Sound Programming

제9장 Introduction to Effects Extension in OpenAL

Effects Extension

- **OpenAL**에서 제공하는 기본 **3D** 음향 효과
  - distance based roll-off / directivity & velocity / Doppler Shift
- 기본 **OpenAL**에는 없던 환경 영향 시뮬레이션 확장 기능
  - 반향(Reveberation)
  - 반사(Reflections)
  - Sound Occlusion(폐쇄) or obstruction(방해물) by intervening objects

➔**Environmental Audio**와 **Environmental Filtering** 기능을 통해 이들 음향 효과를 지원하는 확장 기능이 **"Effects Extension"**.

Environmental Audio

- **Reverberation(반향) 효과가 필요한 이유 예**
  - 방음 장치가 되어 있는 방에서의 칼 싸움 소리와 큰 성당 같은 곳에서의 칼 싸움 소리는 달라야 함
- **Sound Occlusion(폐쇄, 차단) 효과가 필요한 예**
  - 바로 옆에서 들리는 비명 소리와 옆 방에서 들리는 비명 소리
- **Environmental Audio(환경 음향 효과)**
  - 같은 음원이 환경 조건에 맞게 들릴 수 있게 함으로써 콘텐츠의 몰입도를 높이는 효과
- **cf.) Environmental Filtering(환경 구조 필터링)**
  - 환경의 구조적 특징을 모델링하여 달리 들릴 수 있도록 하는 효과

Reverberation

- **Reverb와 Reflection 음향 효과를 총칭**
  - 3D 음향 환경에 실제와 같은 현실감을 주기 위함.
  - 3D 환경 사용자가 무의식적으로 3D 세계에 빠질 수 있도록 감정적 깊이를 더해주는 효과도 있음
  - 심지어는 보이지 않는 3D 환경에서도 효과를 발휘함
    - 예) 동굴 속 칠흑 같이 어두운 연못가! 연못에 물 한 방울만 떨어져도 물방울 소리의 전파를 느낄 수 있을 것이고, 연못 주위의 동굴로 인해 그 전파 효과는 더욱 커짐을 보이지 않아도 알 수 있을 것임
- **이러한 Reverb와 Reflection 효과를 만들어내려면 이들에 대한 이론적 이해가 필수적임**

1차 반사음(Primary Reflections)

- 어떤 환경에서 음원이 바닥이나 벽, 천장을 한 번 거치고 나서 listener에게 도달하는 현상
- 원음 보다 약간 늦게 listener 귀에 도착할 것임
  - 원음과 합쳐져 전체적으로 소리가 좀 크게 들리는 느낌도 나고,
  - 한 번 반사된 소리는 원음과는 음색이 다르게 느껴지기도 함.
  - 반사판이 음원과 멀리 떨어져 있으면 거의 "echo" 효과가 날 것임.
- 반사된 소리가 원음과 어떻게 다른지가 환경 음향 효과의 중요한 실마리가 됨
  - 즉각적이고 강한 반향은 환경이 listener 가까이에 있다는 의미
  - 음색이 변했다면 환경의 질(벽 종류)이 다름을 유추할 수 있음
    - 어떤 벽은 소리를 흡수하고, 어떤 벽은 소리를 대부분 반사하기도 함.
  - 첫 반사 음만 가지고 환경을 정확히 알아내는 것은 어려움

2차 반사음(Secondary Reflections)

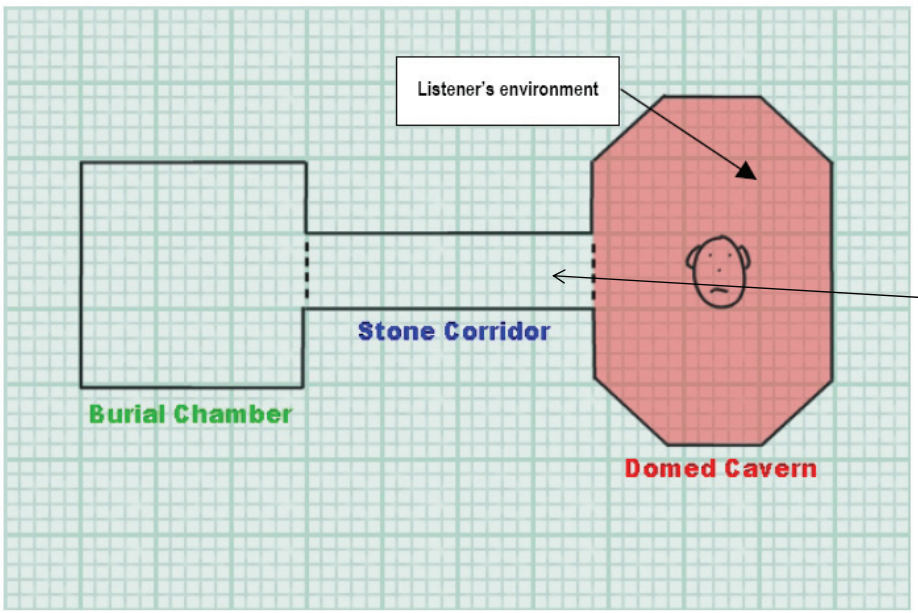
- 1차 반사된 소리가 바닥이나 벽, 천장에 한 번 더 부딪혀 여러 개의 2차 반사 소리를 만들어 냄
  - 2차 반사 소리가 listener의 귀에 도달할 때에는 여러 개가 섞일 가능성이 있음
- 2차 반사 소리는 1차 반사 소리보다는 덜 명료할 것임
  - 2차 반사 소리로 원음의 위치를 인지하기는 더 어려워짐
  - 2차 반사된 여러 음이 섞여서 그렇기도 하고, 2차 반사되면서 공기 중에서 음의 명료함이 떨어지기 때문이기도 함.
  - 2차 반사 과정에서 공기 중에 소리가 흩어지기도(뭉게지기도) 함
  - 벽, 바닥, 천장이 소리를 반사하는 성질이 떨어지는 재질일수록 2차 반사 소리는 더 무뎌질 것임
- 2차 반사 소리의 크기 역시 1차 반사보다 작아짐
  - 귀에 도달할 때까지 시간도 오래 걸렸을 것이고,
  - 전파 과정에서 상당 부분 유실되기 때문이다.

반향 효과의 실체

- 소리는 한 번 반사될 때마다 반사되는 힘을 잃어감. 따라서 반향 효과는 실제로는 "sound tail"에 해당함
  - 이전 반사음에 묻히는 부분을 제외한 반사음의 끝 부분을 재생시켜 주는 효과
  - This sound tail is most commonly called reverberation.
- 반향을 인간의 귀(뇌)가 느끼는 방법
  - 1차-반사음과 2차-반사음 등을 구분하여 듣기는 불가능
  - 모두 합하여 들으면서, 들리는 반향의 길이나 강도로 주변 환경이 어떨지를 이해하게 됨
    - 반사가 많이 되고 방이 클수록 반향이 더 오래 지속된다고 느낄 것이고
    - 반사가 많이 되고 방이 작을수록 반향의 강도는 더 세다고 느낄 것이다
- 반향 제어 요소들
  - room size, reverb level, reflections level, air absorption,
  - reverberation decay time

다중-음향 환경 모델링

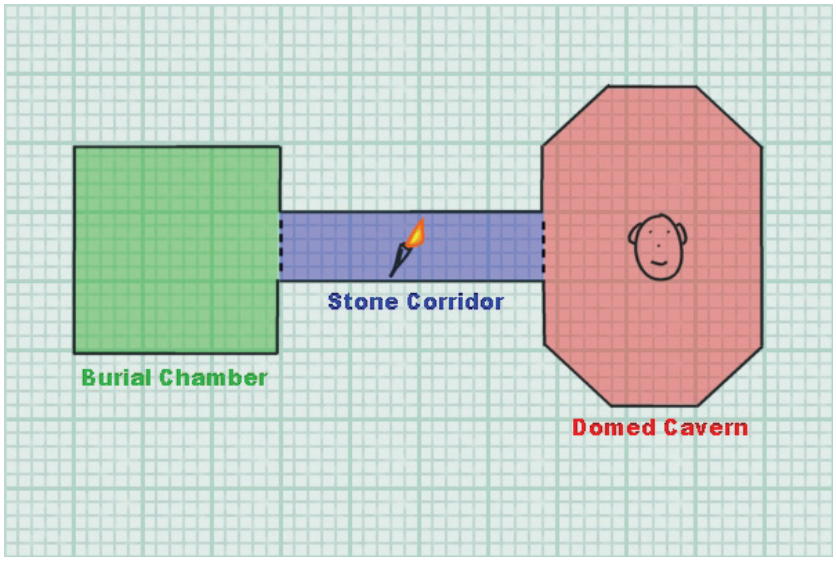
- **With only one reverb, only the listener's environment can be modeled**



Listener가 한 음향 공간에서 다른 음향 공간으로 이동한다면, 그에 맞게 반향 효과도 바꿔주어야 현실감이 있을 것임 ➔ 다중 음향 환경 모델링이 필요한 이유

다중-음향 환경 모델링 (계속)

- 다중 음향-환경 모델링은 응용의 **'soundscape'**을 넓힘



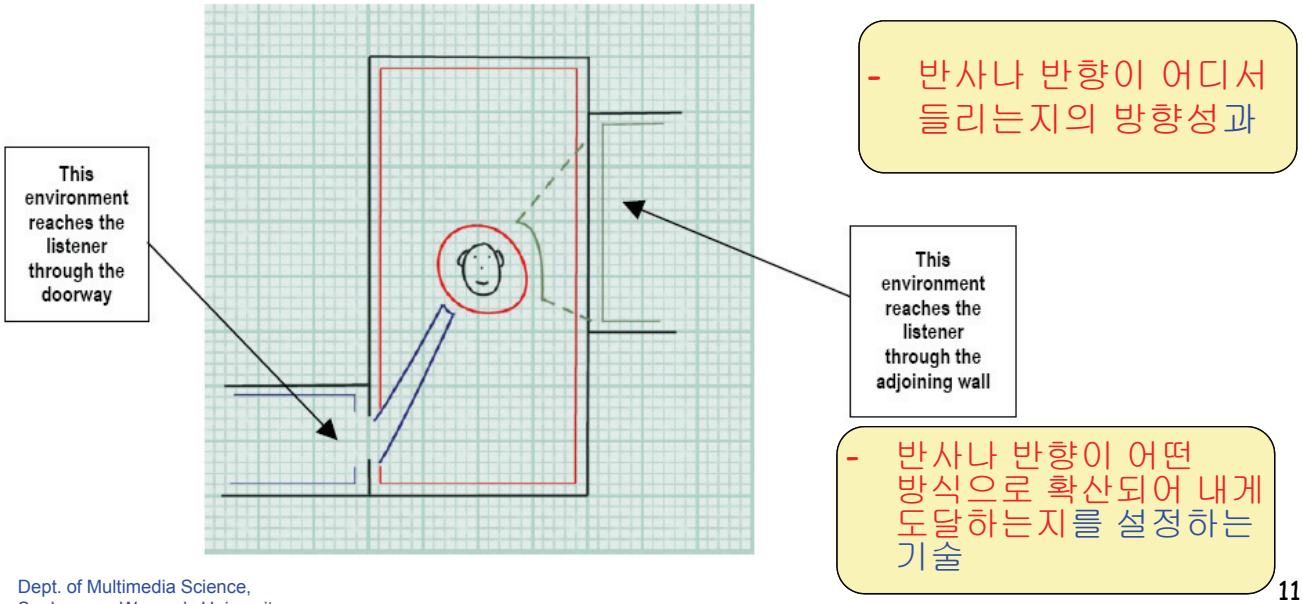
- 단일 음향-환경 모델링으로는 Listener가 듣게 될 정확한 소리를 내기 어려움
- 'Domed Cavern"효과 + muffled flaming torch(filtering effect) 정도만 가능할 것임

다중-음향 환경 모델링 (계속)

- 앞 그림에서 **listener**는 다음의 소리를 들을 수 있어야 함
  - ① 자신이 있는 Domed Cavern 내에서 나는 소리 + 반향
  - ② 횃불 타는 소리가 Stone Corridor와 열린 문을 통해 직접 들림 + Stone Corridor를 거치며 발생한 반향
  - ③ ②에서 발생한 소리가 Domed Cavern에서 일으킨 반향
- 단, 다중 음향-환경에서는 각 반향 소리의 방향성은 느끼기 어렵다는 단점이 존재
  - 반사된 여러 소리가 섞이면 반향의 진원을 찾기 어려워지기 때문
  - 소리의 방향성은 "panning environments" 설정을 통해 가능함

Panning Environments

- **Listener**가 자신이 속해 있지 않은 외부 환경으로부터 반사되어 나오는 소리를 들을 때 외부 환경의 위치를 인지할 수 있게 해주는 기능



- 반사나 반향이 어디서 들리는지의 방향성과
- 반사나 반향이 어떤 방식으로 확산되어 내게 도달하는지를 설정하는 기술

Panning Environments (계속)

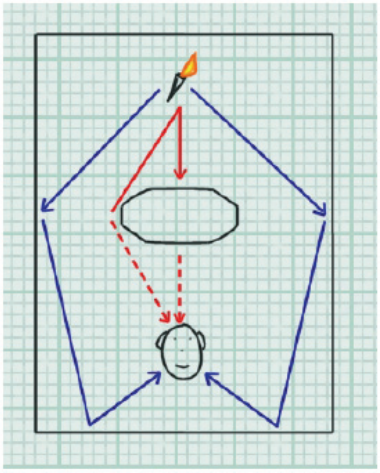
- 사운드 응용은 다른 환경에서 들려오는 소리를 "마치 멀리 어디선가 들려오는 것처럼" 해줘야 함
- 앞의 그림과 같은 예에서
  - 문이 하나 있고, 그 문을 통해 다른 환경으로부터의 소리가 들려오는 효과를 내려면,
  - 그 다른 환경을 위한 **panning parameters**를 "마치 그 문을 통해 다른 환경에서 나는 소리가 들려오는 것처럼"설정해야 함
  - 반대로 문이 없고 벽으로 가로 막힌 다른 환경에서 들려오는 소리처럼 들리게 하려면
  - 그에 부합하는 **panning** 설정과 **occlusion filtering** 설정을 해주면 됨.

Environmental Filtering

- 벽이나 기둥 같은 장애물에 가로막힌 소리 표현을 위한 기술
  - 기둥이나 벽이 소리를 흡수하기 때문에 장애물 없는 소리와는 확연하게 차이가 있음
  - 일종의 **muffling effects**에 해당
  - 벽의 두껍기나 장애물의 재질에 따라 소리 감쇠 정도와 필터링 방식을 조절한다면 현실감 있는 환경 필터링을 구현할 수 있음
- 환경 필터링의 종류
  - Sound Obstruction(방해물)
  - Sound Occlusion(폐쇄)
  - Sound Exclusion(제외)

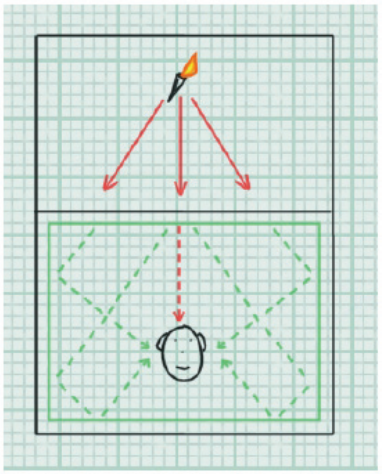
Sound Obstruction

- 음원과 **Listener** 사이에 **direct path**를 방해하는 장애물이 있는 경우에 발생
  - direct-path 상에 있는 장애물이 소리의 일부만 통과시키기 때문.
  - 반사된 소리와 장애물에 의해 일정 부분 소멸된 소리만 듣게 되는 현상
- **Muffled direct sound + normal reflections and reverberation** ➔ 인간의 뇌로 하여금 음원이 장애물 뒤에 있다고 느끼게 함
  - 돌아 나오는 direct sound의 각도가 클수록 "muffled" 효과는 커짐



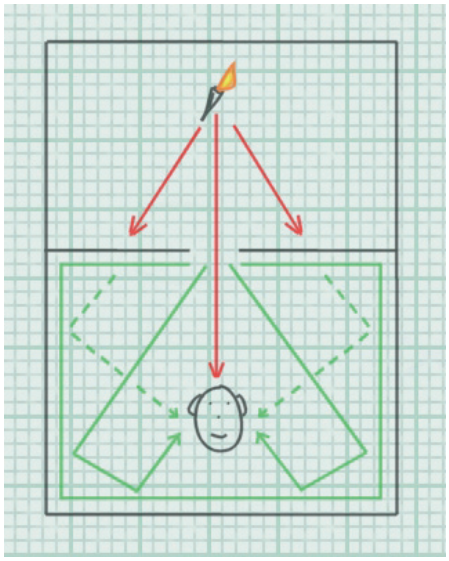
Sound Occlusion

- 장애물이 음원과 **listener**를 완전히 가리는 상황을 의미
- 가로막은 벽이 **filter** 역할을 함
  - 필터를 건너며 high frequency 소리가 filtered out 된다.
- **Obstruction**과의 차이점
  - Direct sound뿐만 아니라 1st, 2nd-reflected sound도 muffled되어 도달한다는 점
- 가로막은 벽의 특성에 따라 **muffled sound**의 품질도 달라짐
  - 밀도, 두께, 강도, 물성, 등



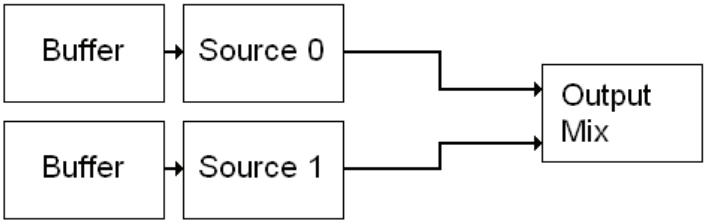
Sound Exclusion

- 음원과 **listener**를 가로막던 벽에 문을 달고 열어두면
  - direct path가 확보될 수도 있다.
  - 하지만 여전히 sound는 listener 방으로부터 격리(exclusion)되어 있음.
- 복잡하고 다양한 사운드 시나리오
  - Direct path가 생길 수도 있지만 음원의 상당 부분은 muffled 될 것임.
  - Listener가 느끼는 음량은, 문의 크기와 문과 음원과의 거리 등에 따라 다르다.
- **Listener**의 위치가 어디냐에 따라 **exclusion**과 **obstruction**이 섞이는 효과가 날 수도 있음.



OpenAL's Architecture for 3D Audio

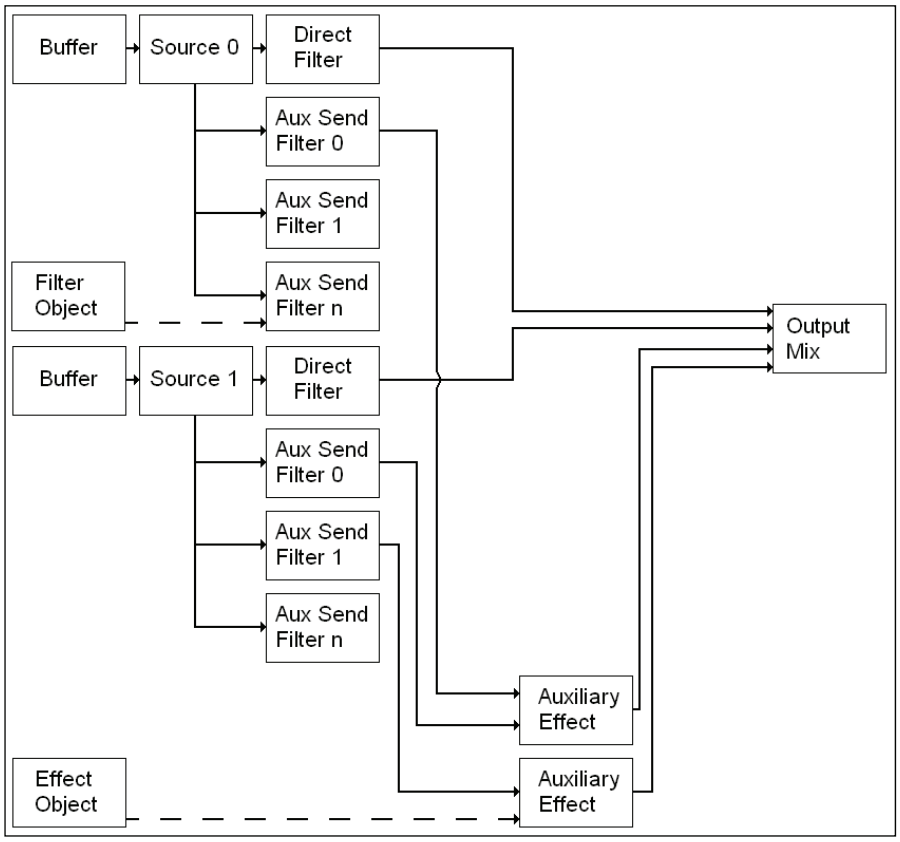
- **Basic OpenAL architecture**



  - Source의 각종 속성들을 모두 합쳐 3D 가상화 기법을 적용하여 실제 상황을 최대한 시뮬레이션 한다.
- **Interactive 3D Audio 처리의 두 가지 기법**
  - Per-source environmental filtering (occlusion, obstruction)
  - Auxiliary effect sends (e.g. environmental reverb)

Effects Extension Architecture

- **OpenAL with Effects Extension Architecture**



Effects Extension Architecture(계속)

- "OpenAL with Effects Extension Architecture"의 의미
  - Source는 Filter를 거친 후 재생될 수 있다.
  - 새로 도입된 Auxiliary Effect Sends로 인해 음향 효과 처리가 Source 그룹에 한꺼번에 적용될 수 있음.
  - Auxiliary Effect의 출력이 최종 mix에 들어감
  - Source → Filtering → Auxiliary Effect의 과정을 거치게 함으로써 다양한 3D 음향 효과를 줄 기회가 많다.

Effects Extension Objects

- Effects Extensions으로 인해 OpenAL에는 새로운 객체들이 추가되었음
- Auxiliary Effect Slot Object
  - Source 객체들에 가해질 음향 효과 하나를 위한 객체
  - 음향 효과 유형과 설정은 이 Auxiliary Effect Slot 객체에 덧붙여질 Effect Object에 의해 결정된다.
- Effect Object
  - Auxiliary Effect를 정의하는 데에 필요한 parameter들로 구성된다.
    - Effect 유형(reverb, chorus, etc…)과 그 Effect 제어에 필요한 parameter 값들을 담는다.

Effects Extension Objects(계속)

- **Filter Object**
  - Filter 설정에 필요한 정보를 담는 객체
  - Filter type(low-pass, high-pass 등)과, filter amount나 cut-off 같은 설정 값들을 갖는다.
  - Filter 객체의 용도
    - to filter the direct path (dry signal) of a Source
    - to filter the send path (wet signal) to any of the Auxiliary Effect Slots
- **Source, Listener, Context 객체 확장**
  - Effects Extension은 기존 OpenAL 객체들의 속성에 새로운 속성들을 추가함.

Auxiliary Effects Slots Object

- **음향 효과 저장소로써**
  - 음향 효과 처리 과정의 최말단에 위치
  - 여러 개의 Source가 같은 aux. effect slot에 연결될 수 있음
    - But, 한 source가 연결될 수 있는 aux. effect slot의 개수는 제한
- **Aux. Effect Slot 생성/제거 함수**
  - alGenAuxiliaryEffectSlots() / alDeleteAuxiliaryEffectSlots()
- **Auxiliary Effect Slot에 Effect Object를 붙이면 음향효과가 적재되었다고 함**
  - Effect Object는 음향 효과의 유형과 관련 parameter 값들을 담는 객체임
  - Device에 따라 지원되는 음향 효과가 다르므로 모든 음향 효과 유형이 aux. effect slot에 적재가능 하지 않을 수 있음.
  - 이미 적재된 effect object는 그 설정 값을 바꿔도 효과는 바뀌지 않으므로, 이럴 경우 empty effect object를 적재했다가 다시 effect object를 적재해야 함.

Effects Object

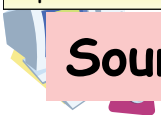
- 음향 효과 유형과 **parameter** 값들을 담는 객체
  - 여러 음향 효과 parameter 값들을 미리 설정해서 담아두는 'presets' 저장소로 사용됨
    - 화장실 환경 모델링 parameter 값들을 모아 놓은 reverb effect 객체
    - 동굴 환경 모델링 parameter 값들을 모아 놓은 reverb effect 객체,
    - 계곡 모델링 parameter 값들을 모아 놓은 echo effect 객체 등
- **Effect Object** 생성/제거 함수
  - alGenEffects() / alDeleteEffects()
- **Effect** 객체가 반영되기 까지의 과정
  - Auxiliary Effect Slot에 적재되어야 함
  - 일단 어떤 effect slot에 적재되고 나면, 그 effect slot으로 들어오도록 설정되어 있는 source들은 이 effect 객체로부터 자동으로 parameter 값들을 제공받는다.

Filters Object

- 필터 유형과 관련 **parameter** 값들을 담는 객체로써
  - 여러 filter parameter 값들을 미리 설정해서 담아두는 filter 'presets' 저장소로 사용됨
    - 콘크리트 벽을 통과하는 소리 모델링에 필요한 parameter를 담고 있는 low-pass filter
    - 전화 목소리 모델링에 필요한 parameter를 담고 있는 band-pass filter
- **Filter Object** 생성/제거 함수
  - alGenFilters() / alDeleteFilters()
- Filter object가 사용되는 두 가지 방법
  - OpenAL Source를 직접 filtering 하기
    - Filter 객체가 source에 직접 붙으면(이 경우의 filter를 Direct Filter 또는 Dry Filter라 한다), 이 filter는 그 source가 직접 재생될 때에만 적용된다.
  - OpenAL Source에서 Auxiliary Effect Slot에 이르는 중간 send들을 filtering 하기
    - Filter 객체가 Auxiliary Send Filter 자격으로 Source에 붙으면, 이 filter는 그 Auxiliary Effect Slot. 으로 가는 모든 signal에 적용된다.

Source Extensions for Effects Extensions

- **Source에 추가된 기능**
  - Source가 Filter 객체와 Effect 객체, Auxiliary Effect Slot 객체를 사용할 수 있게
  - OpenAL의 3D 공간 모델 향상
- **Enabling/Disable a Source Auxiliary Send**
  - Aux. Effect Slot에 적재된 Effect 객체에 Source가 데이터를 흘려 보낼 수 있으려면 Source의 auxiliary send(데이터 출구)를 설정해야만 한다.
    - Source의 aux. send를 활성화하려면 destination Auxiliary Effect Slot ID와 Auxiliary Send 번호, Filter ID(옵션)를 알아야 한다.
  - 반대로 Source의 aux. send를 비활성화하려면 그 send를 NULL Auxiliary Effect Slot으로 보내면 된다.



Source Extensions for Effects Extensions (계속)

- **Enabling/Disable a Source Filter**
  - 한 source의 direct-path (dry signal)에 filtering을 가하려면 Filter 객체를 그 source에 붙여야 한다.
  - source에 가하던 filtering을 해제하려면 NULL Filter 객체를 붙이면 된다.
- **추가된 3D Spatialization Modeling Properties**
  - 공기에 의한 소리 감쇠 정도를 조절할 수 있는 기능이 source 속성에 추가됨
  - Cone parameter 사용 시 source가 listener로부터 멀어질 때 적용할 low-pass filter를 제어할 수 있는 기능이 source 속성에 추가됨.
  - Source의 aux. send level에 적용할 소리 감쇠의 정도를 조절할 수 있게 함
  - Source의 direct-path에서 고주파 소리의 감쇠 수준 조절 기능
  - source-listener 거리와 source 방향에 기반한 반사-소리 감쇠 수준 조절 기능
  - source 방향에 기반한 반사-소리의 고주파 부분 감쇠 수준 조절

Listener Extensions for Effects Extensions

- **OpenAL Listener** 객체에는 딱 하나의 속성이 추가됨
  - "응용이 Effects Extension이 사용할 거리 단위 정보를 설정할 수 있게 하여 Air Absorption 같은 기능이 올바로 작동하도록" 하였다.
  - 응용이 사용할 거리 단위 alListenerf()에 인자로 AL_METERS_PER_UNIT를 주고 호출하면 된다.
    - 응용이 거리 단위로 센티미터를 사용하고자 한다면 단위 값으로 0.01을 줘야 할 것임.
    - 그래야 적용될 air absorption 수준이 100배가 되지 않을 것이다.

Context Extensions for Effects Extensions

- **Context** 객체에 추가된 속성 **1**: 각 **OpenAL source**마다 허용 가능한 **Auxiliary Send**의 개수
  - Context 객체 생성 시 각 source에 허용하기 원하는 Auxiliary Send의 최대 개수를 지정할 수 있음
  - 허용하기 원하는 aux. send의 최대 개수를 지정했다고 그 개수 만큼 사용 가능한 것은 아니므로 응용은 사용 가능한 개수를 알아보고(alcGetIntergerv()) 프로그래밍 해야 함.
- **Context** 객체에 추가된 기능 **2**: **OpenAL**이 지원하는 현재의 **Effects Extension** 버전 알아보기 기능
  - alcGetIntegerv()에 ALC_EFX_MAJOR_VERSION과 ALC_EFX_MINOR_VERSION 속성을 주면 Effect Extension의 버전을 알아볼 수 있음.

Programming the Effects Extension

- **Tutorial 1: OpenAL과 Effects Extension 초기화**

```
ALCdevice *pDevice = NULL;
ALCcontext *pContext = NULL;
ALint attribs[4] = { 0 };
ALCint iSends = 0;

pDevice = alcOpenDevice(NULL);        // 디폴트 OpenAL device를 오픈한다.
if (!pDevice) return;

// Effect Extension 기능이 지원되는 장치인지 확인한다.
if (alcIsExtensionPresent(pDevice, "ALC_EXT_EFX") == AL_FALSE) return;
printf("EFX Extension found!\n");

// Context 생성. 생성하면서 source 당 최대 4개의 Auxiliary Send를 허용하겠다고 설정함.
attribs[0] = ALC_MAX_AUXILIARY_SENDS;
attribs[1] = 4;
pContext = alcCreateContext(pDevice, attribs);
if (!pContext) return;
……
```


Tutorial 1: Initializing OpenAL and the Effects Extension(계속)

```
……
alcMakeContextCurrent(pContext);     // 생성한 context를 활성화

// 각 Source 당 실제 사용 가능한 Aux. Send 개수를 얻는다.
alcGetIntegerv(pDevice, ALC_MAX_AUXILIARY_SENDS, 1, &iSends);
printf("Device supports %d Aux Sends per Source\n", iSends);

// Effect Extension 함수 포인터를 얻는다.
alGenEffects = (LPALGENEFFECTS) alGetProcAddress("alGenEffects");
alDeleteEffects = (LPALDELETEEFFECTS) alGetProcAddress("alDeleteEffects");
alIsEffect = (LPALISEFFECT) alGetProcAddress("alIsEffect");
……          // 필요한 모든 Effect Extension 함수들의 포인터를 얻는다.
// 얻은 function pointer들이 유효한지 검사한다.
if (!(alGenEffects && alDeleteEffects && alIsEffect)) return;

// EFX 초기화가 완료되어 이제부터 EFX 프로그래밍이 가능해졌음!!!!
```

Tutorial 2: Auxiliary Effect Slots 객체, Effects 객체, Filters 객체 생성하기

```
ALuint uiEffectSlot[4] = { 0 };
ALuint uiEffect[2] = { 0 };
ALuint uiFilter[1] = { 0 };
ALuint uiLoop;

// 4개의 Auxiliary Effect Slot 객체 생성을 시도함.
alGetError();
for (uiLoop = 0; uiLoop < 4; uiLoop++) {
   alGenAuxiliaryEffectSlots(1, &uiEffectSlot[uiLoop]);
   if (alGetError() != AL_NO_ERROR)
      break;
}
printf("Generated %d Aux Effect Slots\n", uiLoop);

// 2개의 Effect 객체 생성을 시도함
for (uiLoop = 0; uiLoop < 2; uiLoop++) {
   alGenEffects(1, &uiEffect[uiLoop]);
   if (alGetError() != AL_NO_ERROR)
      break;
}
printf("Generated %d Effects\n", uiLoop);
```


Tutorial 2: Auxiliary Effect Slots 객체, Effects 객체, Filters 객체 생성하기 (계속)

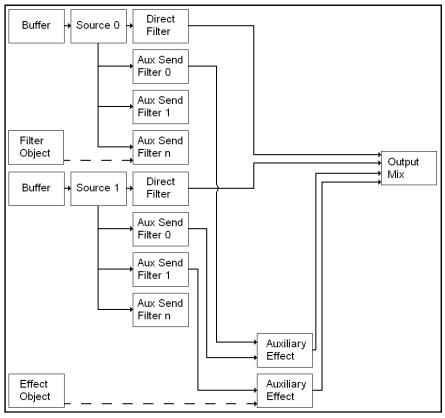
```
// 첫 번째 Effect 객체의 Type을 Reverb로 설정하고, Decay Time을 수정한다.
alGetError();
if (alIsEffect(uiEffect[0])) {
   alEffecti(uiEffect[0], AL_EFFECT_TYPE, AL_EFFECT_REVERB);
   if (alGetError() != AL_NO_ERROR)
      printf("Reverb Effect not supported\n");
   else
      alEffectf(uiEffect[0], AL_REVERB_DECAY_TIME, 5.0f);
}

// 두 번째 Effect 객체의 Type을 Flanger로 하고, Phase를 수정한다.
alGetError();
if (alIsEffect(uiEffect[1])) {
   alEffecti(uiEffect[1],AL_EFFECT_TYPE,AL_EFFECT_FLANGER);
   if (alGetError() != AL_NO_ERROR)
      printf("Flanger effect not support\n");
   else
      alEffecti(uiEffect[1], AL_FLANGER_PHASE, 180);
}
```

Tutorial 2: Auxiliary Effect Slots 객체, Effects 객체, Filters 객체 생성하기 (계속)

```
// Filter 객체 한 개를 생성한다.
alGetError();
alGenFilters(1, &uiFilter[0]);
if (alGetError() == AL_NO_ERROR)
   printf("Generated a Filter\n");
if (alIsFilter(uiFilter[0])) {
   // 생성한 Filter의 유형을 Low-Pass로 하고 원하는 parameter들을 설정한다.
   alFilteri(uiFilter[0],AL_FILTER_TYPE,AL_FILTER_LOWPASS);
   if (alGetError() != AL_NO_ERROR)
      printf("Low Pass Filter not supported\n");
   else {
      alFilterf(uiFilter[0], AL_LOWPASS_GAIN, 0.5f);
      alFilterf(uiFilter[0], AL_LOWPASS_GAINHF, 0.5f);
   }
}
```


Tutorial 3: Effect 객체를 Auxiliary Effect Slot 객체에 붙이기

```
// Attach Effect to Auxiliary Effect Slot

// uiEffectSlot[0]이 한 Aux. Effect Slot의 ID를 담고 있다고 가정
// uiEffect[0]은 한 Effect의 ID를 담고 있다고 가정
alAuxiliaryEffectSloti(uiEffectSlot[0], AL_EFFECTSLOT_EFFECT, uiEffect[0]);
if (alGetError() == AL_NO_ERROR)
   printf("Successfully loaded effect into effect slot\n");
```



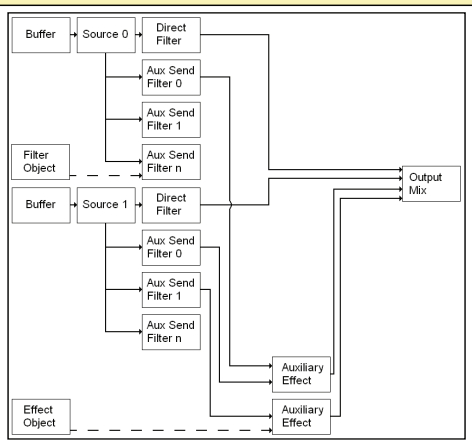
Tutorial 4: Configuring Source Auxiliary Sends

- **Source**에 있는 **Aux. Send Filter**가 **Aux. Effect Slot**으로 연결되도록 하여 데이터가 흘러가게 하는 방법

```
/*
 * Configure Source Auxiliary Effect Slot Sends
 */
// uiEffectSlot[0] and uiEffectSlot[1] are Auxiliary Effect Slot IDs
// uiEffect[0] is an Effect ID, uiFilter[0] is a Filter ID
// uiSource is a Source ID

// Set Source Send 0 to feed uiEffectSlot[0] without filtering
alSource3i(uiSource, AL_AUXILIARY_SEND_FILTER, uiEffectSlot[0], 0, NULL);
if (alGetError() != AL_NO_ERROR)
   printf("Failed to configure Source Send 0\n");

// Set Source Send 1 to feed uiEffectSlot[1] with filter uiFilter[0]
alSource3i(uiSource, AL_AUXILIARY_SEND_FILTER, uiEffectSlot[1], 1, uiFilter[0]);
if (alGetError() != AL_NO_ERROR)
   printf("Failed to configure Source Send 1\n");
```


Tutorial 4: Configuring Source Auxiliary Sends (계속)

```
// Disable Send 0
alSource3i(uiSource, AL_AUXILIARY_SEND_FILTER, AL_EFFECTSLOT_NULL, 0, NULL);
if (alGetError() != AL_NO_ERROR)
   printf("Failed to disable Source Send 0\n");

// Disable Send 1
alSource3i(uiSource, AL_AUXILIARY_SEND_FILTER, AL_EFFECTSLOT_NULL, 1, NULL);
if (alGetError() != AL_NO_ERROR)
   printf("Failed to disable Source Send 1\n");
```



Tutorial 5: Source에 Filter 객체 붙이기

- **Source에 Filter 객체를 연결하는 방법**
- **Filter를 이용해 direct signal(dry path)과 send signal (wet path)를 필터링하는 방법**

```
// 'uiSource ' 는 필터링 할 source ID
alSourcei(uiSource, AL_DIRECT_FILTER, uiFilter[0]);
if (alGetError() == AL_NO_ERROR) {
   printf("Successfully applied a direct path filter\n");
   // Remove Filter from 'uiSource'
   alSourcei(uiSource, AL_DIRECT_FILTER, AL_FILTER_NULL);
   if (alGetError() == AL_NO_ERROR)
      printf("Successfully removed direct filter\n");
}
// Source 'uiSource'에서 Auxiliary Effect Slot uiEffectSlot[0]로 나가게 설정되어 있는 Send를
//  Filter uiFilter[0]를 이용해 필터링 한다.
alSource3i(uiSource, AL_AUXILIARY_SEND_FILTER, uiEffectSlot[0], 0, uiFilter[0]);
if (alGetError() == AL_NO_ERROR) {
   printf("Successfully applied aux send filter\n");
   // Remove Filter from Source Auxiliary Send
   alSource3i(uiSource, AL_AUXILIARY_SEND_FILTER, uiEffectSlot[0], 0, AL_FILTER_NULL);
   if (alGetError() == AL_NO_ERROR)
      printf("Successfully removed filter\n");
}
```


Tutorial 6,7: Source, Listener Properties

- **How to set Effects Extension specific Source/Listener properties**

```
// Set Source Cone Outer Gain HF value
alSourcef(uiSource, AL_CONE_OUTER_GAINHF, 0.5f);
if (alGetError() == AL_NO_ERROR)
   printf("Successfully set cone outside gain filter\n");

// Set distance units to be in feet
alListenerf(AL_METERS_PER_UNIT, 0.3f);
if (alGetError() == AL_NO_ERROR)
   printf("Successfully set distance units\n");
```

Environmental Audio Programming Techniques

- **Creating a single environment world with the Effects Extension**
  - 하나의 aux. effect slot만을 사용한 단일 환경 구성은 간단함.
  - 같은 음향 효과를 적용 받는 영역은 sound designer가 설계
  - Obstruction 검사
    - using ray-casting to determine if any objects are in between the Source and the Listener
  - Occlusion 검사
    - whenever the Source and Listener are in different environments with no line-of-sight between them
  - Exclusion 검사
    - when the Source and Listener are in different environments but there is a line-of-sight between them
  - Source나 Listener의 parameter 값이 동적으로 바뀌면 interactive하게 갱신해주기만 하면 됨

Creating a multi-environment world with the Effects Extension

- 다중 음향 환경을 구축하고 Dynamic하게 재생하는 것은 쉽지 않음
  - Listener의 주변 소리 환경을 정확히 전달해야 함
  - Listener 주변의 여러 소리 환경 중 가장 잘 들려야 하는 환경 순으로 들리게 해야 함.
  - 각 Source가 의도한 aux. effect slot으로 정확히 데이터가 흘러가야 함
  - 특히 게임에서 사용된다면 각 게임 사용자 별로 처한 상황에 맞게 음향 효과가 따로 따로 제공되어야 한다.
  - 각 음향 환경들은 각 Source와 Listener 쌍에 대해 occlusion, exclusion, obstruction의 상태를 확인하여 Filter 효과를 시시각각 갱신해야 할 것임.

다중 음향환경 구축 시 적용되는 원리들

- **Environmental Zones**
  - 가상 공간에서 서로 같은 음향 효과가 적용되는 영역
- **Apertures between environmental zones**
  - 서로 다른 env. zone 간에 door나 portal 같은 연결 통로가 있는가 ?
- **Source to listener direct path**
  - Source와 Listener 간에 소리가 직접 들리는 경로가 있는가?
- **Low-detail models and shared systems**
  - 게임 엔진에 포함되어 있는 지형,지물 감지 엔진 등을 활용할 수 있는가?
  - AI, collision detection and physics system 등이 지형지물 계산 시 음향 효과 계산을 같이 하도록 하면 성능이 향상될 것임

Environmental Zones

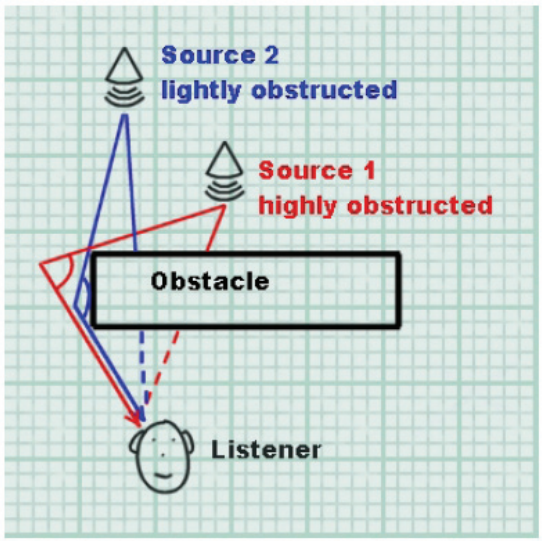
- 가상 공간에서 서로 같은 음향 효과가 적용되는 영역
  - Source와 Listener의 위치는 일련의 좌표로 표현 가능
  - 하나의 env. zone이 규명되면 env. zone ID가 부여되고, 각 zone 별로 고유한 음향효과 parameter들이 설정됨
  - 각 env. zone 별로 음향효과 parameter 값을 알아내는 일은 sound designer가 해야 할 일임.
- 코딩 시나리오 예
  - 어떤 Source가 Listener와 다른 zone에 있게 되었음이 감지되면, 약간의 occlusion 필터링을 실시할 수도...
  - 소리가 서로 다른 zone 간을 이동할 때 occlusion 필터링을 얼마나 강하게(약하게) 줄 것이냐는 동적으로 계산되어야 함.

Apertures between environmental zones

- 다중 음향 환경에서 **exclusion** 효과가 필요한가?
  - env. zone 간에 'apertures' 또는 'portals'이 있는지가 관건
  - aperture: 두 env. zone 사이에 존재하는 문이나 창문 같이 공기가 통하는 열린 공간
  - 어느 한 환경에서 발생하는 반향 효과는 aperture를 통해 Listener가 있는 다른 환경에서도 들려야 함
  - 서로 다른 두 음향 환경 사이에 aperture가 없다면 occlusion 필터링을 사용해야 할 것임

Source to listener direct path

- **Source**와 **Listener** 사이의 직선 구간에 장애물이 있는가?
  - 장애물이 있으면 Obstruction 필터링을 해야 함.
- **Variable amounts of obstruction**



- 장애물에 가려진 **Source**와 **Listener** 사이의 직선이 장애물의 끝부분에 가까울수록 소리 감쇠는 적게 발생한다.
  - Source와 장애물 끝부분이 이루는 각도가 작을수록 Obstruction 효과는 커야 한다.

Multi-environment run-time management algorithm

- 동적 음향 환경 효과 관리 알고리즘 예
  - 알고리즘에서 *n*은 동시에 생성 가능한 음향 효과 환경의 개수를 의미(예를 들면, 4 on Sound Blaster X-Fi, 2 on Sound Blaster Audigy)

각 오디오 프레임에 대해:
Step 1: Update Environments

If (listener 위치가 변했으면) {
  Listener와 가까운 순으로 n-1 개의 음향 환경을 찾는다(물론 listener가 놓여 있는 음향 환경은 항상 소리를 내고 있어야 한다):
    • 방법 1: 각 음향 환경 zone의 지름 같은 특성에 기반한 방법
    • 방법 2: Listener와 음향 환경 중심과의 거리에 기반한 방법
    • 방법 3: 음향 환경과 가장 가까운 aperture에 기반한 방법
  각 음향 환경을 위한 Aux. Effect Slot 설정 값을 변경한다:
    • 찾은 n개의 음향 환경 중 소리를 내지 않아도 되는 Auxiliary Effect slot이 있으면, Auxiliary slot gain을 0으로(silence) 하고, 각 음향 환경에 맞는 Reverb 세팅을 적재한다.
    • 각 Auxiliary Effect slot에 대해, Listener의 방향과 Listener가 환경과 떨어져 있는 거리 등을 고려하여 적절한 Pan과 Reverb gain level을 설정한다.
}

Multi-environment run-time management algorithm (계속)

Else if (listener의 방향(orientation)이 바뀌었으면) {
  각 auxiliary effect slot에 대해:
    • Listener의 방향과 Listener가 환경과 떨어져 있는 거리 등을 고려하여 적절한 Pan과 Reverb gain level을 설정한다.
}

Step 2: Update sources
For every source:

If (source의 환경이 변했거나 || listener 위치가 변했으면) {
  If (source와 listener가 같은 환경에 놓여 있으면) {
    • Listener 환경을 담당하는 aux. effect slot으로 향하게 끔 source send 0을 활성화한다. 이 때 같은 환경에 있으므로 filtering은 필요 없음.
    • source send 1은 deactivate 한다.
    • 필요하다면 obstruction을 위해 listener와 source 간 direct path에 필터링을 한다.
  } Else {
    • Listener 환경을 담당하는 aux. effect slot으로 향하게 끔 source send 0을 활성화한다. 이 때 occlusion을 위한 filtering을 추가한다.
    • If (source 환경에서 나고 있는 소리를 내야 한다면)
        source send 1을 활성화하여 source 환경을 담당하는 aux. effect slot으로 향하게 한다.
      Else
        source send 1을 deactivate 한다.
  }
}

모든 aux. effect slot의 gain을 1(최대)로 설정한다.

Q & A