Sound Propagation In Hitman

Stepan Boev
Audio Programmer, Io-Interactive

GDC 'Eu

GAME DEVELOPERS CONFERENCE™ EUROPE
CONGRESS-CENTRUM OST KOELNMESSE · COLOGNE, GERMANY
AUGUST 3-4, 2015

About me

- Audio programmer
- 10 years in the games industry
- Projects:


Agenda

- About Hitman
- Motivation
- Solution
- Implementation
- Challenges
- Conclusion

Agenda

- **About Hitman**
- Motivation
- Solution
- Implementation
- Challenges
- Conclusion

[Hitman gameplay trailer]

Game facts

(that affect sound)

- Stealth-oriented - "hide in plain sight"
  - Exploring level in disguise and listening to NPC dialogue
- Large sandbox levels
  - Both indoor and outdoor environments
- Mostly static level geometry
  - With basic dynamic elements, such as:
    - Opening doors, shooting out windows
    - Possibly shooting holes in the walls

Development facts

(that affect sound)

- Powered by Glacier 2 engine (proprietary)
  - Focus on visual programming (graphs)
- Sound system is based on Wwise
- Audio team
  - 6 sound designers
  - 1 programmer

Agenda

- About Hitman
- **Motivation**
- Solution
- Implementation
- Challenges
- Conclusion

Background

- Sound waves are:
  - Absorbed, diffracted and reflected when interacting with materials
- Simulating this can improve user experience
  - Physically-based simulation is computationally expensive
  - We are interested in an efficient approximation

Impact on user experience

- Less confusion
  - Smarter attenuation than falloff
    - Man talking two meters away from you should not be heard if there is a thick wall in between
- More immersion
  - Give player aural clues about environment
    - Somebody is talking in a room nearby
    - The voice gets clearer as you approach the door leading to the room
    - There is a radio playing behind that crate

How do we achieve this?

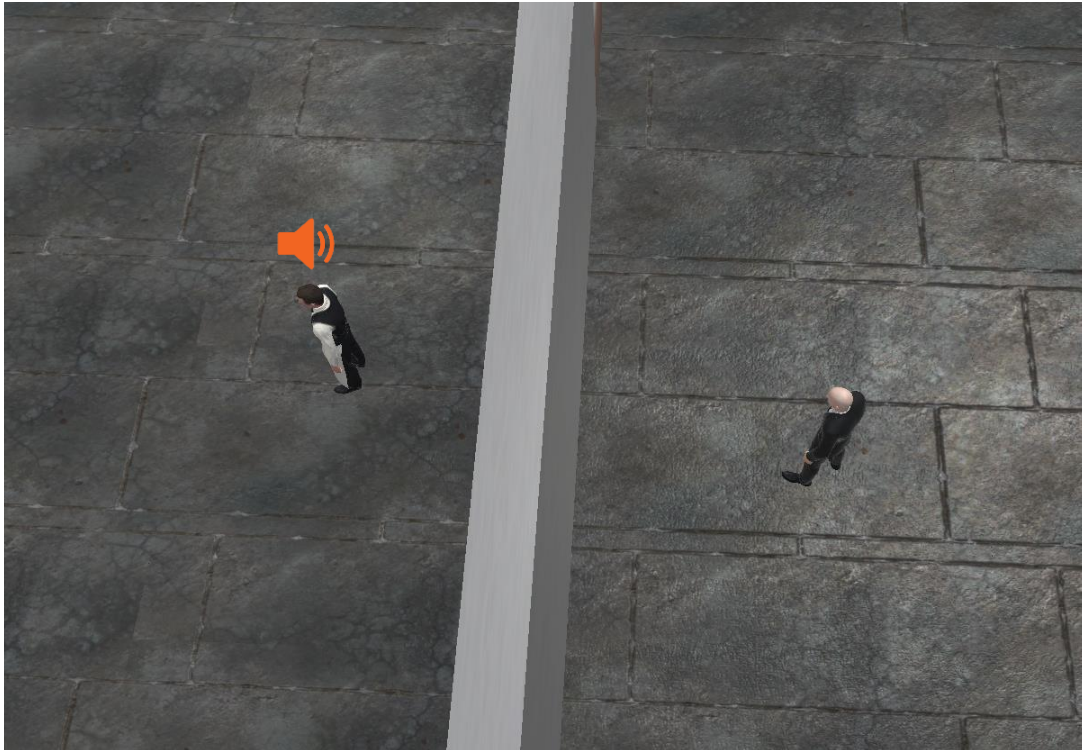
- Concrete goals
  1. Muffle sounds occluded by walls
  2. Muffle sounds obstructed by game objects
  3. Make sounds appear to emanate from openings

- Let’s look at them in more detail

Goal 1: Muffle sounds occluded by walls

- Sound doesn't know about level geometry

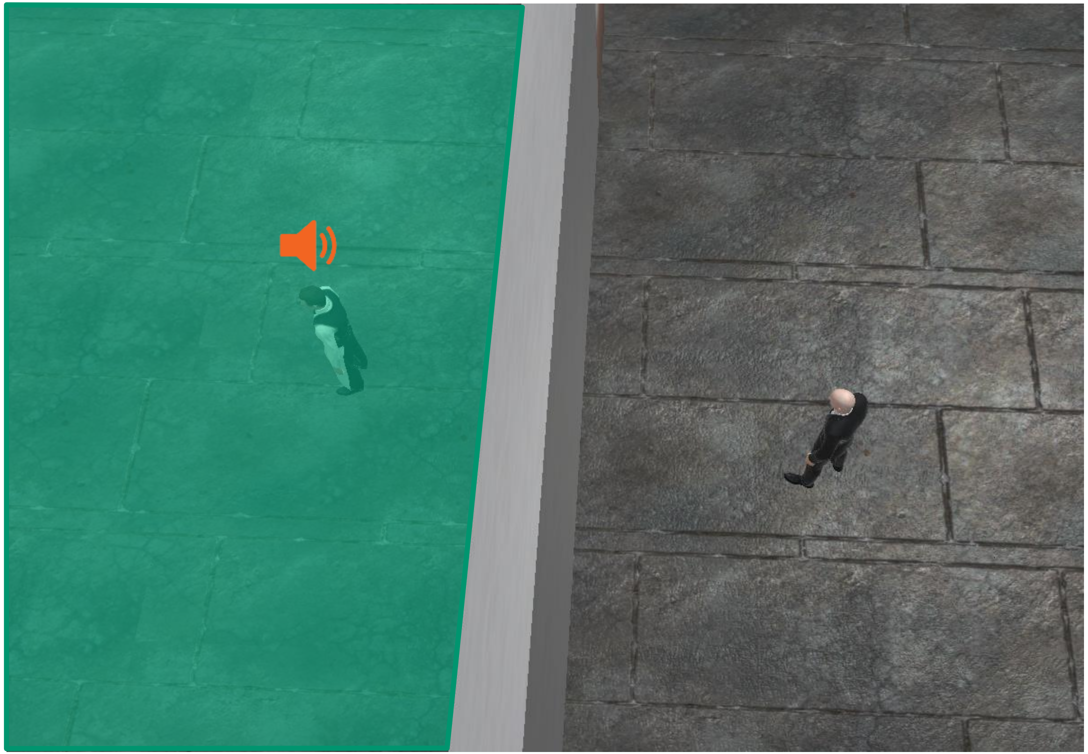
- Regular distance-based attenuation does not handle sound's path being blocked by e.g. walls and ceilings



Goal 1: Muffle sounds occluded by walls

- With wall occlusion, sound is muffled or muted by the wall
- Preventing player's confusion upon hearing 'ghost sound'
- Example: gunshot

Unoccluded / semi-occluded



Goal 2: Muffle sounds obstructed by objects

- When there are objects between sound and listener, sound should be slightly muffled

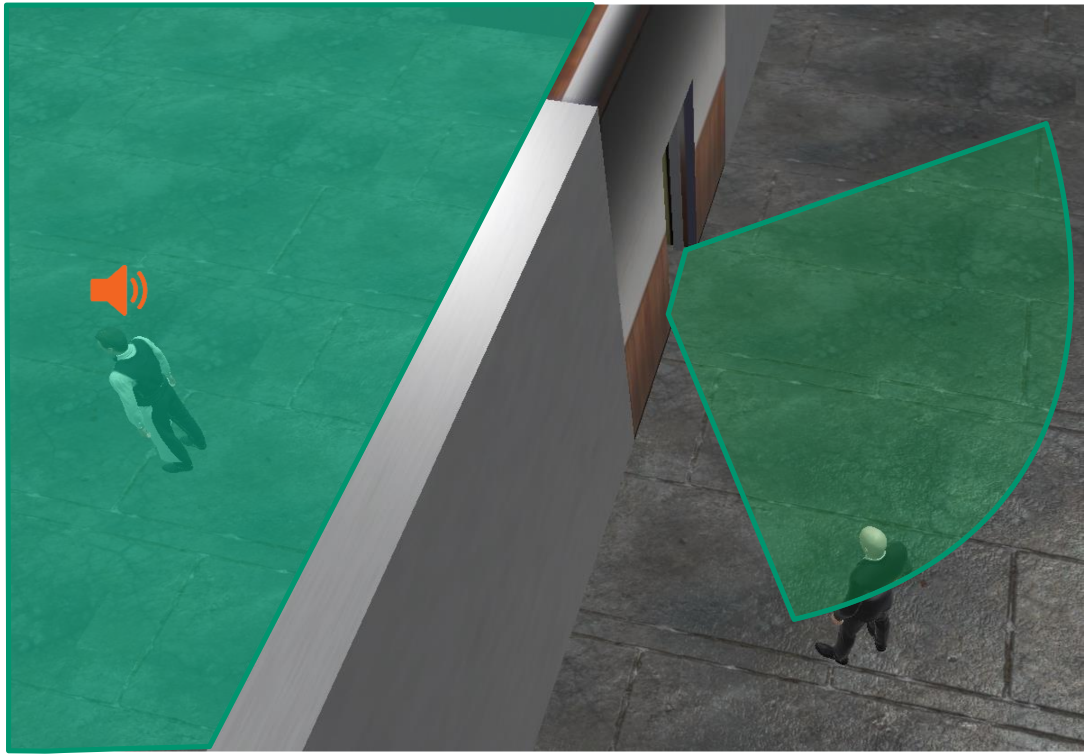
Goal 2: Muffle sounds obstructed by objects

- When line of sight is restored, sound should go back to normal
- Transition between muffled and normal should be smooth

Goal 3: Make sounds appear to emanate from openings

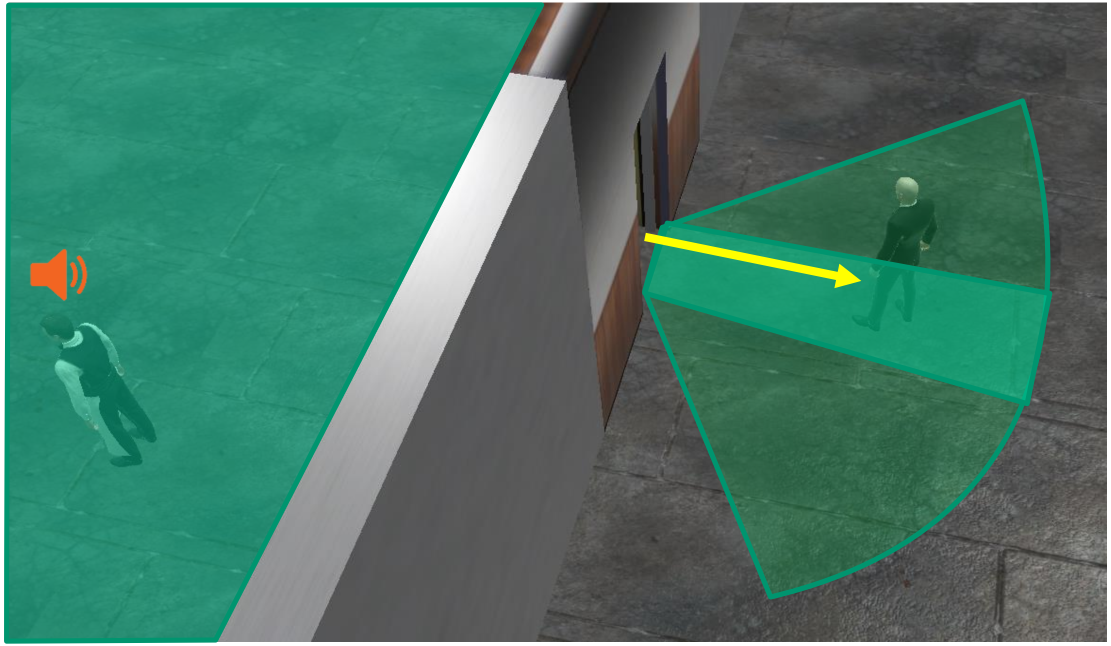
- Sound is diffracted around opening's edges

- It gradually gets louder as listener is approaching the opening…



Goal 3: Make sounds appear to emanate from openings

- …becoming completely unoccluded when listener is in front of the opening
- Sound is also perceived to be emanating from the direction of the opening



Propagation system requirements

- Immersive
- Consistent
    - No jumps in attenuation or other audible artefacts
- Computationally inexpensive
    - Total CPU budget for audio in Hitman:
        - 1ms on the main thread
        - Up to 50% of one core
- Support for dynamic geometry
- Reasonable implementation time

Agenda

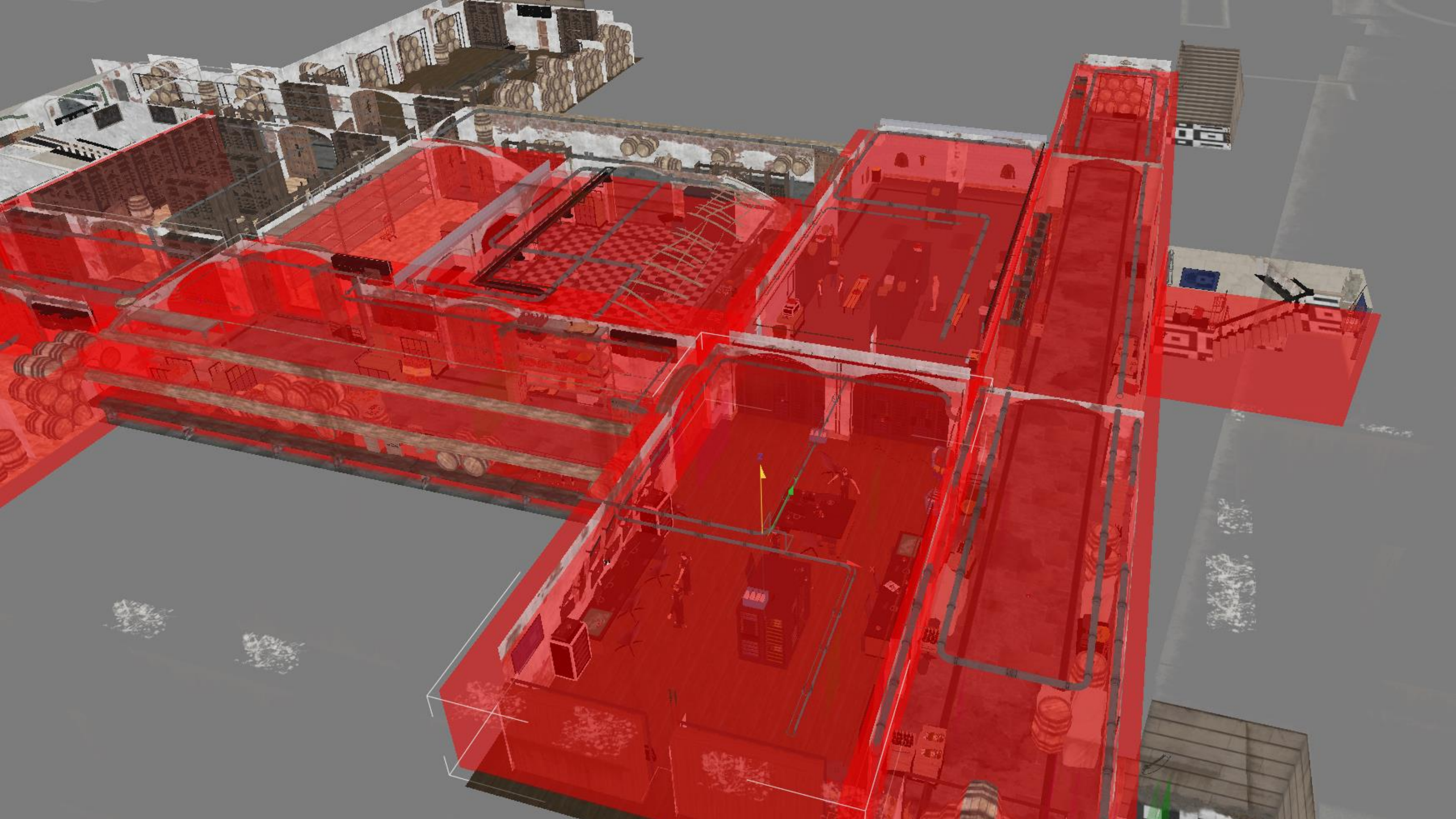
- About Hitman
- Motivation
- **Solution**
- Implementation
- Challenges
- Conclusion

Solution components

- Propagation geometry
- Occlusion
- Propagation paths
- Obstruction

Propagation geometry

- First, we need some meta data about the level
  - Rough replica of level geometry…
  - …that should allow to efficiently compute sound propagation paths
- Physics/graphical data usually can’t be used as is
  - We need special propagation geometry

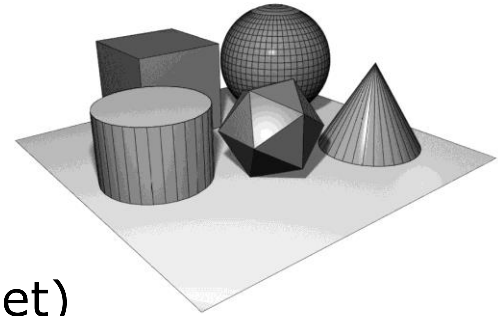


Propagation geometry

- Consists of “rooms” and “portals”
  - Rooms represent environments
    - Both indoor and outdoor
  - Portals represent openings
- Rooms keep track of sound sources in them
- Portals connect adjacent rooms
  - They are used for computing propagation paths

Propagation geometry

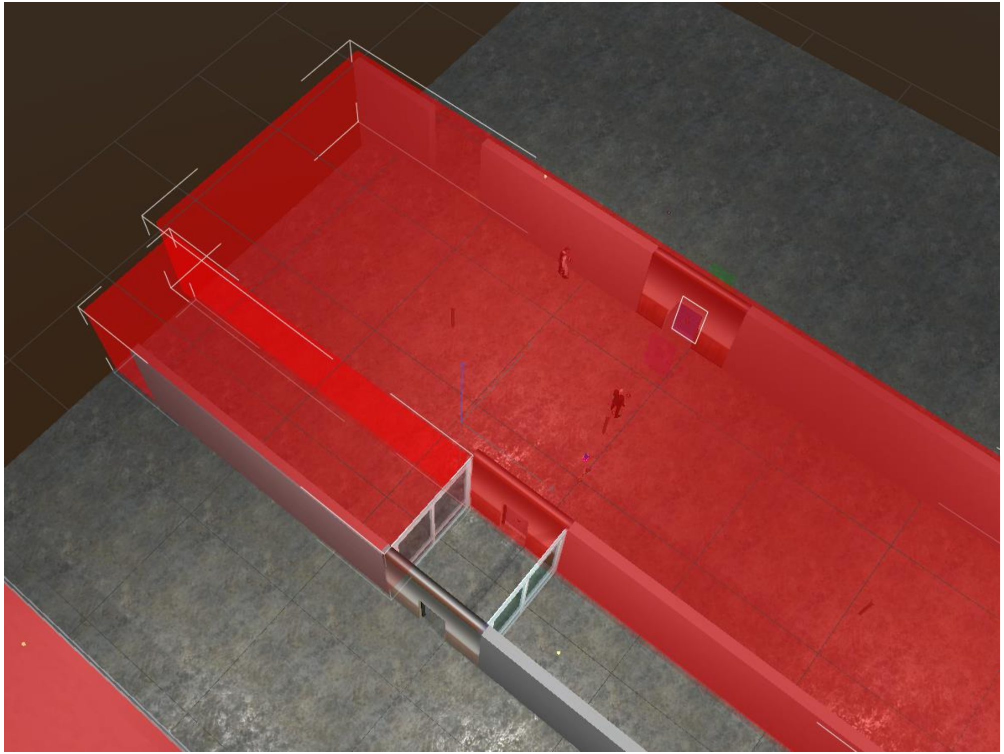
- Rooms can be arbitrary 3D primitives
  - Boxes, cylinders, polygonal shapes
- Portals are 2D primitives
  - They are 3D-positioned and rotated
  - Rectangles, circles, polygonal shapes



- (We didn't need polygonal shapes in our levels yet)

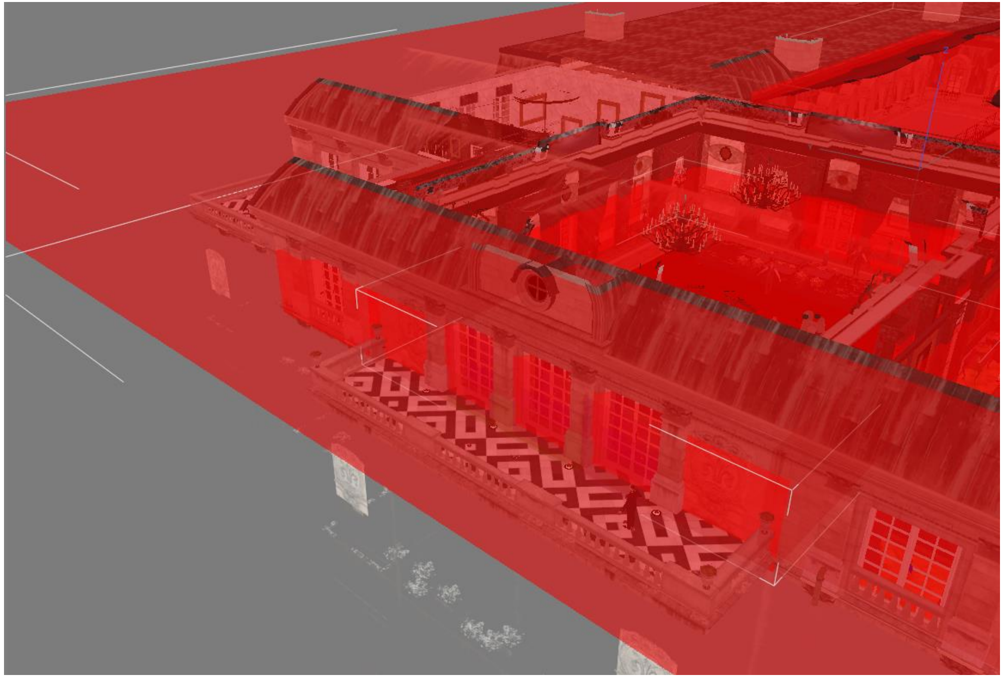
Grouping of rooms

- Primitives can be grouped, forming complex shapes
- In this example, red rooms are grouped and treated as one



Nesting of rooms

- Rooms can be nested
- For example, we can have one shape for the whole building and a shape for each room in that building

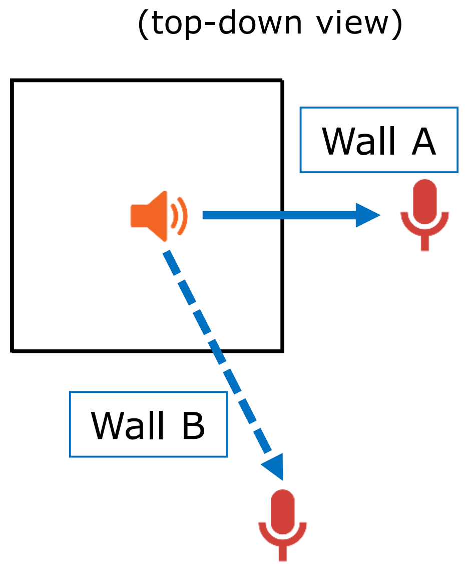


Geometry properties

- Every room has “occlusion value”
  - Defines how much sound is absorbed by it walls
  - e.g. different values for concrete and wooden rooms
- Every portal has occlusion value as well
- These values are dynamic and can be driven by gameplay

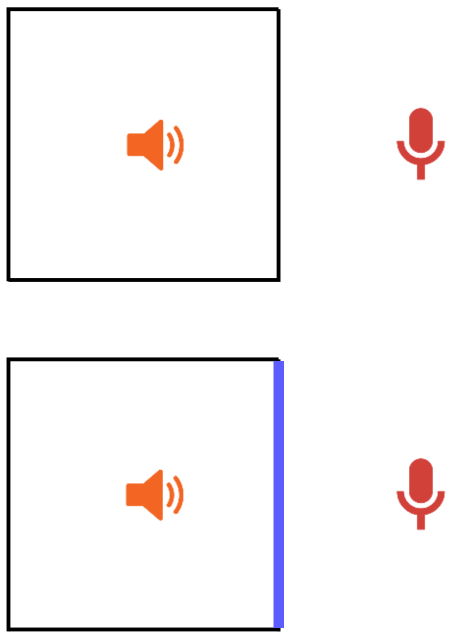
Room occlusion value

- Why single value for all walls?



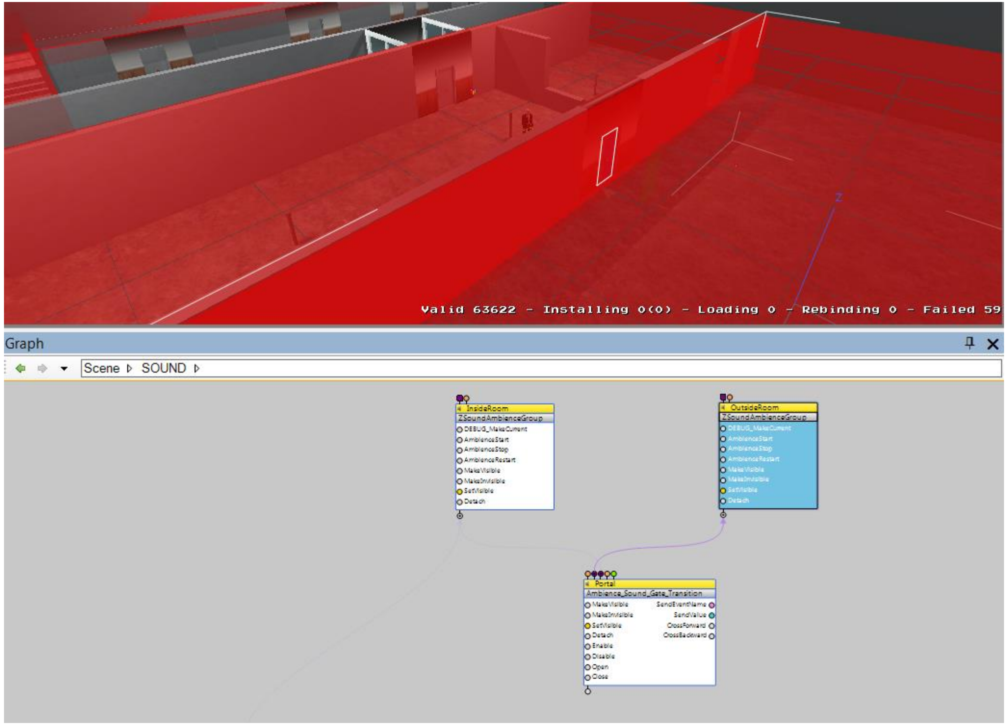
Room occlusion value

- Why single value for all walls?
- If a certain wall must have a different value, create a portal for it (blue line)
- This can be done at runtime, e.g. if a graphical wall is blown up
- Portals may overlap one another
  - e.g. a portal for wall and a portal for a window on that wall



Geometry setup

- Currently done manually in the editor
- Takes 1-2 days to setup a production level
  - Plus additional overhead to maintain
- But, we are looking into automated generation



Geometry stats

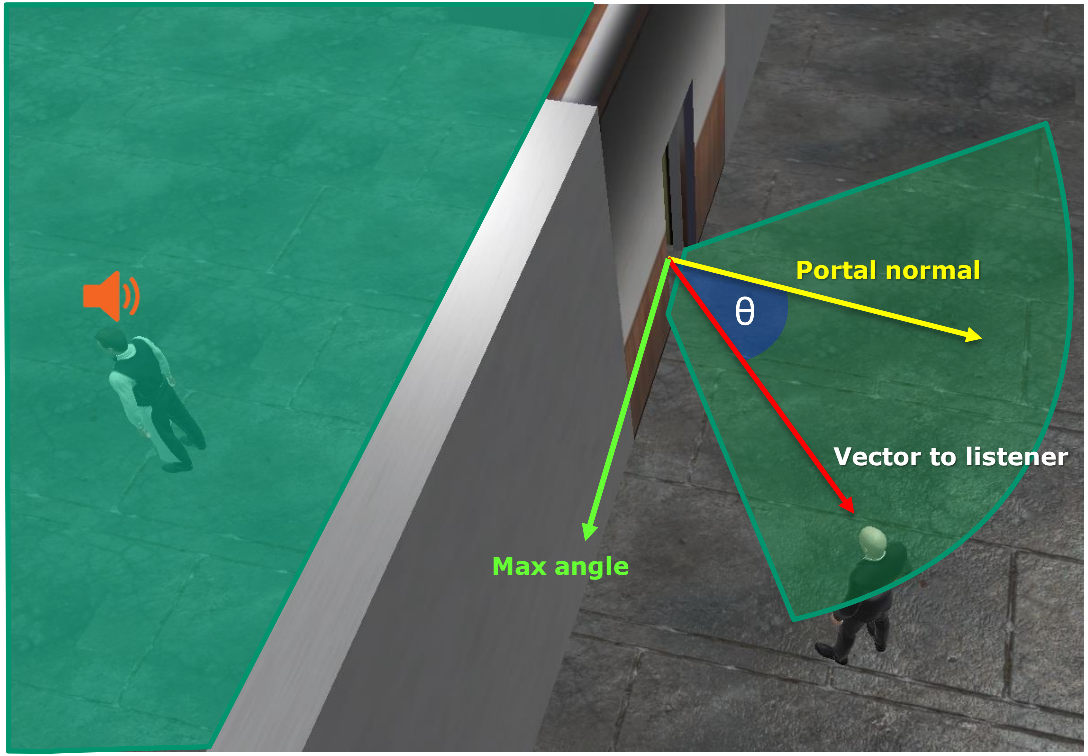
- Average per game level in Hitman:
  - 275 rooms
  - 800 portals
- These values can go much higher
  - …without significant impact on performance

Computing occlusion

- Find closest point to listener on the portal

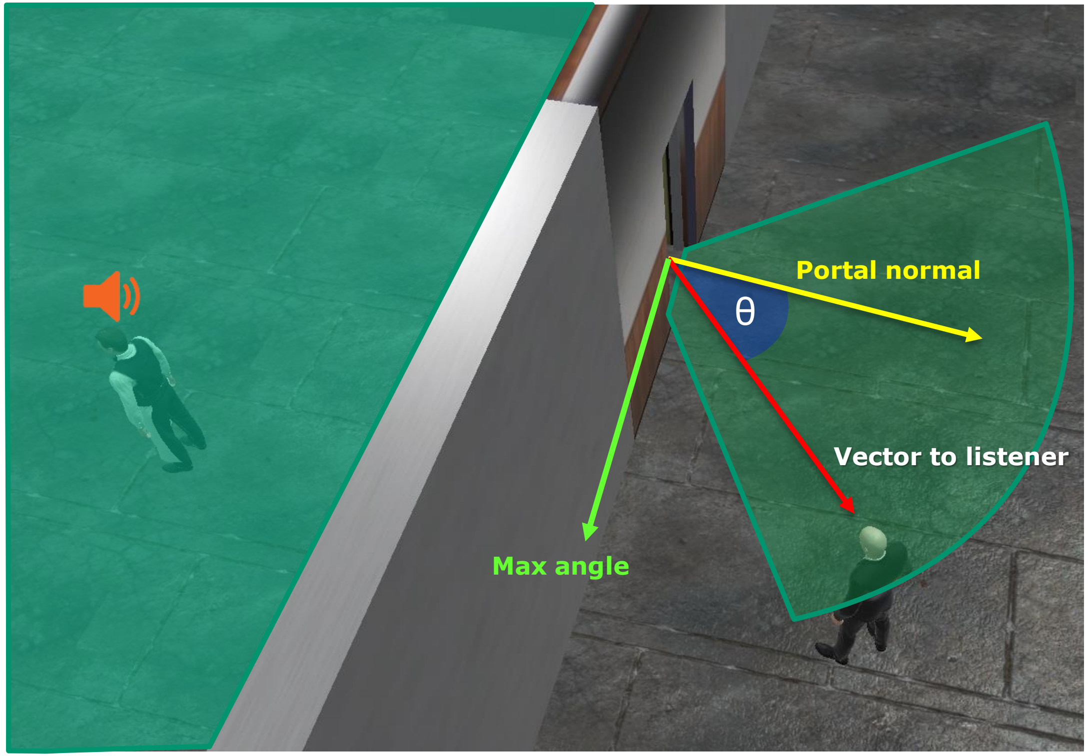
- Angle θ between portal normal and vector from that point toward listener defines interpolation factor…

- …between portal's and room's occlusion values



Computing occlusion

- This means that occlusion value will gradually transition from room to portal occlusion value as listener approaches a portal
- Maximum angle can be set individually for each portal
- Each portal also adds some occlusion based on distance to listener or previous portal

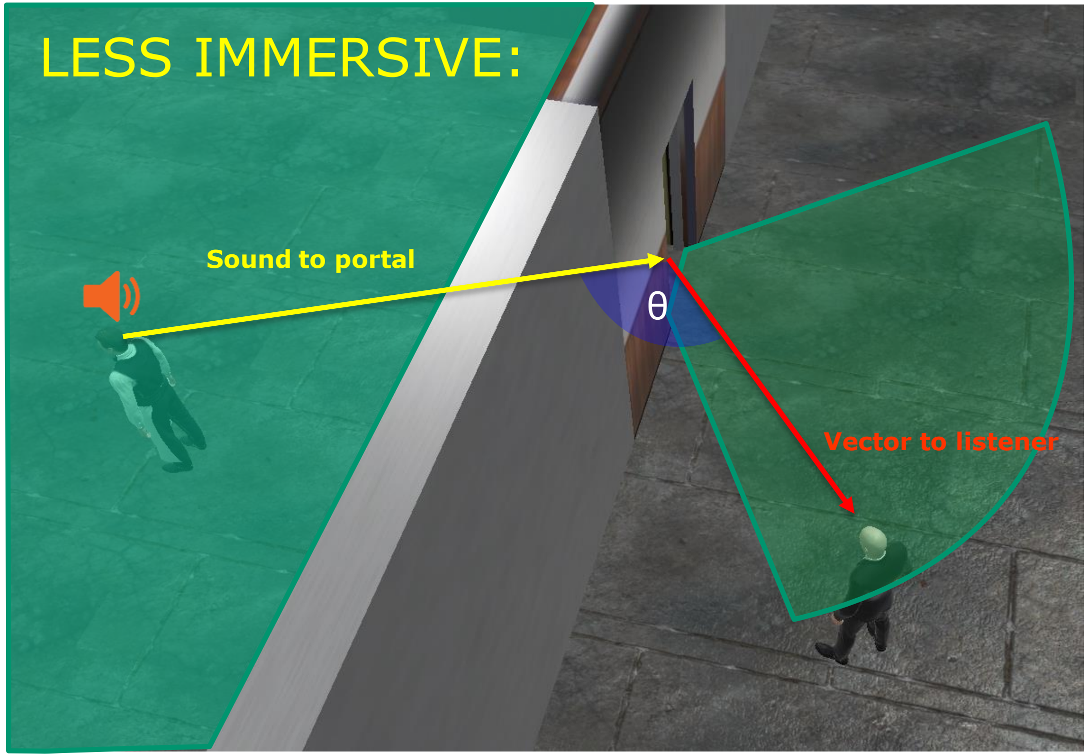


Computing occlusion

- We have also tried calculation based on the angle as shown

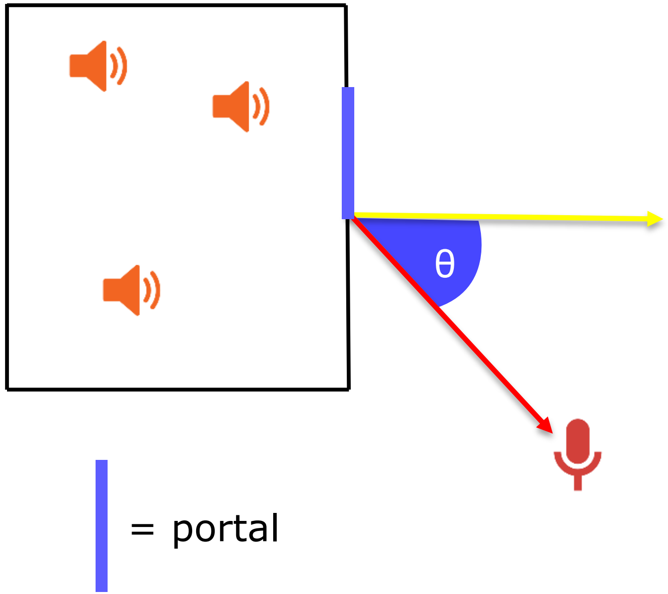
- This sounds less immersive

- That’s why we ignore how sound is positioned in relation to the portal when calculating occlusion



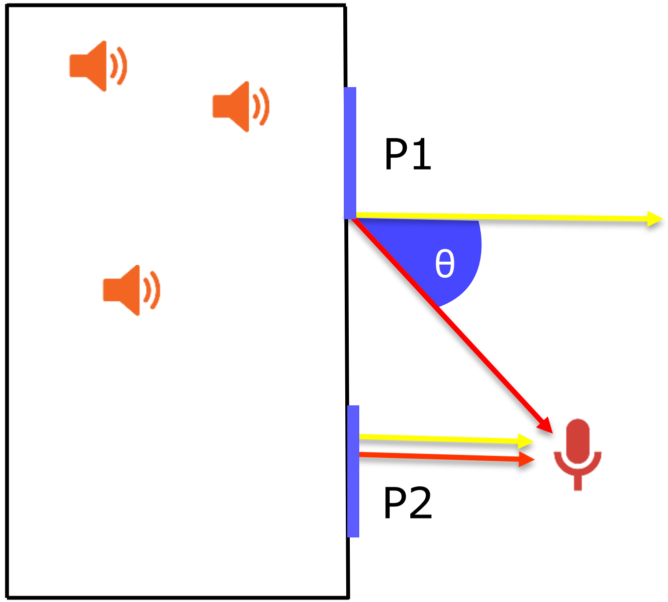
Computing occlusion

- This means we don’t have to calculate occlusion for individual sounds
- Computing occlusion for a portal gives us one value for all sounds in the opposite room
    - Also good for short-lived emitters
- This is faster and empirically sounds better



Computing occlusion

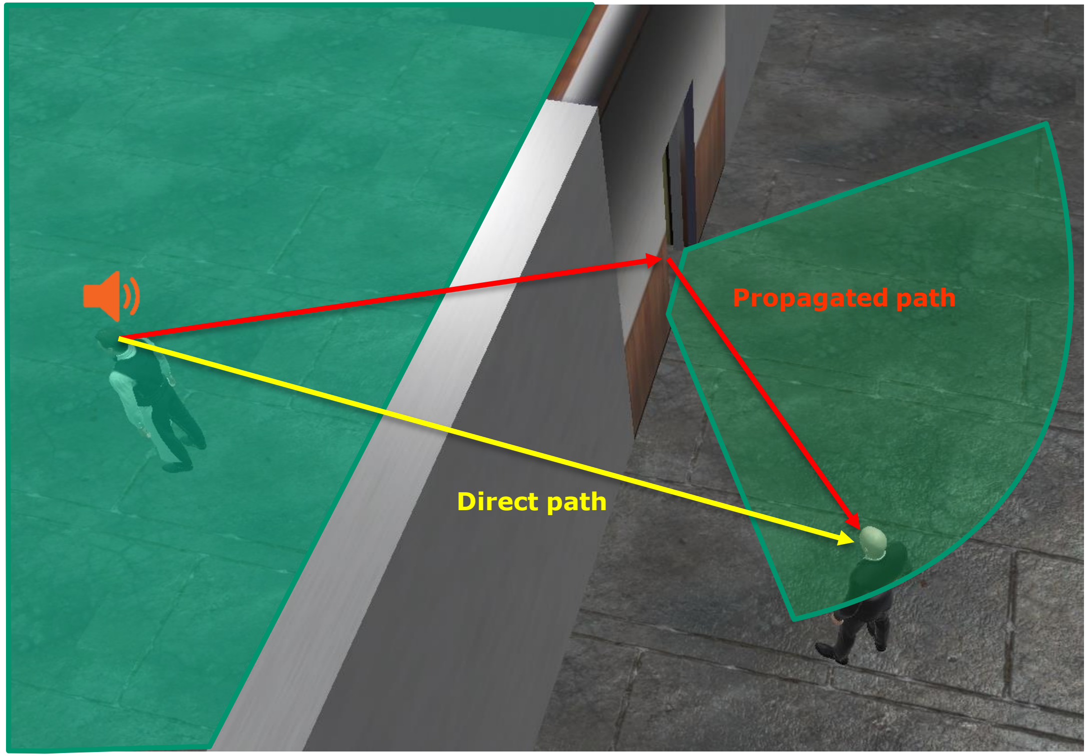
- What if there are several portals?
- We compute occlusion value for all and pick the one with the lower value
- In this example, P2 would be preferred, because the listener is right in front of it
- However, if P2 was a closed door, and P1 was an open window, we would probably prefer P1



[video demo]

Propagation increases falloff distance

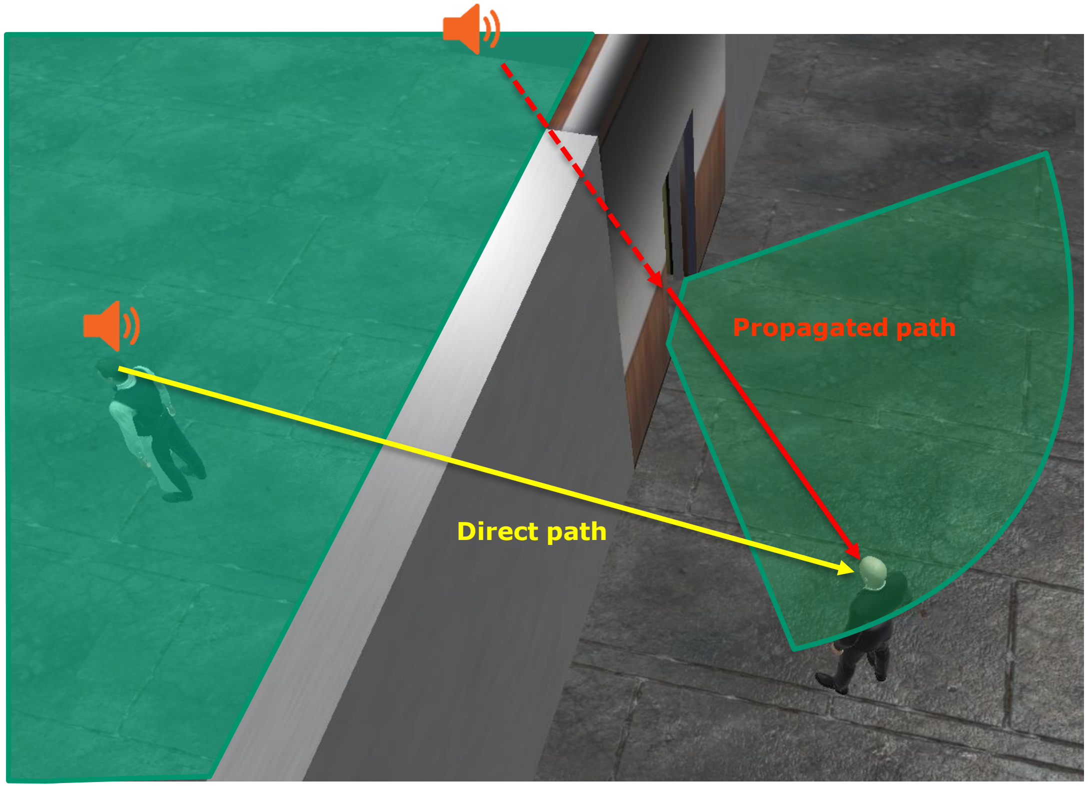
- Distance from sound to listener becomes longer with diffraction
- Falloff attenuation should be calculated using the propagated distance
- Reposition the emitter farther away along the yellow vector or scale the falloff distance



Perceived position

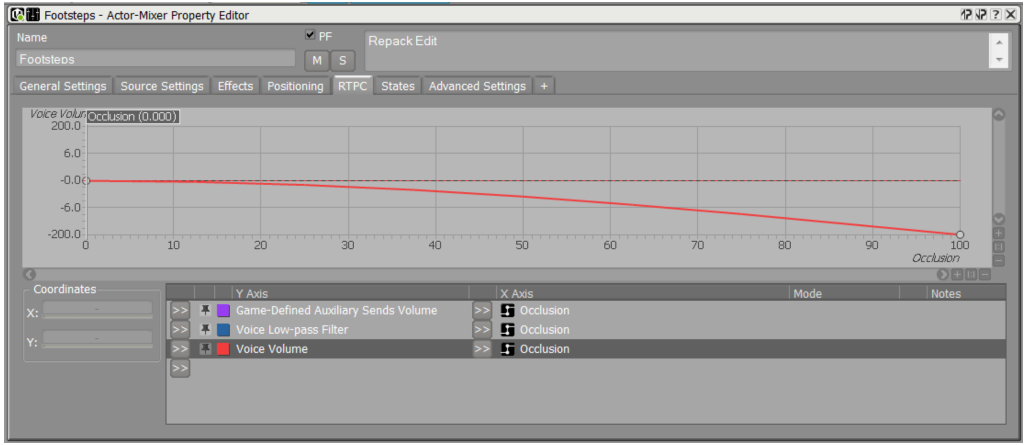
- You may also want to create a 'virtual' emitter along the red vector

- This will create perception of sound coming from the direction of the portal



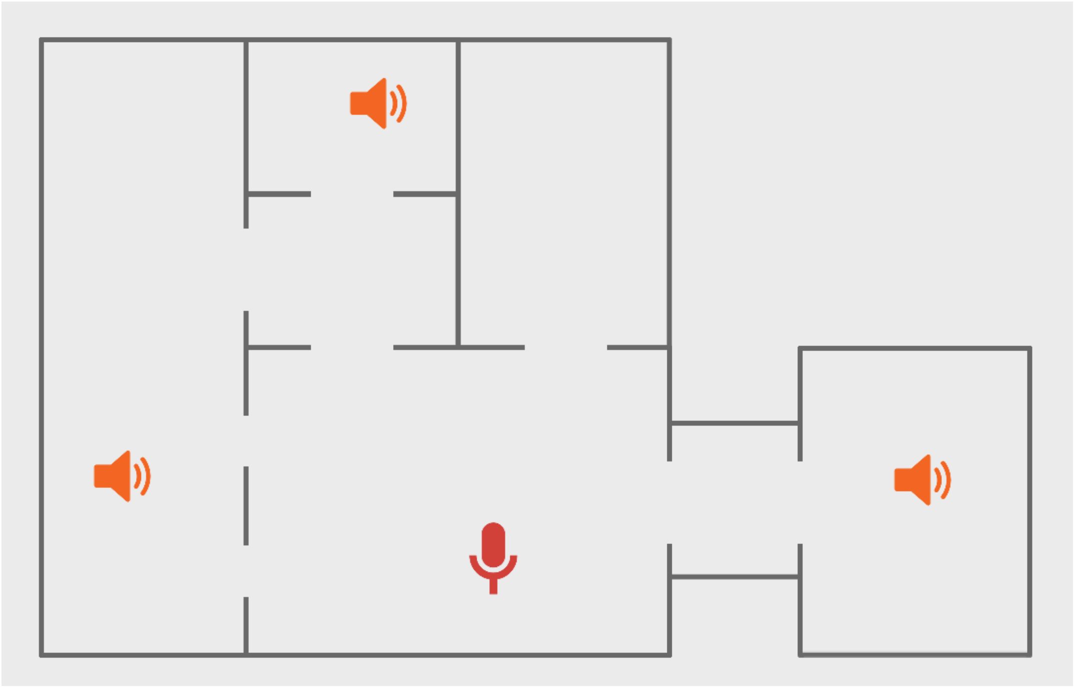
Translating occlusion value

- Occlusion value is a float in range from 0 to 100
- It is translated into attenuation and filtering (e.g. low-pass filter) applied on sound
- We use RTPCs in Wwise to have different occlusion settings for different sounds



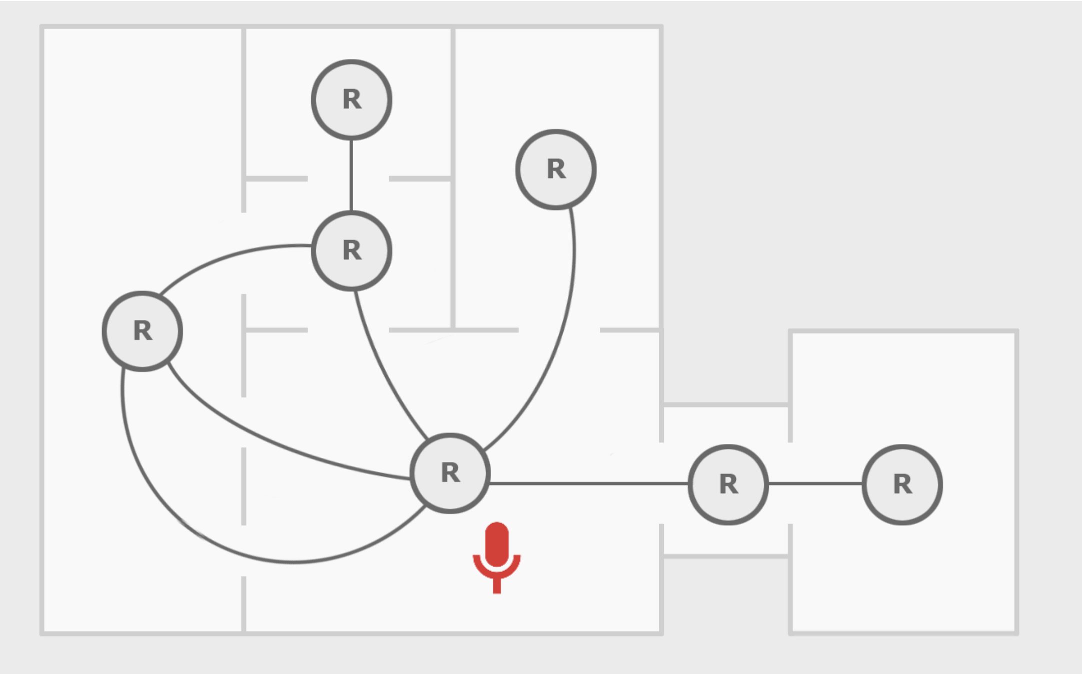
Computing propagation paths

- Sound can reach listener via multiple rooms and portals
- How do we compute occlusion value efficiently?



Computing propagation paths

- Interconnected rooms and portals form a graph
- Rooms are nodes, portals are edges
- Traverse the graph, starting from the listener's room, to compute occlusion value for each room



Obstruction

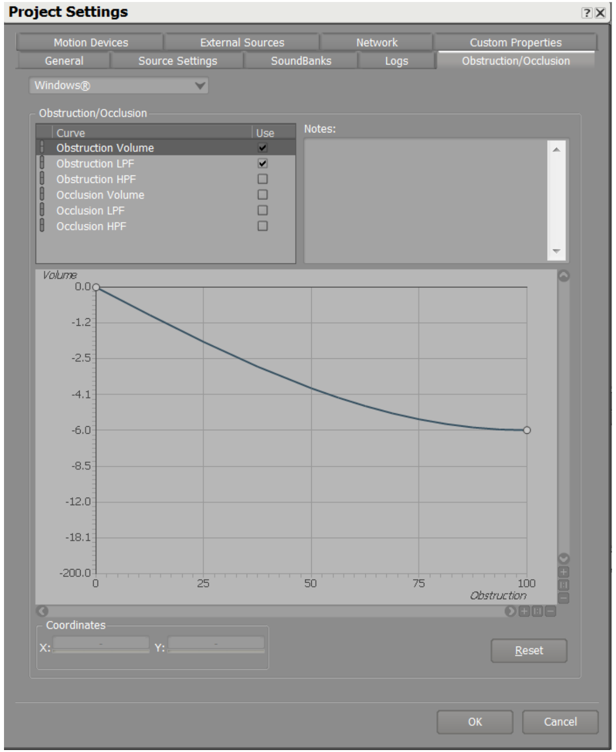
- Before talking more about the graph…
- What about obstruction by objects?
  - Used for mild attenuation (-9 dB) in Hitman
  - Cannot use propagation geometry
    - It doesn’t include dynamic objects or static object (e.g. crates)
    - Level geometry (e.g. walls, doors) should be ignored by obstruction

Obstruction

- Cone raycast
  - Obstruction value depends on how many rays hit an obstacle
- If you want to reduce number of raycasts
  - Single raycast with fade-out when ray hits and fade-in when it doesn’t
- Raycast obstruction is computed only on emitters in:
  - Listener’s room
  - Different room, but with direct line-of-sight between emitter and listener through a portal
    - Less emitters to process + we ignore emitters occluded by walls

Translating obstruction value

- Obstruction value is a float in range from 0 to 100
- It is translated via global obstruction volume/LPF curves in Wwise



Agenda

- About Hitman
- Motivation
- Solution
- **Implementation**
- Challenges
- Conclusion

Sound registration

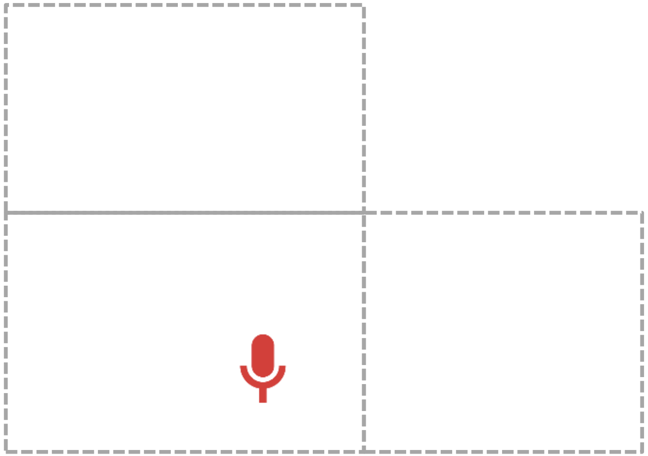
- Every 3D sound registers itself with the room it is located in
  - At game start
  - When its position changes
  - Its room's shape changes
- This requires fast routine for room lookup
  - Described later

Computing propagation paths

- At runtime, traverse graph at regular intervals
  - Not necessarily on every frame
    - 10-15 FPS is fine
  - Can be done in a separate thread
- After each traversal pass…
  - Apply computed occlusion values on sounds
  - Calculate positions and falloff scaling for each sound

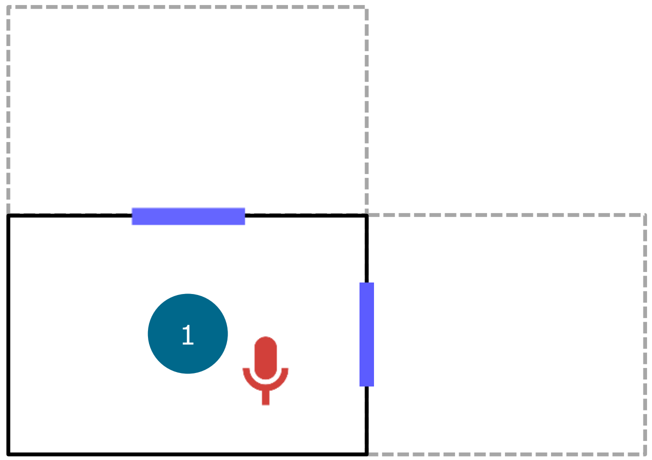
Traversal routine

- Start with all rooms marked as fully occluded
- Perform breadth-first traversal of the graph
    - Have a queue of rooms to traverse



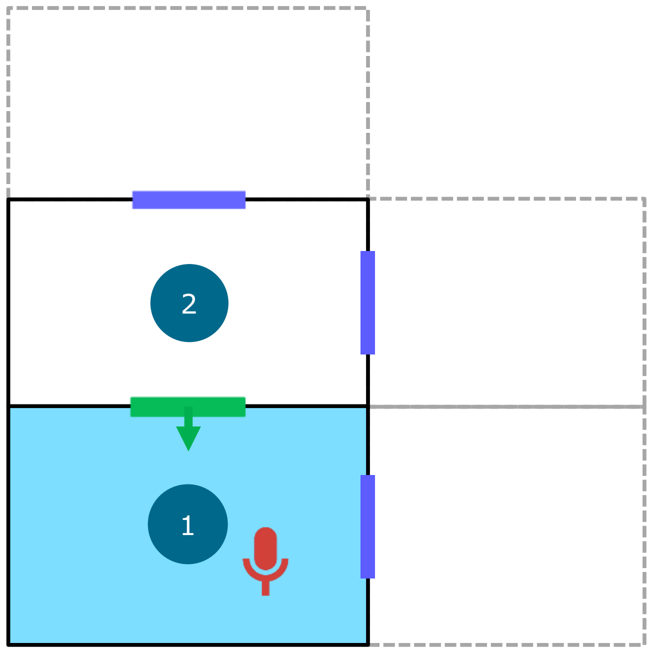
Traversal routine

- Start with all rooms marked as fully occluded
- Perform breadth-first traversal of the graph
  - Have a queue of rooms to traverse
  - Find and enqueue listener's room
  - For each queued room, compute occlusion for each portal and queue connected rooms if they are not fully occluded
  - Process the queue until it's empty

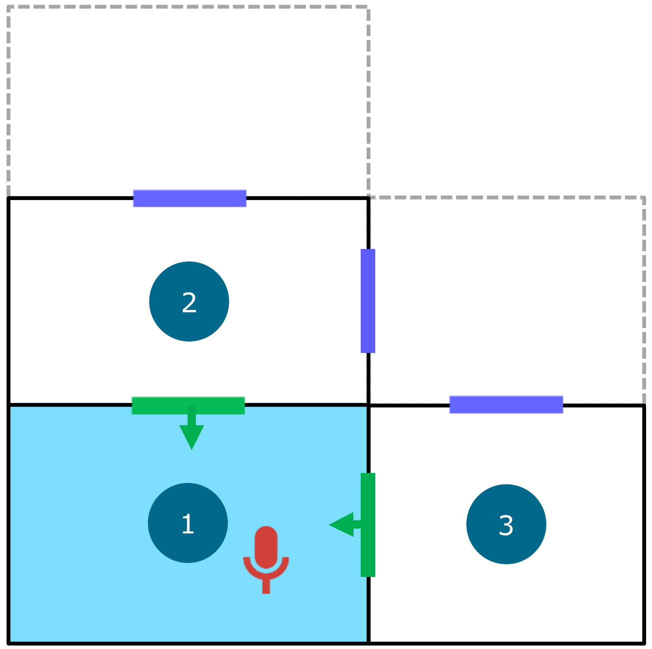


Traversal example

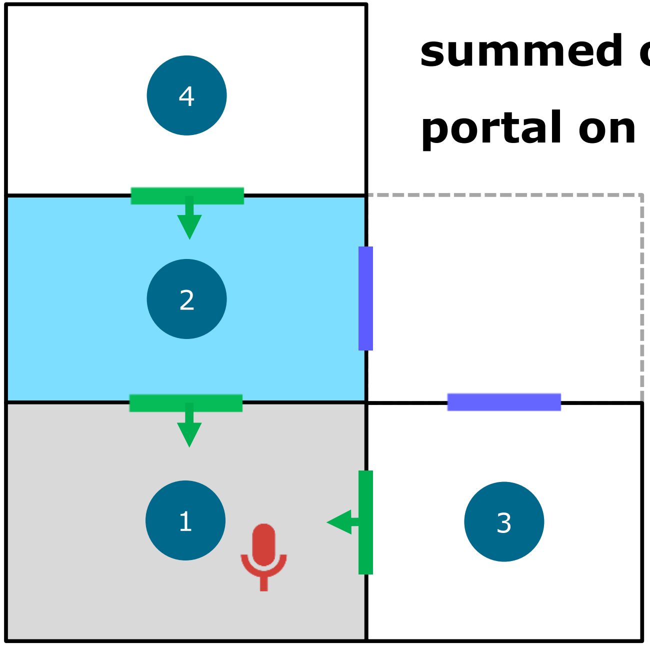
- Compute occlusion value for green portal
- Arrow indicates direction of sound propagation
- Store computed value on room 2 along with a portal pointer
  - Only if it's lower than room's current value
- Add room 2 to the traversal queue



Traversal example

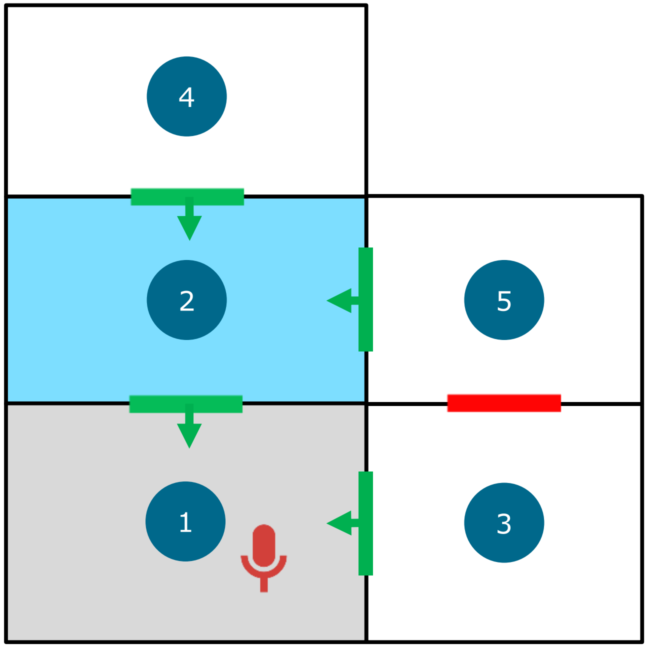


Traversal example



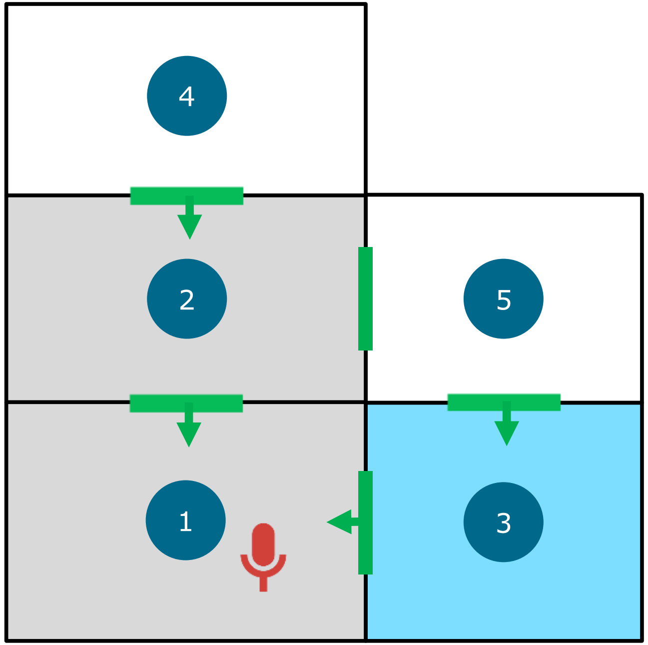
Occlusion value for 4 is summed occlusion of each portal on its way to listener

Traversal example

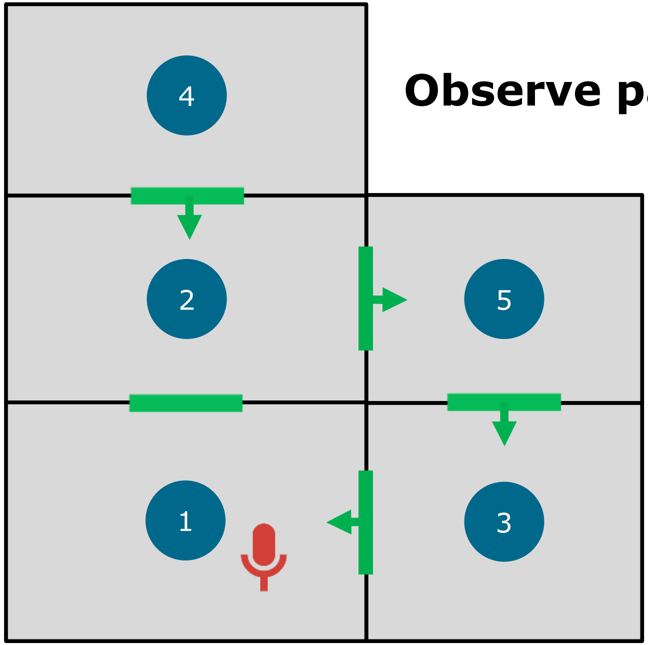


Traversal example

- What if path from room 3 into room 5 is better than from room 2?
    - Lower occlusion value
- Override occlusion value and portal pointer on room 5
- **Re-queue room 5**
    - If it isn't already in the queue
    - Because room 2 and room 4 occlusion has to be recalculated!



Final propagation paths



Observe path from 2 **into** 1

Traversal rules

- We do not queue fully occluded rooms
- We do not process a portal if…
  - … it is farther away from listener than a certain scripted maximum distance
- This makes subset of traversed, "audible" rooms a very small part of total geometry

Fast room lookup

- GetRoomFromPoint()
- We use spatial subdivision data structure “Implicit grid”
  - May already be implemented in your check physics or render code
  - Described in detail in Real-Time Collision Detection (2005, C. Ericson)
  - Optimal when number of rooms is relatively low
- Other spatial subdivision techniques might be used
  - BVH
  - Octree

Agenda

- About Hitman
- Motivation
- Solution
- Implementation
- **Challenges**
- Conclusion

Challenges

- Reducing overhead of maintaining geometry
  - Good start: put portals within templates of graphical openings (doors, windows)
  - Auto-generate geometry
    - Must be possible to tweak generated results manually
- Reverb

Reverb

- Sound should reflect environment type
  - e.g. shots fired in a tunnel
- Every room should have a reverb preset
- Easy solution
  - Apply listener’s room reverb on all sounds
  - Crossfade with reverb of the room which listener is moving into
- Negatives
  - Reverb for every sound is played in all speakers (no reverb directionality)
  - Gives inaccurate clue about environment for sounds not in the listener’s room

Reverb – better solution

- Do not apply listener's room reverb on all sounds
- Find 'optimal' (usually longest) reverb

Optimal reverb

- For sounds not in the listener’s room
  - Of all rooms that sound propagates through, pick the one which has reverb with highest weight
  - Highest weight = longest reverb by default, but can be overridden manually
- Potentially many reverb instances playing
  - Even if we submix sounds that use same preset
  - Works with fast reverbs

Reverb directionality

- Using single-channel un-panned reverbs means…
  - Reverb for the guy shooting on your left will play in all speakers
  - Not a problem, until we have, say, 10 audible rooms = 10 reverbs in all speakers ☺
- It may sound bad

Reverb directionality

- Use single-channel reverbs, but…
  - Have one instance (auxiliary bus) per room
  - Pan them at runtime
- Or, use multi-channel reverbs
  - Can be expensive

Agenda

- About Hitman
- Motivation
- Solution
- Implementation
- Challenges
- **Conclusion**

Advantages of our system

- Delivers immersive and consistent results
- Quite fast to implement
- Inexpensive CPU- and memory-wise
- Scalable
- Supports dynamic and irregular geometry

Advantages of our system

- Support for dynamic/destructible geometry
  - Occlusion values are modifiable at runtime
  - Rooms and portals are modifiable at runtime
    - No offline export step
  - Raycast-based obstruction
- Support irregularly-shaped geometry
  - Rooms and portals can have arbitrary shapes and be grouped

Credits

- People who designed and first used the system:
  - **Frank Lindeskov** – Lead Sound Designer
  - **Jonas Breum Jensen** – Senior Sound Designer

Thank you!

Questions?

stepanboev@gmail.com

https://se.linkedin.com/in/stepanboev