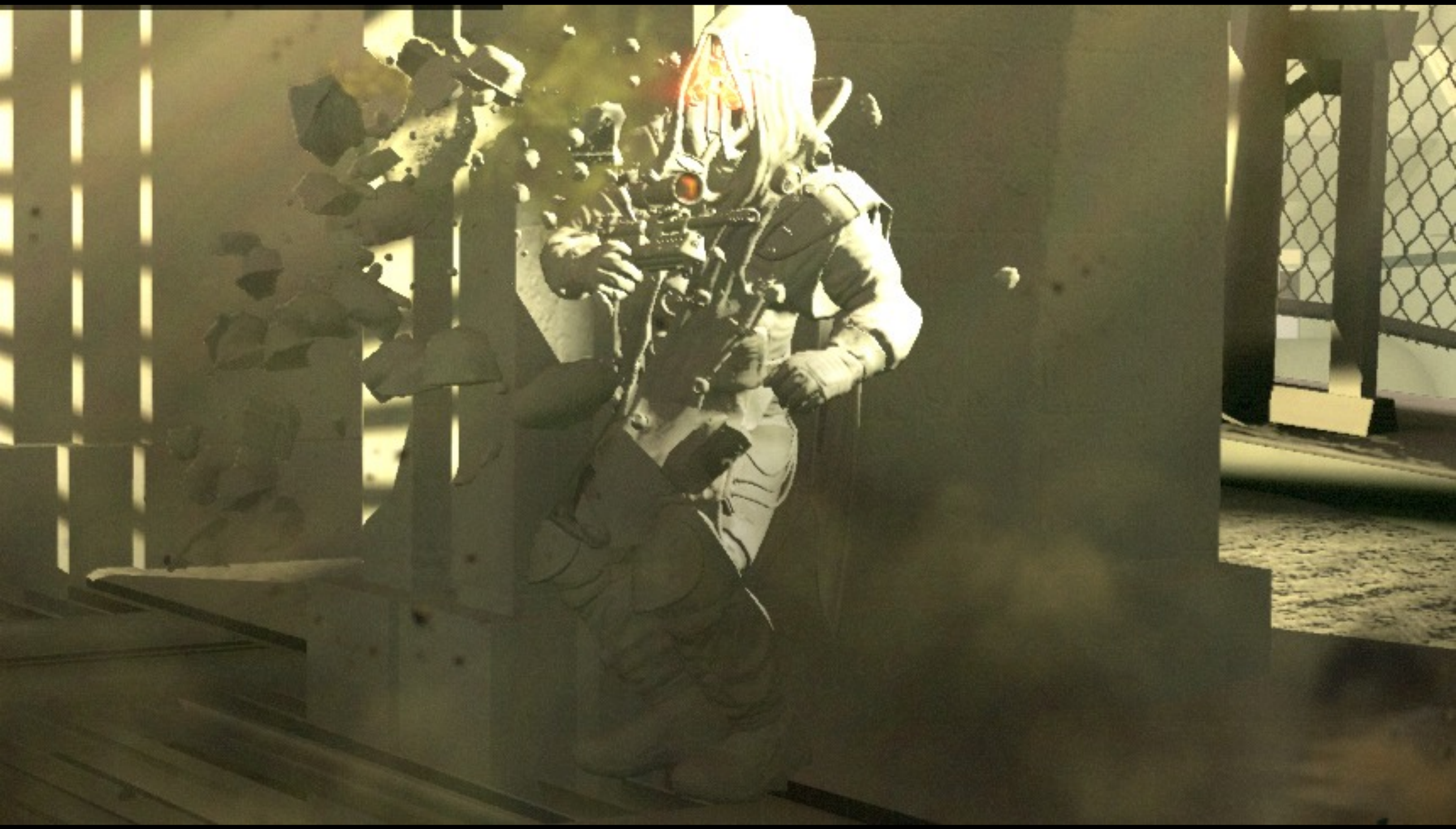
IBL Placement

IBL Samples around Sev (a dynamically lit object)

IBL Samples Texture maps

Intepolated IBL at Sev's position

Many dynamic objects, white test IBL, no textures



...add IBL sampling and sunlight

...add textures

PARTICLE SIMULATION

- We're quite particle heavy, per 30 Hz frame:
  - ~ 250 particle systems simulated
  - ~ 3000 particles updated
  - ~ 150 systems drawn
  - ~ 200 collision ray casts (w/ Havok)

- Difficult to optimize for multi-core
  - System had grown feature-heavy over time
  - Had to refactor in-place, incremental steps
  - Code was quite optimized, but generic C++
  - Memory accesses all over the place

PARTICLE SIMULATION

- System was refactored for SPUs in three steps
  - Vertex generation (+1 month)
  - Particle simulation inner loop (+2 months)
  - Initialization and deletion of particles (+3 months)
  - High-level management / glue (+4 months)

- Everything now done on SPUs except
  - Updating global scene graph
  - Starting & stopping sounds

- We learned a lot from porting the high-level code!

CODE STYLE DIFFERENCE

PPU Version

- Single-threaded
- Malloc & free
- Pointers to objects
- Input control curves
- Raycasts during update

- ~ 20ms update on PPU

SPU Version

- Heavily parallel
- Linear memory block
- Embedded objects
- Sampled lookup tables
- Queues raycast jobs

- < 1ms update on PPU
- ~ 15ms update on SPUs

20x Faster on PPU!
Incredible amount of work

GFX POST-PROCESSING

- Effects done on SPU
  - Motion blur
  - Depth of field
  - Bloom

- SPUs assist the RSX with post-processing
  - RSX prepares low-res image buffers in XDR
  - Then triggers interrupt to start SPUs
  - SPUs perform image operations
  - RSX already starts on next frame
  - Results processed by RSX in next frame

- Improved version in PlayStation®Edge 1.1!

POST-PROCESSING GENERAL

- Comparison
  - @ 30 Hz

	RSX Time	SPU Time	Quality
RSX	20%	0%	Medium
RSX + 5 SPUs	14%	12%	High

- SPUs are compute-bound
  - Bandwidth not a problem
  - Code can be optimized further

- Our trade-of RSX vs. SPU time
  - SPUs take longer
  - But SPUs look better
  - And RSX was our bottleneck

Input quarter-res image in XDR

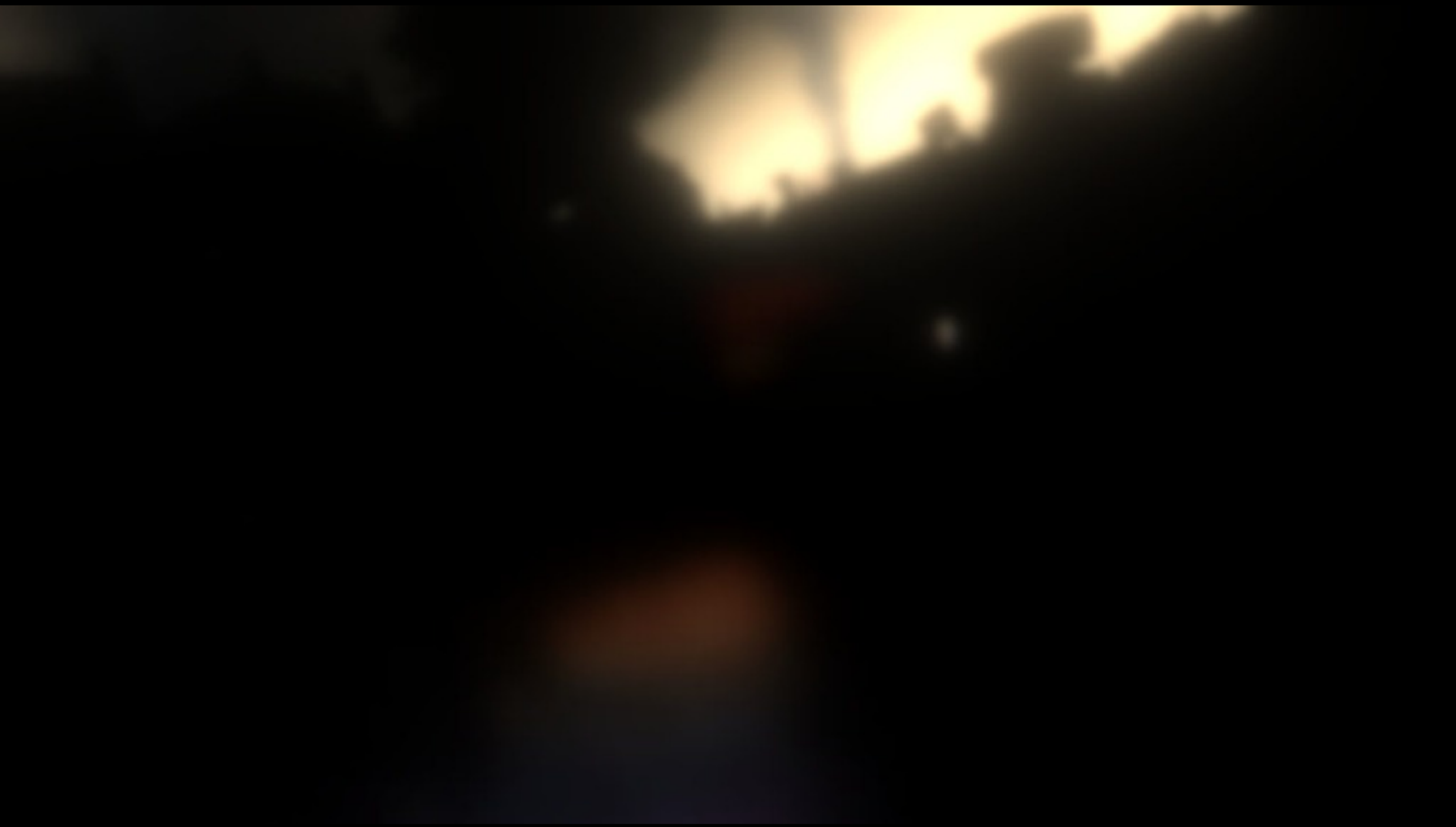
Image generated on the SPUs (bloom, DoF, motion blur)

Composited image

BLOOM + ILR

- Takes roughly 2.6% of five SPUs

- SPUs do
  - Depth-dependent intensity response curve
  - Hierarchical 7x7 gaussian blur (16 bit fixed point)
  - Upscaling results from different levels
  - Internal Lens Reflection (inspired by Masaki Kawase)
  - Accumulating into results buffer



Bloom + ILR Combined

MOTION BLUR

- Takes roughly 1.9% of five SPUs
- Input
  - Quarter-res image from deferred renderer
  - Sixteenth-res 2D motion vector stored as u8u8
- Steps:
  - Blur motion vectors (dilation)
  - Then blur image along motion vectors
    - SPU version does 8-tap point sampled
  - Combine blurred image with source
    - Use motion vector amplitude as alpha
- Low-motion areas are unaffected (alpha = 0)

MOTION BLUR

- Picture of scene
- Picture of motion vectors
- Image transition

GDC

Placeholder

DEPTH OF FIELD

- Takes roughly 4.6% of five SPUs
- Input
  - Quarter-res image from deferred renderer
  - Quarter-res depth buffer
- Convert depth buffer to 'fuzzy buffer'
  - 0=In, 1=Out of focus
- Samples image in floating point
  - 36 jittered disc point samples
  - Weighted by data from fuzziness buffer
  - Normalized by sum of fuzziness

DEPTH OF FIELD

- Picture of scene
- Picture of blurry bits
- Image transition

GDC

Placeholder

SPUs in

GAME CODE

ANIMATION SAMPLING

- Using Edge Animation (in PS3 SDK)
  - Our extensions for IK, look-at controller, etc.
- Time per frame
  - ✣50 animation jobs
  - ✣500 animations sampled
  - Less than 2,5% SPU time on five SPUs
  - Was ~20% PPU time with our old code!
- High-level animation logic still too heavy

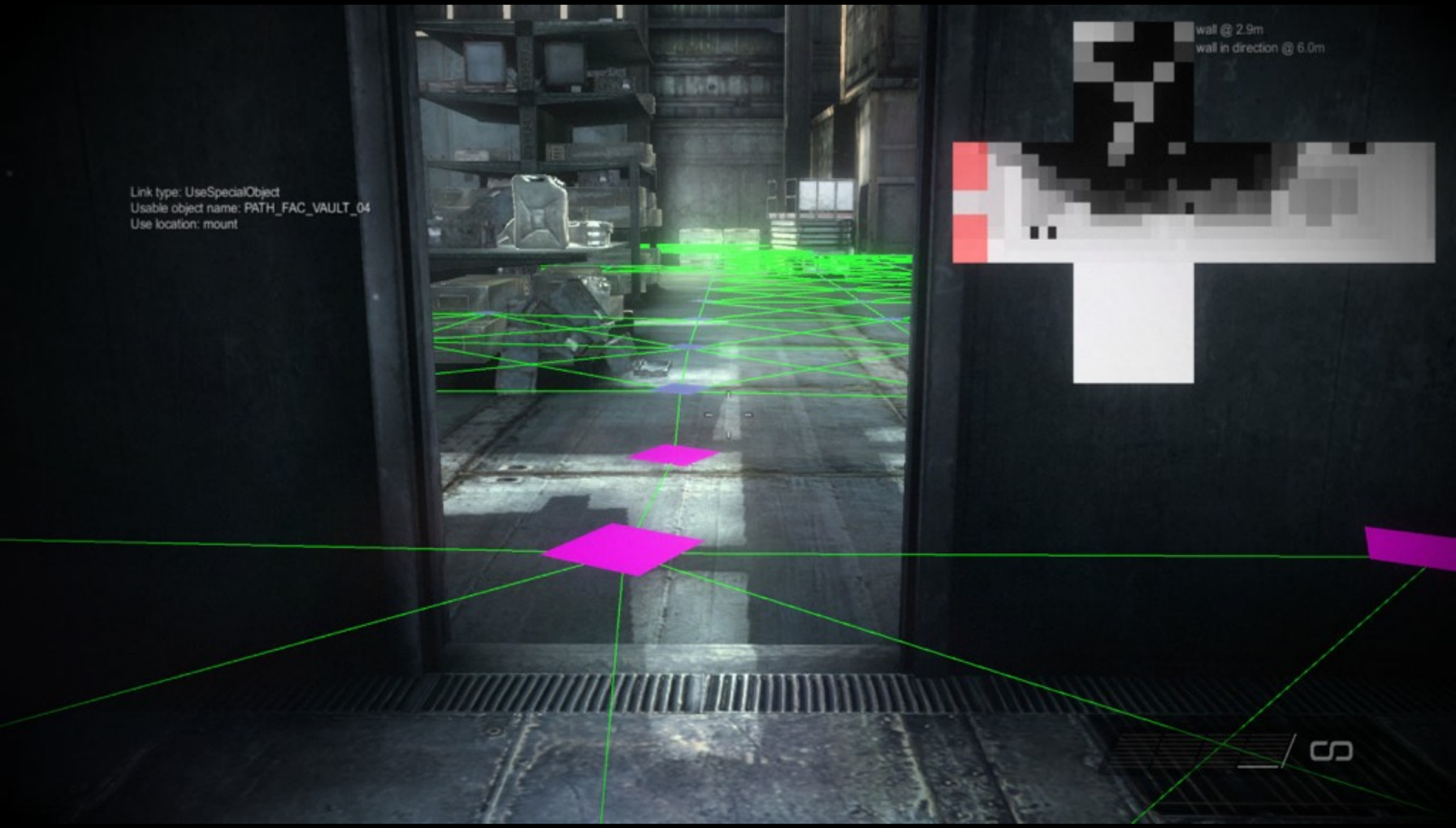
Edge Animation Is Fast!

SPUs in ARTIFICIAL INTELLIGENCE

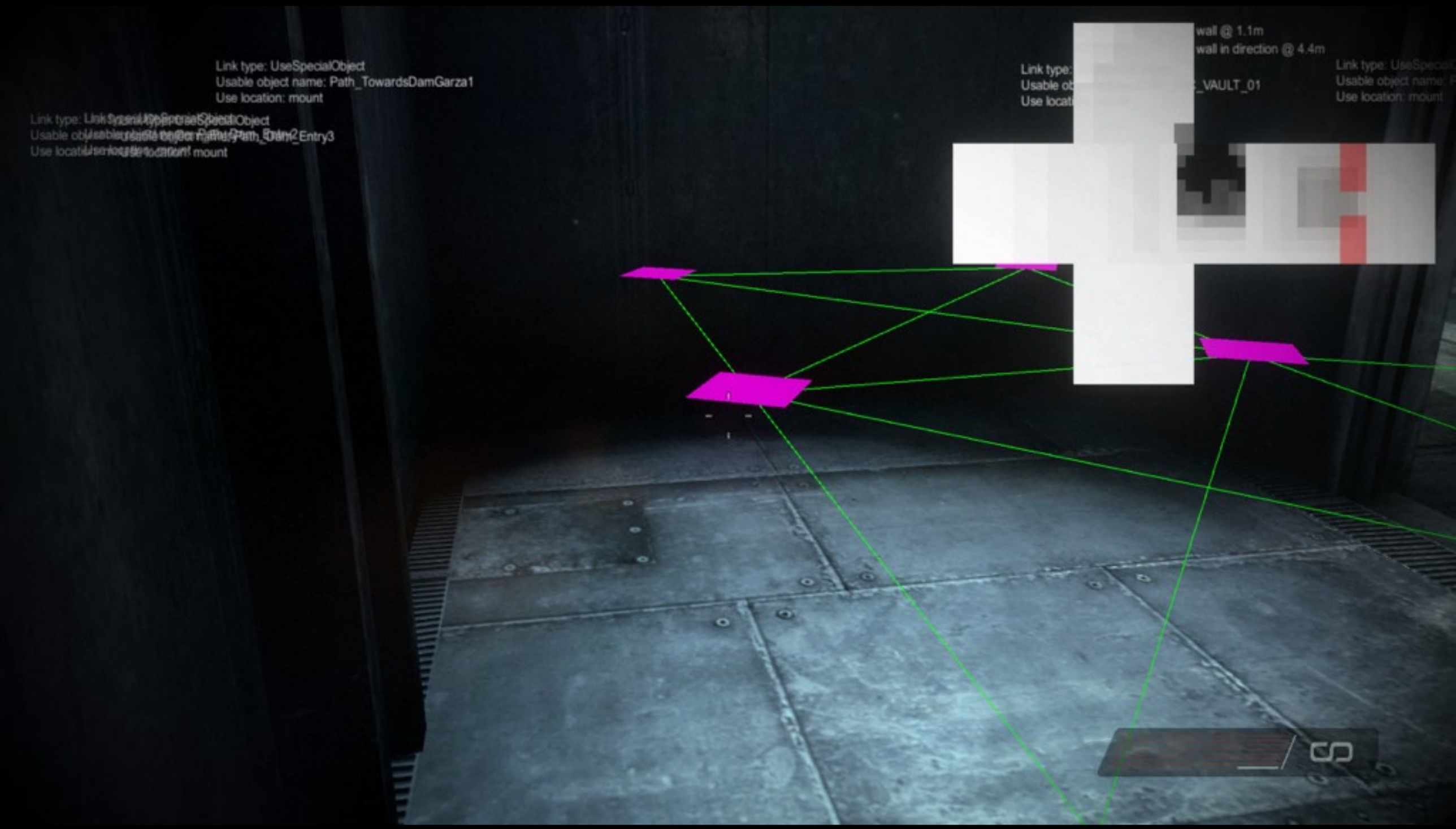
WAYPOINT COVER MAPS

- Killzone 2's cover maps are waypoint-based
  - Each waypoint has a depth cubemap
  - Allows line-of-fire checks between waypoints
  - This is how the AI understands the environment

- Suitable for SPUs
  - Small data size (compressed cubemaps)
  - Very compute-heavy
  - Can stream waypoint data easily



Waypoint cover map



Waypoint cover map

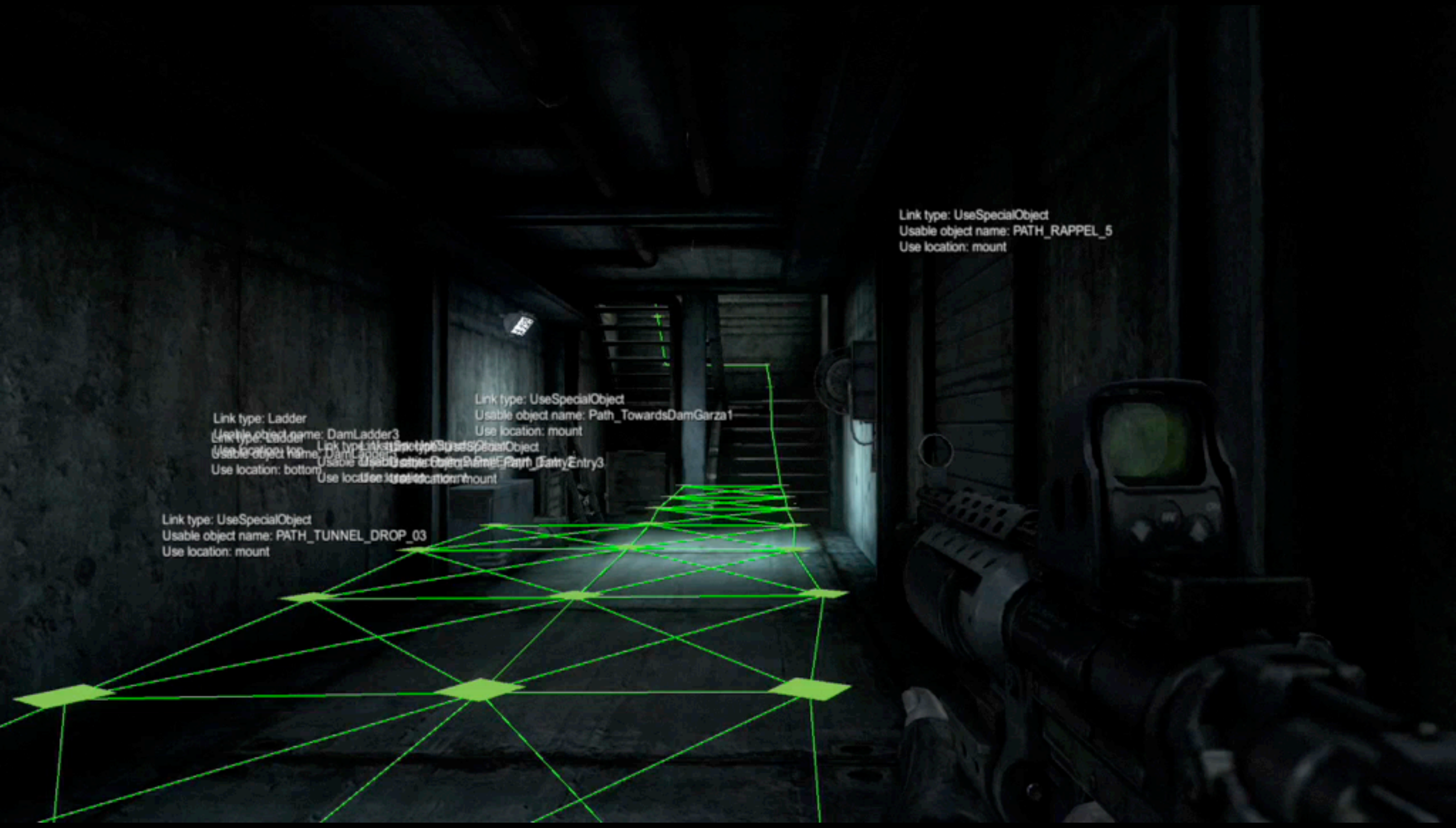
THREAT PREDICTION

- **Example:** You hide behind cover
- **Result:** AI searches for you, suppressive fire…
  - How?

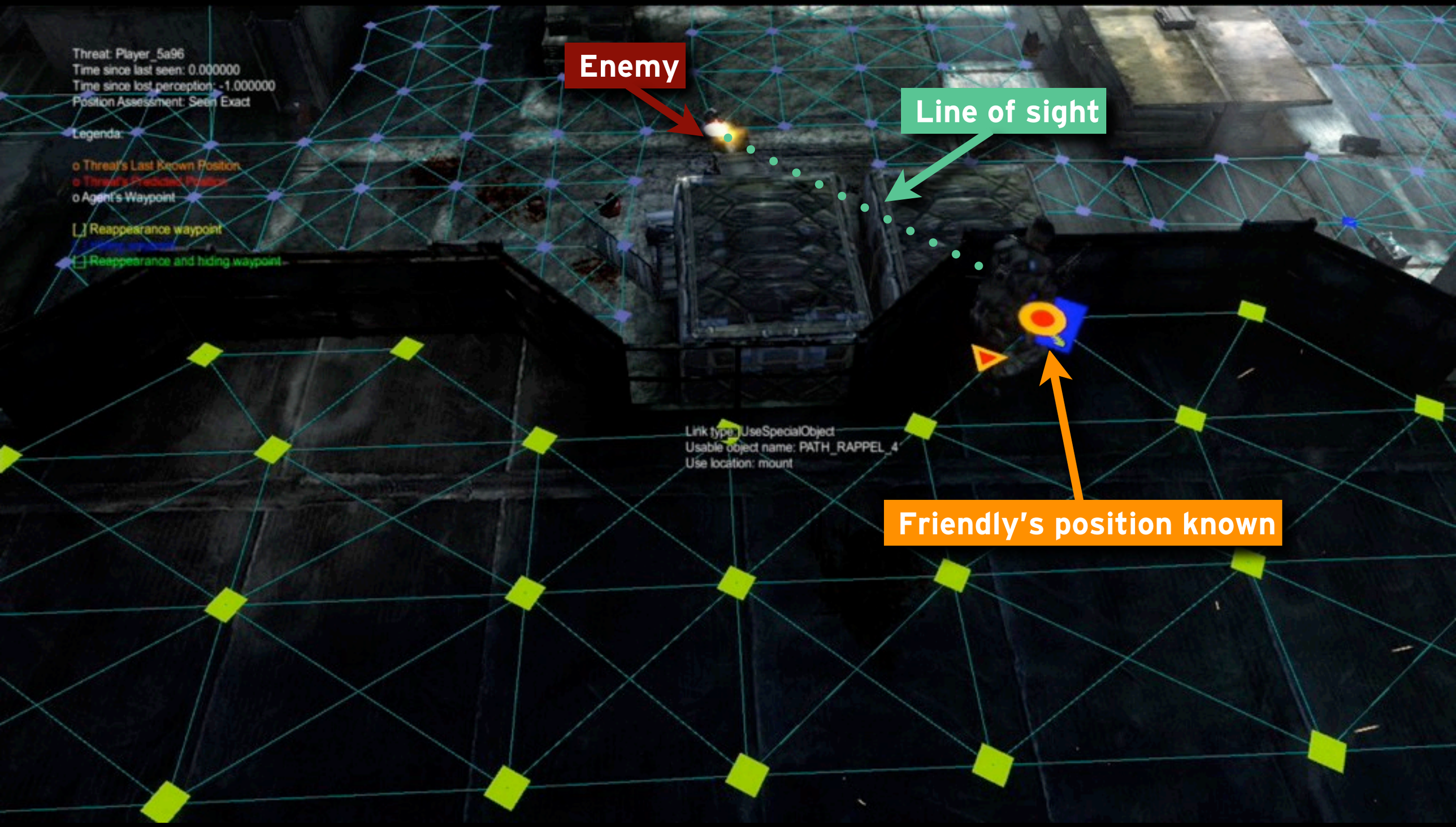
(Killzone AI doesn't cheat! - that much)

- AI remember time and waypoint of last contact
  - Mark waypoints where threat could move to
  - If waypoints are visible then remove from list
    - i.e. you can see waypoints and threat's not there!
  - If waypoint list grows too long, then stop expanding

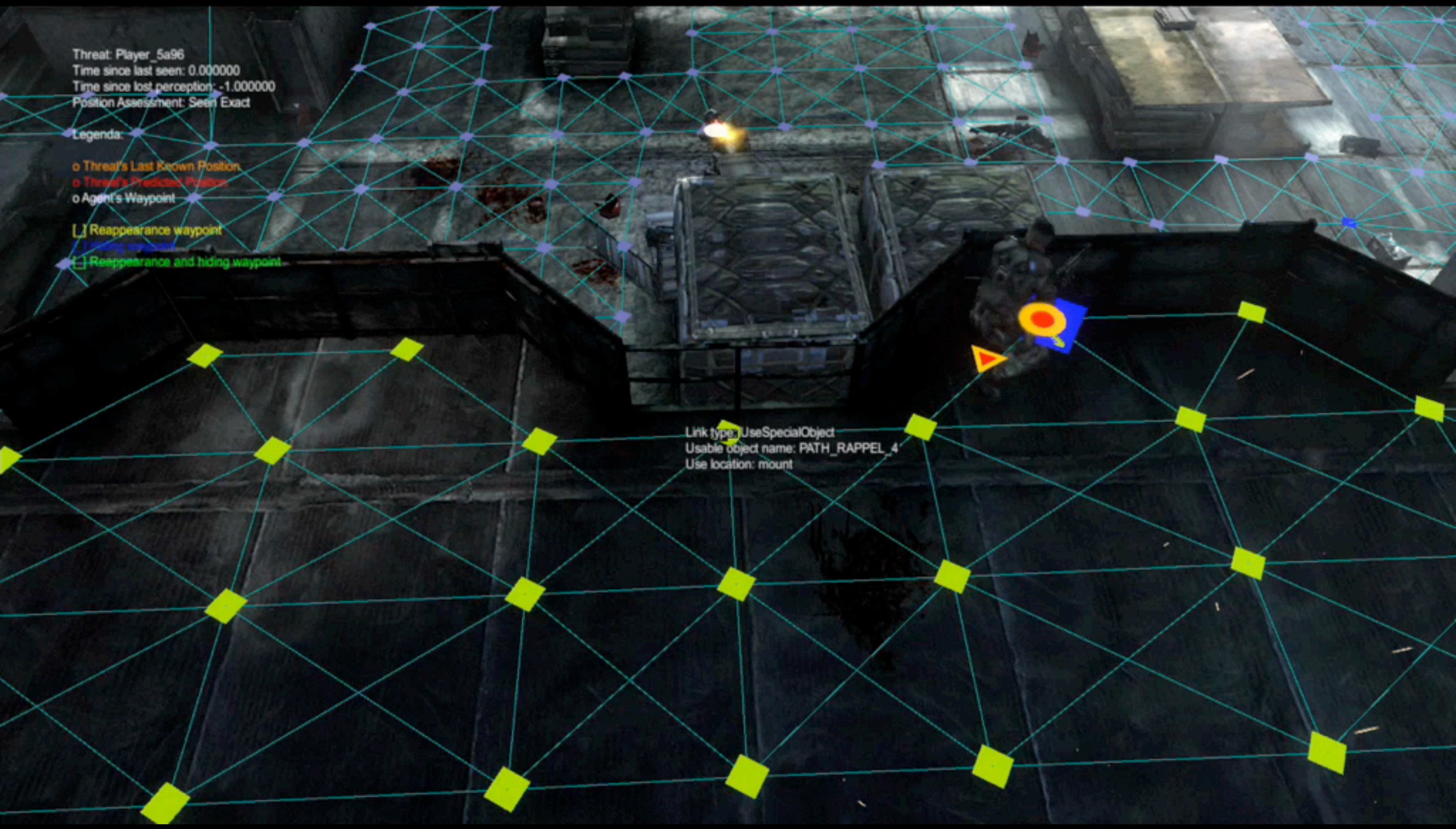
- If threat's predicted position is a small set then
  - Based on how AI's brain is configured…
  - …attack position or investigate possible location



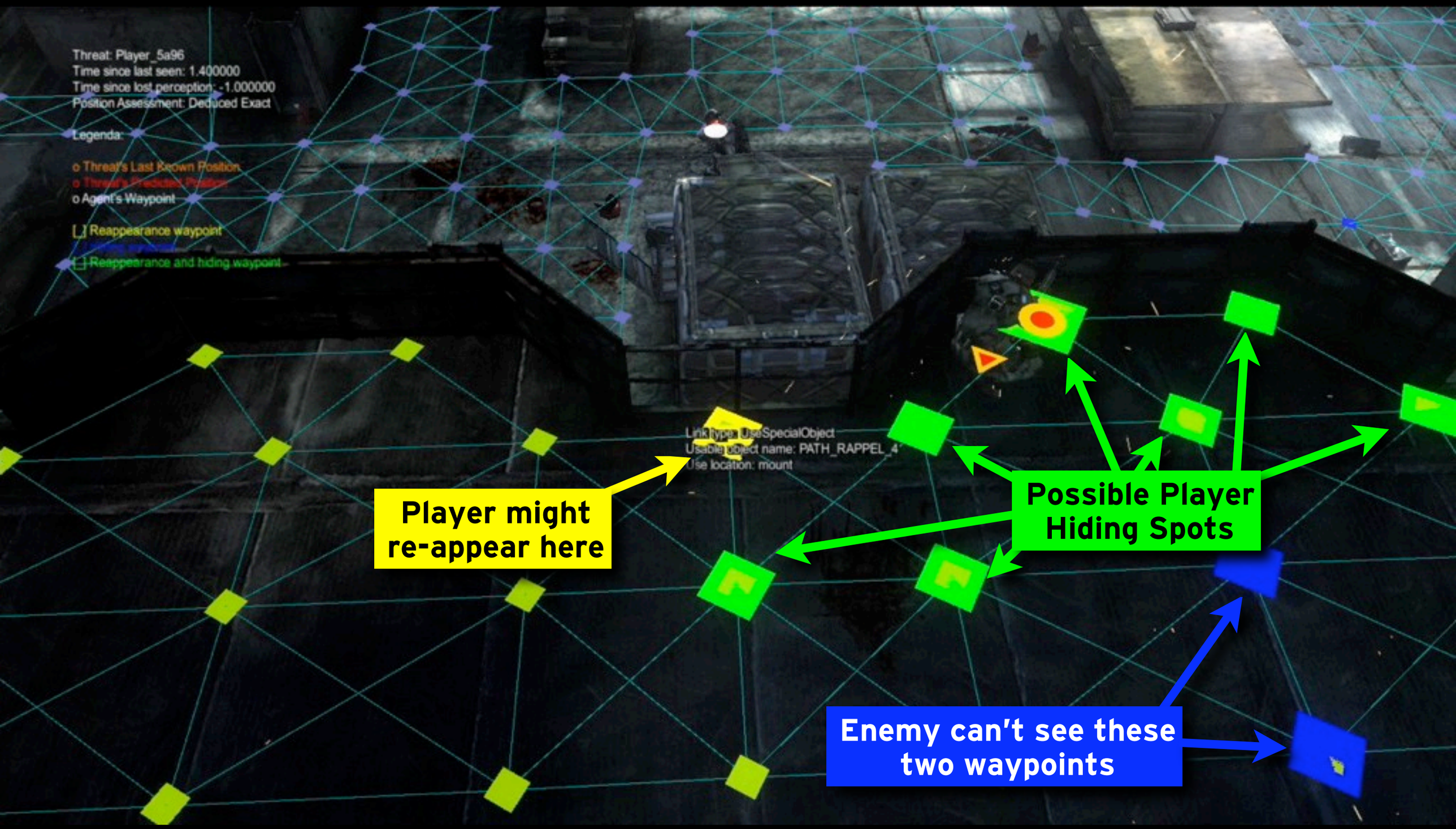
Threat Prediction using SPUs and cover maps



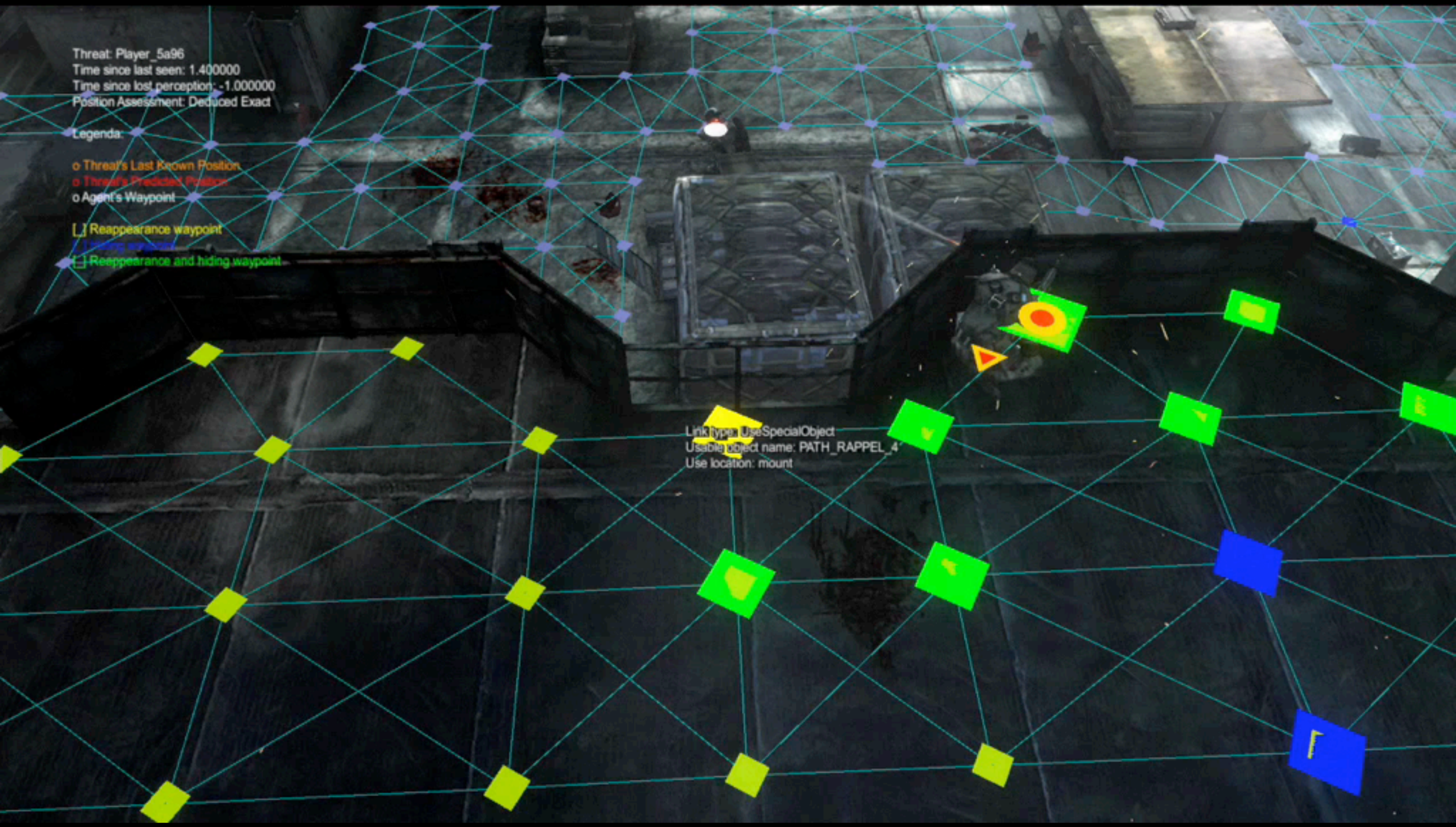
Encounter between enemy and friendly



Player enters cover position



Player enters cover position



Enemy looking for player

LINE OF FIRE

- **Problem:** AI running into each other's line-of-fire
- **Solution:** Line of fire annotations

- Each AI agent publishes 'hints'
  - Calculate which waypoints may be in my line of fire
  - Published as 'advice' for other entities
  - Most AI tasks use this advice in path planning

- SPU does lots many tests
  - Each line-of-fire against each waypoint link
  - Both LoF and links are tapered-capsules

Two friendlies engaging a foe

Capsule indicate Line of Fire

Safety first: Avoid standing on red links to not die

PUTTING IT ALL TOGETHER

SCHEDULING

- Almost all SPU code we have are Jobs
  - Many different job managers (middleware)
  - Managed by own high-level job manager
    - Manages the other job managers / workloads
    - Not much more than a container for all job queues
    - Easy to write your own, or grab somebody else's

- High-level scheduling hardcoded in main loop
  - Use barriers, mutexes and semaphores sensibly
  - Could use generalization, but good enough for now

- None of this is rocket science, just work

003. KZ

002. KZ

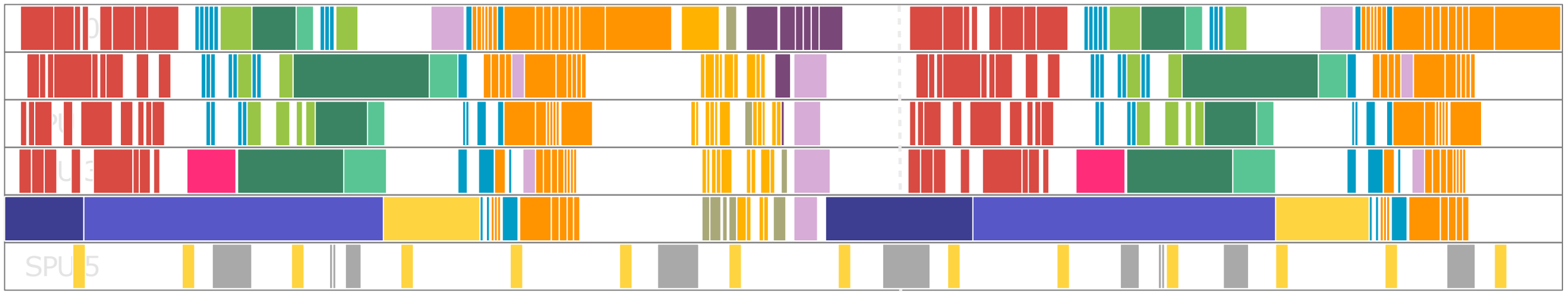
A FRAME IN THE LIFE OF...

PLEASE MAKE A SELECTION 001 KZ

PPU THREAD

PHYSICS	AI	GAME CODE	PRE-DRAW	KICK	PHYSICS	AI	GAME CODE

SPU JOBS



RSX PUSHBUFFER

LEGEND

- Audio
- System, unzip, etc.
- Havok Physics
- AI Jobs
- Animation
- Ray casts
- Particle update
- Particle vertices
- Skinning
- Main scene graph
- Main display list
- IBL Sampling
- Forward display list
- Lighting display list
- Shadowmap scene graph
- Shadowmap display list
- Post-processing

LESSONS LEARNED

WHAT WE DID (WRONG)

- We started off re-writing a lot of stuff for SPUs
  - Special SPU versions of code
  - Minimalistic, mirrored equivalents of old structures
- Huge amount of code and data duplication
  - Difficult to develop, maintain and debug
  - Massive waste of time! Reverted it all!

- Learned how bloated our data structures were
  - And how lean they could be

WHAT WE DID (RIGHT)

- All header files cross compile for PPU and SPU
  - Many data structures simplified and ‘lowered’
  - Low-level code is cross-platform inline in headers
- Code and data structures shared
  - DMA high-level engine classes to SPU and back
  - Use C++ templates in SPU code

- We try to treat them as generic CPUs and spend our time making generic optimizations, debugging tools, etc.

003. KZ

002. KZ

RECOMMENDATIONS

PLEASE MAKE A SELECTION 001 KZ

INVEST IN THE FUTURE

- The future is memory-local and excessively parallel
- SPUs are just one of these 'new architectures'
- Optimize for the concept, not the implementation
- Keep code portable, maintain only one version
- Keep time in schedule for parallelization of code

TREAT CPU POWER AS A CLUSTER

- Many separate equal resources
- Think in workloads / jobs
- Build latency into algorithms
- Favor computation-intensive code
- Avoid random memory accesses, globals

DON'T OPTIMIZE TOO MUCH

- Blocking DMA equals L2 cache miss penalty
- Most important is that your algorithms **scale**
- Optimizations make your code unmaintainable
- You can always low-level optimize **later**

AVOID RE-WRITING SYSTEMS

- Refactoring is often good enough
- Refactor in incremental steps, in-place
- Don't try to port your spaghetti code
- Overshoot and you might fail

RECOMMENDATIONS

1. Invest in the Future
2. Treat Your CPU Power as a Cluster
3. Don’t Optimize Too Much
4. Avoid Re-Writing Complete Systems

Thank You for Listening
Any Questions?