La propagation du son dans le jeu vidéo

Dossier personnel de 3ème année - Thibaut Monnet

Sommaire

Introduction

La fidélité sonore, au même titre que la fidélité visuelle, joue un rôle important dans l'expérience de jeu. Simuler la propagation du son dans l'environnement est un des points clés dans l'immersion du joueur, mais aussi un véritable défi technique.

En effet une simulation physiquement exacte de la propagation du son est non seulement complexe à développer, mais aussi coûteuse en temps de calcul. La production d'un jeu vidéo implique de nombreuses concessions, et la partie sonore du jeu en pâtit souvent la première avec des limites fortes en terme de ressources allouées, que ce soit au niveau de la mémoire ou de la charge de calcul.

Cependant le but visé est l'immersion et non pas l'exactitude. Les développeurs graphiques l'ont d'ailleurs très bien compris. La propagation de la lumière dans les jeux n'est jamais exacte. Il en va de même pour la propagation du son. Les comportements du son les plus évidents pour le joueur doivent être implémentés avec soin pour une expérience crédible mais les limites imposées par le média laisse l'optimisation au centre du développement.

Comment implémenter de manière efficace la propagation du son dans un jeu vidéo ?

Je tenterais de répondre à cette question en étudiant les comportements physiques notables du sons et leurs différentes implémentations dans les jeux modernes.

"Tom Clancy's Rainbow Six: Siege" - Ubisoft

I) Comportement physique du son

Le son est une vibration mécanique d'un fluide, qui se propage sous forme d'onde. Il respecte donc la physique ondulatoire qu'on utilise pour calculer le comportement de la lumière.

A) Réflexion

La réflexion du son correspond aux rebonds sur les surfaces.

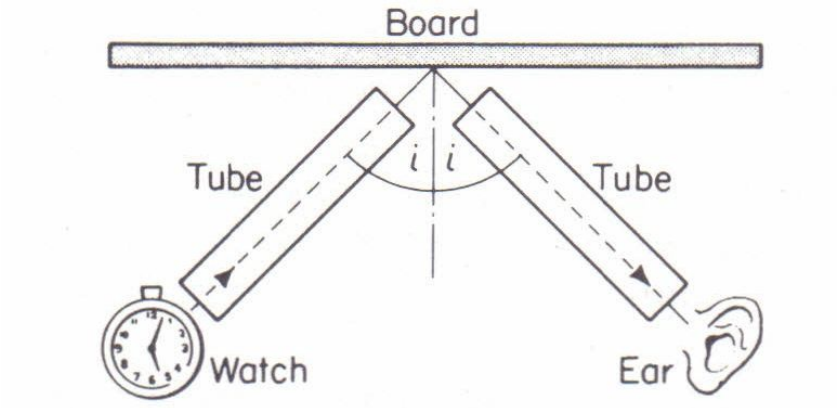
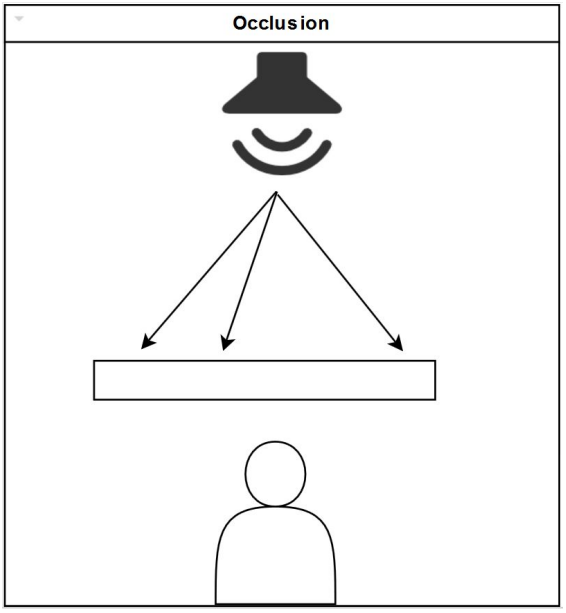


Fig. 27.5. Reflection of sound

L'onde sonore rebondit symétriquement à la normale de la surface. C'est cette propriété qui provoque la **réverbération** et les **échos** lorsque le son revient vers l'auditeur.

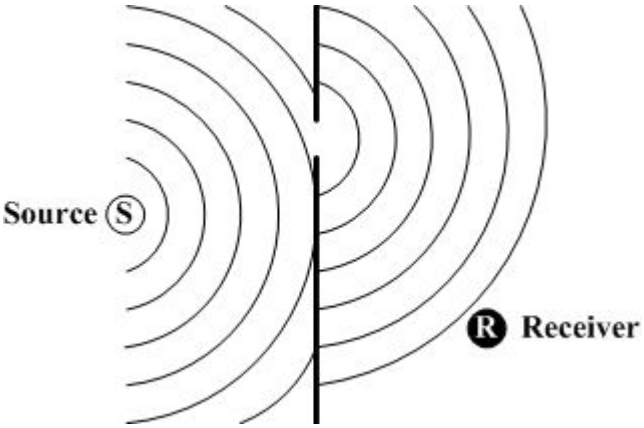
B) Absorption

L'absorption correspond à l'intensité de son atténué par le milieu traversé. Un mur par exemple va stopper une grande partie de l'onde sonore qui l'atteint.
Cette propriété correspond à l'**occlusion** du son.



C) Diffraction

Lorsqu'une onde sonore atteint un obstacle ou une ouverture, elle va se tordre sur ses bords et se diffuser partiellement au delà.
Le son se diffusera plus largement si la taille de l'ouverture correspond à sa longueur d'onde.



Il existe d'autres propriétés en physique ondulatoire qui s'appliquent aussi à la propagation du son, telles que la réfraction ou les interférences. Cependant les effets perceptibles par l'auditeur sont négligeables. Ces propriétés ne seront pas détaillées ici car elles sont simplement ignorées par les programmeurs lors de l'implémentation.

II) Implémentation

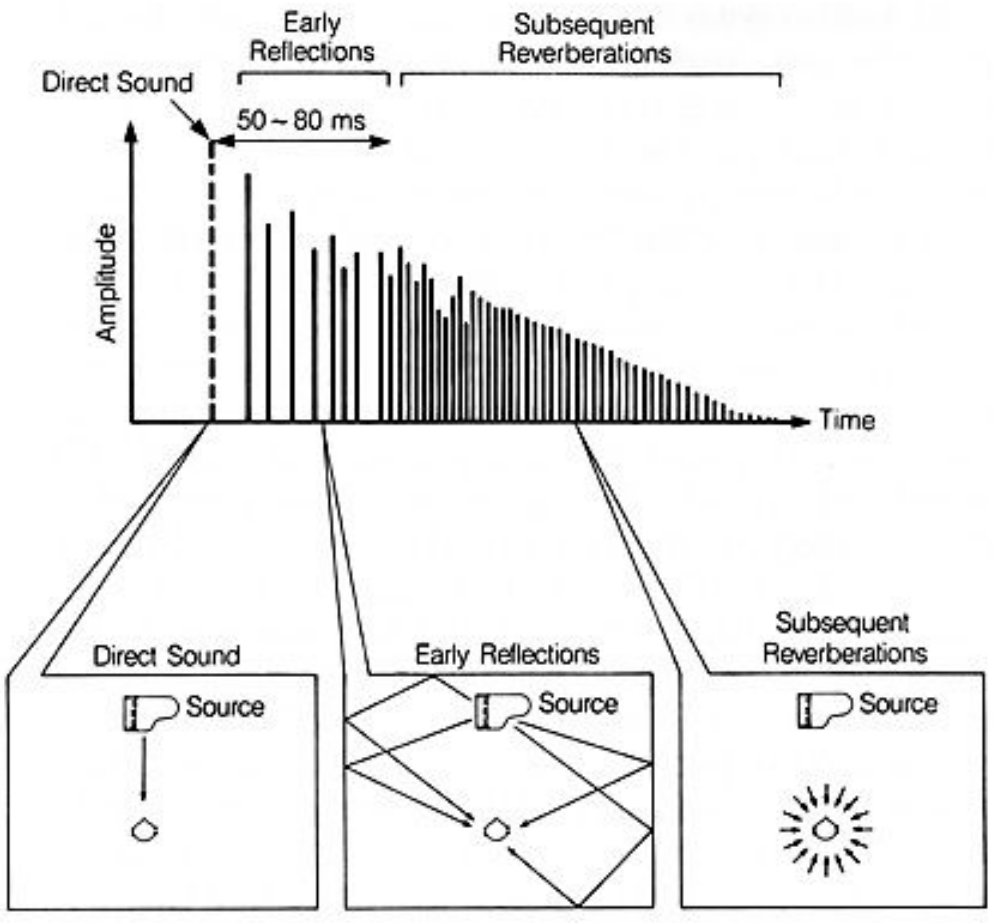
A) Réverbération

La réverbération du son correspond à la réflexion des ondes sonores dans l'environnement. Une réverbération réelle comporte trois parties importantes du point de vue psychoacoustique:

- le son direct qui correspond au signal original non altéré. Varier son volume indépendamment du volume des autres parties de la reverb donnera une impression d'éloignement de la source sonore plus ou moins prononcée

- les premières réflexions caractérisent la distance des murs les plus proches de l'auditeur/de la source

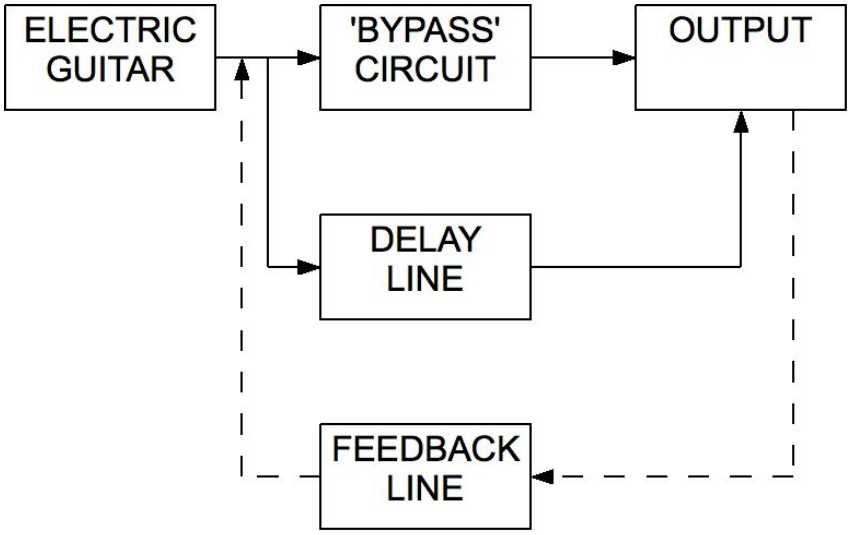
- les réverbérations à proprement parlé, qui détermine la taille globale de la salle (ces réverbérations sont quasi inexistantes si l'environnement est ouvert)



Ils existent deux grandes familles de réverbération numérique : les réverbérations purement algorithmiques et les réverbérations par convolution.

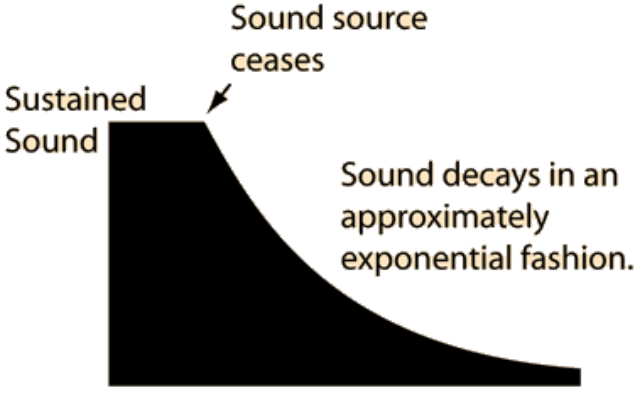
1) Réverbération Algorithmique

Le principe de base est simple. Il est calqué sur celui d'une reverb analogique. Une réverbération est une série d'écho très resserré. Le signal de base passe à travers un réseau de delay. Un delay garde un signal en mémoire puis le joue après un temps spécifié.



Le signal de base est envoyé en sortie, puis renvoyé au delay avec un volume de x % (c'est le feedback), qui le garde en mémoire y secondes avant de le renvoyer en sortie. Dans le cas d'une réverbération la plus simple possible, le y est égale à la fréquence d'échantillonnage du signal (bien souvent 44100 hz). C'est la plus petite valeure possible dans un fichier audio numérique.

La réverbération s'arrête une fois le feedback nul.

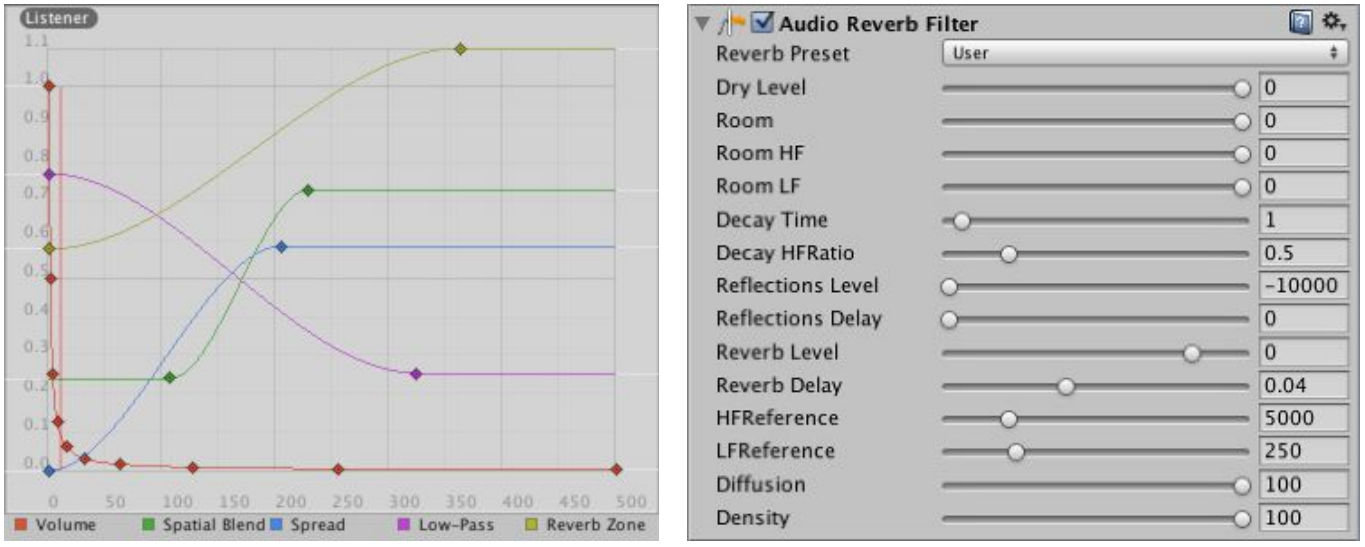


Une réverbération simple en C++ correspondrait à ceci :

```
int delayMilliseconds = 500; // pre-delay
int delaySamples = (int)((float)delayMilliseconds * 44.1f); // 44100 Hz sample rate

float decay = 0.5f; //feedback

for (int i = 0; i < buffer.length - delaySamples; i++) {

buffer[i + delaySamples] += (short)((float)buffer[i] * decay);

}
```

Cette reverb très brute ne donne cependant aucune information en terme de spatialisation. Il existe quantité de paramètres qui permettent d'ajouter cette sensation de spatialisation si importante.
Jouer sur la quantité de feedback envoyé aux canaux droites et gauches, la différence de delay entre ces deux canaux, ou encore équalizer les différentes parties de la reverb va permettre de donner une sensation plus fine des propriétés acoustique de l'environnement qui entoure le joueur.

Le Sound Designer pourra à l'aide de ces outils ajuster les paramètres de réverbération pour chaque environnement/salle du jeu.



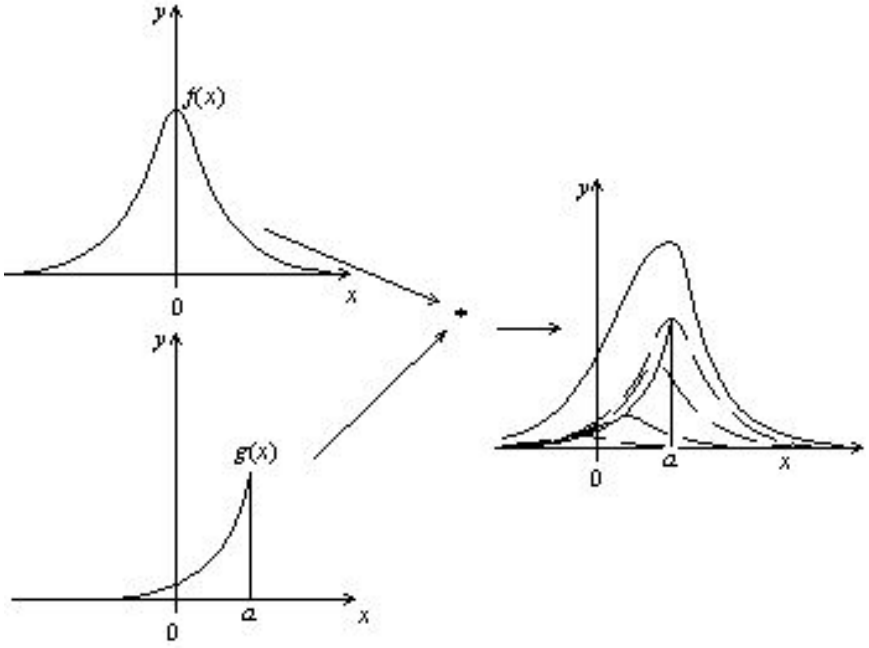
Outils de réverbérations - Unity3D

2) Réverbération par Convolution

Le produit de convolution en mathématiques permet d'obtenir à partir de deux fonctions une troisième fonction qui correspond à un "mix" des fonctions originales. Le résultat est obtenu en calculant l'intégrale d'une fonction en tout point de son domaine, pondérée par la seconde fonction autour de l'origine.

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(x-t)g(t)\,\mathrm{d}t = \int_{-\infty}^{+\infty} f(t)g(x-t)\,\mathrm{d}t$$

Exemple graphique:



La convolution n'est pas calculée telle qu'elle en informatique car le calcul d'intégrale est une opération extrêmement complexe et coûteuse.

En effet, il existe une propriété bien pratique lié à la transformée de Fourier : la transformée de Fourier d'un produit de convolution s'obtient par multiplication des transformées de Fourier des fonctions.

$$\mathcal{F}(f * g) = \mathcal{F}(f)\mathcal{F}(g)\ ;$$

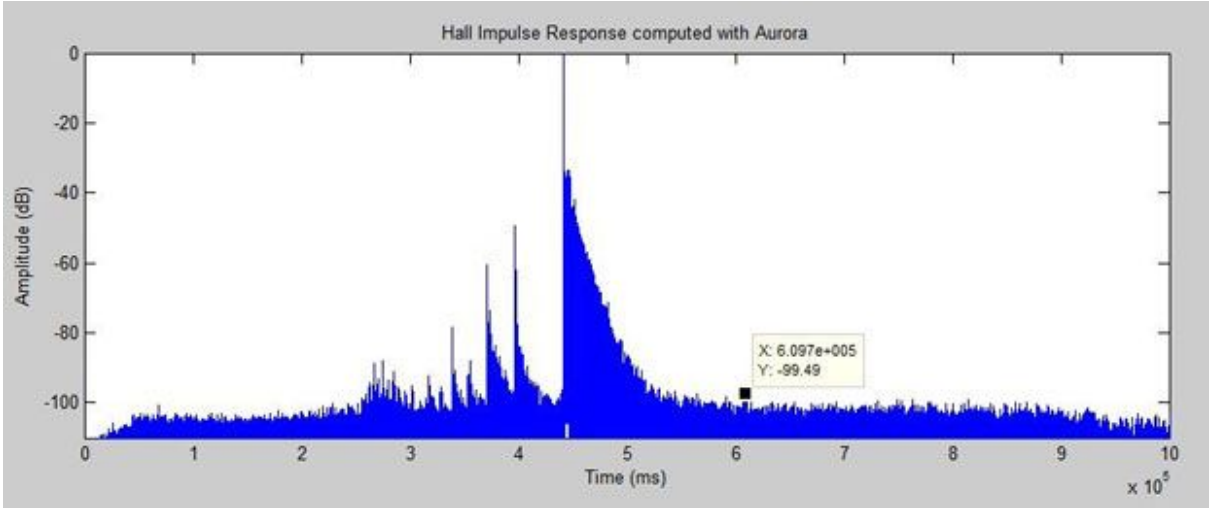
Il suffit donc de calculer la transformée de Fourier des deux fonctions, puis de les multiplier entre elles, et enfin de calculer l'inverse de la transformée de Fourier du résultat pour obtenir la convolution. Il existe des algorithmes rapides de transformée de Fourier (FFT) qui permettent de réduire drastiquement le temps de calcul de la convolution.

La réverbération par convolution n'est rien de plus que la convolution de deux sons: le premier correspond au signal original, le second à la réponse impulsionnelle de l'environnement.

Cette réponse impulsionnelle peut être obtenus de deux manières: à partir d'enregistrement d'un environnement réel, ou à partir d'une simulation précise basée sur le raytracing.

Calculer la réponse impulsionnelle d'un environnement 3D via le raytracing est une opération coûteuse. Cela nécessite de connaître les coefficients d'absorption des différents matériaux dans la scène, de lancer de multiple rayons et de calculer les rebonds de chacun d'entre eux dans l'espace (en prenant en compte la diffraction) pour recréer le signal d'arrivée. C'est une opération très difficile à calculer en temps réel, mais au résultat beaucoup plus réaliste que celui d'une simple reverb algorithmique. Cela correspond plus ou moins au baking des lights dans les moteurs de rendus 3D, version sonore.

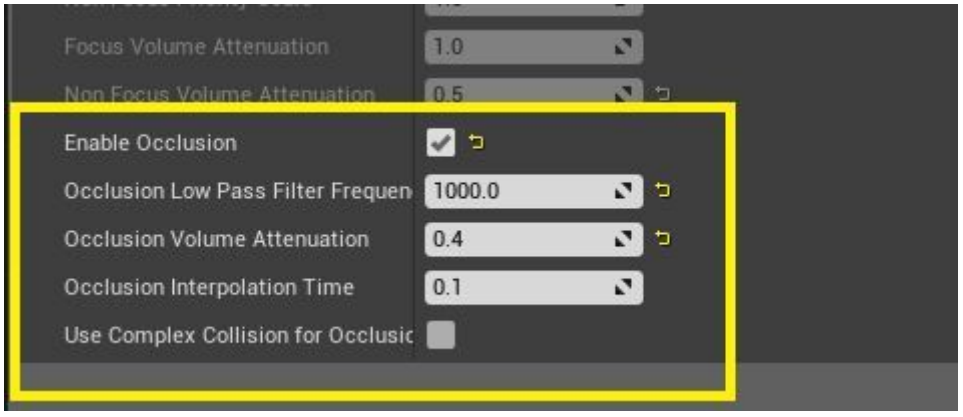
Lorsqu'un jeu utilise de la réverbération par convolution, la réponse impulsionnelle de l'environnement est précalculée. Il suffit ensuite de calculer la convolution de chaque son joué dans la scène avec la réponse impulsionnelle de la salle.



Réponse impulsionnelle d'une salle de concert

B) Occlusion (/Obstruction)

L'occlusion est probablement l'effet le plus simple à simuler.
Si la source sonore n'est pas dans le champ de vision immédiat de l'auditeur, il y a occlusion. En cas d'occlusion totale on coupe le volume de la source; si il y a seulement obstruction on étouffe le son de la source à l'aide d'un filtre passe-bas pour laisser passer uniquement les basses fréquences (en fonction du coefficient d'obstruction de l'obstacle). Un simple raycast entre la source et l'auditeur suffit pour vérifier si l'occlusion doit avoir lieu ou non. C'est une méthode employé par la plupart des moteurs de jeux.



Occlusion parameters - Unreal Engine

Certains moteurs comme le CryEngine permettent au développeur d'attacher à leurs materials un coefficient d'obstruction pour que chaque objet du jeu bloque le son à sa façon.

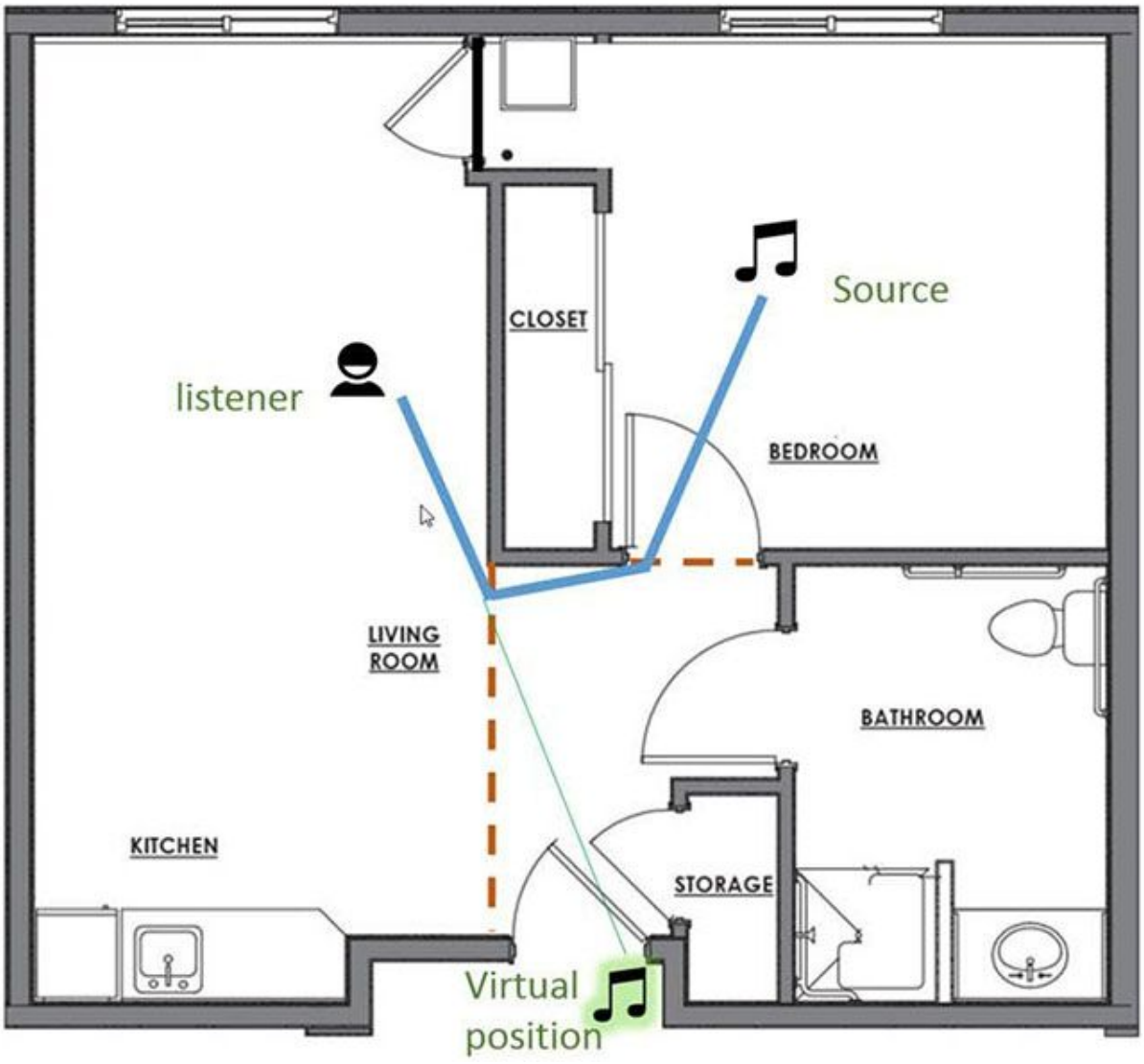
```
<SurfaceType name="mat_fabric">
 <Physics friction="0.8" elasticity="0" pierceability="7" can_shatter="1" sound_obstruction="0.5" />
</SurfaceType>
```

material - CryEngine

L'occlusion va de pair avec la diffraction; l'un sans l'autre les deux effets ne semblent pas réalistes.

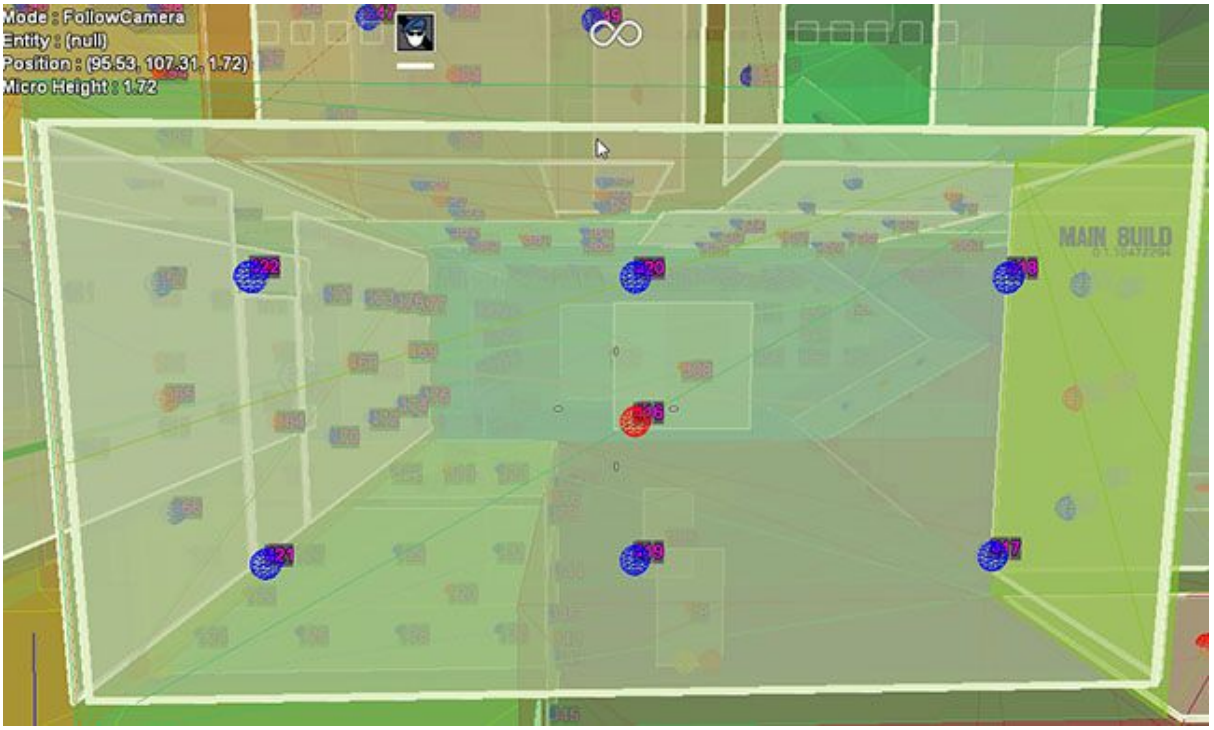
C) Diffraction

La méthode la plus simple pour simuler la diffraction est d’avoir recours à des sources virtuelles.



Si il y a occlusion de la source originale, il suffit de créer une seconde source audio dans le champ de vision du joueur qui rejoue le son de la source originale. Pour la positionner correctement, on utilisera un simple algorithme de pathfinding entre la source originale et le joueur. Le segment du chemin calculé le plus proche du joueur servira de vecteur pour positionner la source. La longueur totale du chemin permettra de positionner plus ou moins la source virtuelle du joueur et d’altérer son rendu à volonté (en ajoutant un delay initial, ou avec un equalizer par exemple) pour une expérience plus réaliste.

C’est cette technique qui est utilisé dans les moteurs de jeux d’Ubisoft depuis Assassin’s Creed 1. Dans un des derniers jeux de la firme, “Tom Clancy's Rainbow Six: Siege”, la simulation de la diffraction/obstruction va encore plus loin. En effet les murs sont destructibles -partiellement- ce qui complique la tâche. Si un partie du mur est détruite le navmesh de la salle reste le même, mais le pathfinding pourra trouver son chemin à travers le petit trou créé par le joueur. Un système de node a été mis en place pour permettre cela.



Debug view - Tom Clancy's Rainbow Six: Siege

Chaque mur du niveau possède un certains nombre de nodes qui sont autant de chemin possible dans le navmesh. Chaque node dispose d’un poids plus ou moins élevé. Si le mur est intact, le poids du node est infini et le pathfinding trouvera son chemin autrement. Si le mur est légèrement détruit, le pathfinding trouvera son chemin à travers le node, mais la source virtuelle sera étouffée. Si le mur est quasi détruit, le poids du node s’approche de zéro, et le son passe presque inchangé au travers du mur.

C’est un moyen simple mais efficace de gérer la diffraction dans un environnement destructible.

Les sources virtuelles peuvent également être utilisées pour simuler la diffraction sur des objets dynamiques. En cas d'occlusion du son par un objet entre le joueur et la source, on va placer deux sources virtuelles, à droite et à gauche des bords de l'objet.

Adaptive Audio Occlusion plugin - Unreal Engine

Les systèmes qui ont été présenté ici restent avant tout des outils au service du Sound Designer. C'est pourquoi aucunes valeurs précises ne sont données. Tous ces systèmes doivent rester totalement paramétrables. Le réalisme est important, mais il ne doit pas desservir le gameplay et le Sound Designer doit être en capacité de "tordre" le comportement de ces systèmes à volonté.

De plus, aucune de ces techniques ne sont physiquement correct(en dehors de la réverbération par convolution, qui reste malgré tout statique). L'optimisation reste la priorité.

III) Performances

En terme de performances la seule opération vraiment coûteuse est le calcul de la réverbération.

A) CPU vs GPU

Le calcul de la réverbération par convolution en GPGPU donne de très bon résultat. En effet, le calcul de la convolution nécessite d'effectuer de très nombreuses transformées de Fourier, qui gagnent à être massivement parallélisées.

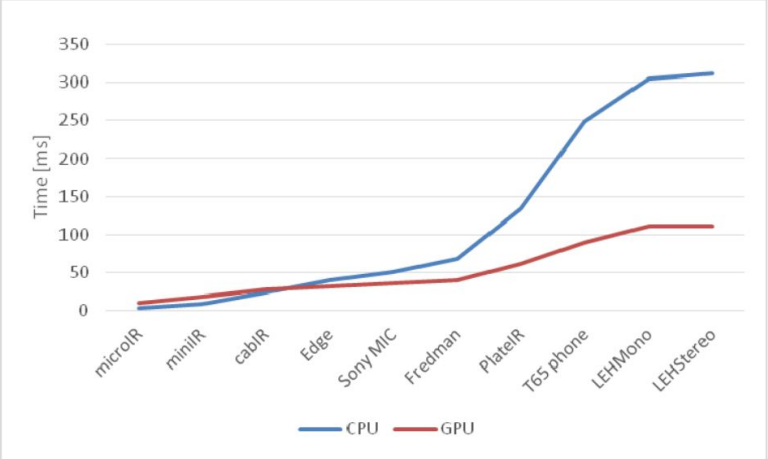


Fig 9 Comparison of CPU and GPU implementations for different impulse response sizes.

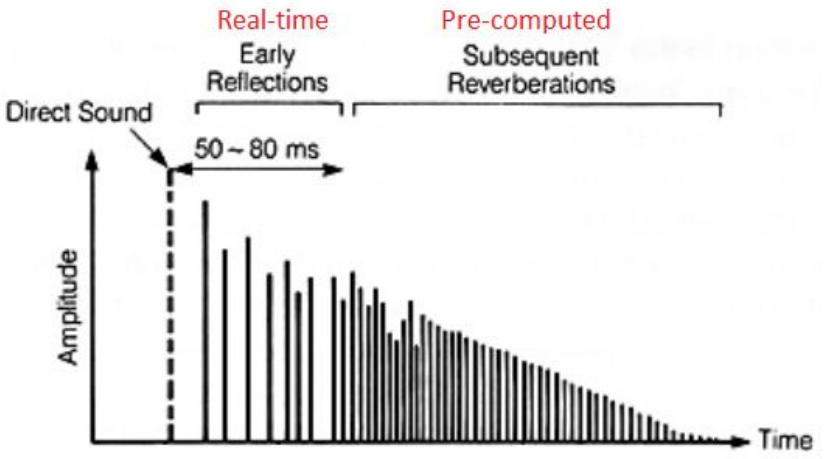
Les moteurs audio exploitant le GPGPU sont cependant très peu répandus dans l'industrie. Cela est dû à la portabilité réduite du code notamment sur mobile, et au peu de budget accordé au son dans les productions, aussi bien en terme de ressources humaines que de ressources système, particulièrement en terme de GPU.

L'Aspic Engine d'Aspic Technologies utilise par exemple une solution 100% CPU, malgré l'utilisation de la convolution, car le moteur vise également les plateformes mobiles.

À l'inverse le jeu exclusif à la Xbox 360 "Crackdown" sorti en 2007, qui utilise le même système que l'Aspic Engine, rend son audio entièrement via GPGPU. La question de la portabilité ne se posant pas, le jeu profitait du gain de vitesse lié au GPGPU et disposait d'un système de réverbération par convolution en temps réel très avancé pour l'époque.

B) Realtime vs Precomputed

Le système de raytracing utilisé pour calculer la réponse impulsionnelle d'un environnement peut également être utilisé en temps réel, mais il est très lent. Certains moteurs comme l'Aspic Engine calculent les premières réflexions en temps réel de cette manière et les réverbérations en fin de signal par convolution. C'est ce même système qui a été utilisé sur Crackdown.



Un jeu avec une réverbération convolutive totalement précalculé aura un rendu audio d'excellente qualité, mais tout l'environnement devra être statique. Un système 100% temps réel est encore infaisable avec le hardware actuel.

La question du temps réel vs précalculé ne se pose pas vraiment, les systèmes audio les plus efficaces étant des hybrides temps réel/précalculé. C'est cet équilibre entre données calculées en temps réel et données précalculées qui donne les résultat les plus réalistes tout en restant rapide.

C) Optimisations diverses

- La fréquence d'actualisation des algorithmes d'occlusion et de diffraction ne doit pas forcément être la même que celle du rendu graphique. Les jeux old gen de chez Ubisoft rafraîchissent leurs algorithmes d'occlusion et de diffraction seulement 5 à 10 fois par seconde, en fonction de la charge CPU globale du système.

- Il est important que les sources se déplaçant le plus rapidement soient updaté en premier par le moteur audio.

- Le pathfinding utilisé pour calculer la diffraction peut se limiter à la 2D.

- Il est important d'utiliser un système de zoning pour charger/décharger les données géométriques inutiles lors du calcul du pathfinding pour la diffraction.

Conclusion

Implémenter la propagation du son dans un jeu vidéo fait appel à de nombreuses techniques pour simuler les propriétés physiques du son :

- Le phénomène de réflexion peut être simulé à l'aide de réverbération
- L'occlusion et la diffraction sont des phénomènes dont les simulations sont liés, en utilisant diverses astuces, notamment l'aide de raycasts, de filtres fréquentiels, de sources virtuelles et de pathfinding.
- L'utilisation du GPGPU dans la simulation de la propagation du son est efficace mais peu courante pour des raisons de portabilité.
- Les réverbérations réalistes les plus optimisées utilisent souvent un système hybride temps réel/précalculé.

Une bonne implémentation de la propagation du son passe avant tout par des compromis. Le système doit être efficient et simple d'utilisation, car il reste un outil au service du Sound Designer.

Sources

Real-time Sound Propagation in Video Games by Jean-François Guay

StackOverflow

Room acoustics modeling using the raytracing method: implementation and evaluation by David Oliva Elorza

Interactive Physically-based Sound Simulation by Nikunj Raghuvanshi

Hyperphysics

Analysis of CPU and GPU Implementations of Convolution Reverb Effect - Telfor Journal

Dynamic audio in destructible levels in Rainbow Six: Siege by Louis Philippe Dion

The Next Big Steps In Game Sound Design by Damian Kastbauer

Surround Sound Acoustic Measurement - University of Victoria

Use of GPUs in room acoustic modeling and auralization - ISRA 2010

Aspic Technologies