TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Jukka-Pekka Rajala

SIMULAATTORIN ÄÄNENTOISTOKOMPONENTTI

Työn ohjaaja Lehtori Tony Torp
Työn teettäjä Creanex Oy, valvojana DI Markku Pusenius
Tampere 2006

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikka
Ohjelmistotekniikka
Jukka-Pekka Rajala
Simulaattorin äänentoistokomponentti
Tutkintotyö 33 sivua
Työn ohjaaja Lehtori Tony Torp
Työn teettäjä Creanex Oy, valvojana DI Markku Pusenius
Toukokuu 2006
Hakusanat DirectSound, OpenAL, EAX, äänentoisto

TIIVISTELMÄ

Tutkintotyön tarkoituksena oli suunnitella ja toteuttaa yrityksessä käynnissä olevaan simulaattoriprojektiin äänentoistokomponentti. Komponentista on tarkoitus tehdä mahdollisimman helppokäyttöinen, mutta silti tarpeeksi monipuolinen, jotta olisi mahdollista käyttää sitä myös tulevissa projekteissa.

Äänimaailma on koulutussimulaattorissa yksi tärkeä osatekijä, minkä vuoksi sen kehittämiseen haluttiin panostaa.

Komponentin kehityksessä ohjelmointikielenä oli C++ ja kehitysympäristönä toimi *Microsoft Visual Studio .NET 2003*. Lisäksi käytössä olivat *Microsoft DirectX 9.0c SDK* ja *EAX 2.0 Extensions SDK*, joita käytettiin äänimaailman luontiin. Komponentista luotiin dynaamisesti linkitettävä kirjsto, joten se oli selkeästi oma erillinen osansa ja näin helppo liittää eri projekteihin. Kirjaston ensimmäinen versio käyttää DirectSound-rajapintaa ja EAX-rajapintalaajennusta.

Työn tuloksena oli toimiva äänentoistokomponentti, joka otettiin osaksi yrityksessä käynnissä olevaan simulaattoriprojektiin. Komponentin avulla simulaattoriin voidaan luoda helposti realistisen kuuloinen äänimaailma, jota voidaan hyödyntää koulutuskäytössä.

Komponenttia tullaan myöhemmin kehittämään uusien käyttötarpeiden ilmetessä ja käytettävissä olevan ajan niin salliessa. Seuraavana isona osana on tuen lisääminen OpenAL-rajapinnalle.

TAMPERE POLYTECHNIC
Computer Systems Engineering
Software Engineering
Jukka-Pekka Rajala Simulators sound reproduction component
Engineering Thesis 33 pages
Thesis Supervisor Tony Torp (MSc)
Commissioning Company Creanex Oy Supervisor: Markku Pusenius (MSc)
May 2006
Keywords DirectSound, OpenAL, EAX, sound reproduction

ABSTRACT

The purpose of this engineering thesis was to design and create a sound component. The component was developed for the Creanex Oy to create and control sound environment in a training simulator. The component was programmed with C++ and the development environment was *Microsoft Visual Studio .NET 2003 with Microsoft DirectX 9.0c SDK* and *EAX 2.0 Extensions SDK*. For easy usage in other applications, the component was made as a dynamically linked library. The first version is using the DirectSound-component together with EAX-extensions. The result of this engineering thesis was a sound component, which can be used in several applications for creating and controlling sound environments.

SISÄLLYSLUETTELO

LYHENTEET JA KÄYTETYT TERMIT

EAX — Enviromental Audio Extensions on Creative Labsin kehittämä äänirajapintojen laajennus, joka mahdollistaa realistisen kuuloisen ääniympäristön luomisen ja hallinnoimisen.

DirectX — DirectX on kokoelma rajapintoja, jotka on kehitetty helpottamaan peliohjelmointia Windows-käyttöjärjestelmällä. Kokoelma sisältää tällä hetkellä seuraavat rajapinnat: Direct3D (grafiikka), DirectSound (ääni) ja DirectInput (peliohjaimet).

OpenAL — Open Audio Library on äänilaitteiston ohjelmointirajapinta. Rajapinta sisältää useita funktioita, joilla ohjelmoija voi luoda ja hallita äänilähteitä 3D-maailmassa.

GUID — Globally Unique Identifier on lähes uniikki 128-bittinen tunnus, joka luodaan käyttämällä verkkokortin tunnusta, kellonaikaa, päivämäärää ja järjestysnumeroa.

SDK — Software Development Kit on kokoelma funktiorajapintoja, joita käyttämällä ohjelmoija voi luoda sovelluksiinsa erilaisia ominaisuuksia.

TEA — Tiny Encryption Algorithm, on erittäin nopea ja tehokas salausalgoritmi. Sen kehittäjiä ovat David Wheeler ja Roger Needham Cambridgen yliopistosta. Salaus tapahtuu salaamalla 64 -bittisiä osia, 128 -bitin salausavaimella.

DLL — Dynamic Link Library on tiedosto, joka sisältää kokoelman funktioita tai dataa, joita voidaan kutsua ja käyttää Windows -sovelluksissa.

INI-tiedosto — Ini-tiedosto on tekstimuotoinen ohjelman parametrien tallennustiedosto. Tiedosto koostuu lohkoista ja avaimista.

MFC — Microsoft Foundation Classes on Windows-ohjelmointia helpottamaan suunniteltu C++-luokkakirjasto.

IASIG — Interactive Audio Special Interest Group on perustettu kehittämään yhteisiä standardeja eri äänirajapintojen välille.

1 JOHDANTO

Tutkintotyössäni oli tarkoitus suunnitella ja toteuttaa Creanex Oy:lle dll-kirjasto, jota käyttämällä voidaan luoda mahdollisimman realistiselta kuulostava simuloitu äänimaailma. Kirjastolle tehtiin myös MFC-luokkakirjastoa käyttäen testaussovellus, jonka avulla kirjaston toimintaa ja ominaisuuksia kehitettiin ja testattiin. Testaussovelluksen suunnittelussa otettiin myös huomioon se mahdollisuus, että sitä tultaisiin käyttämään sovelluksen äänien kehittämiseen. Aihe soveltui hyvin tutkintotyöksi, koska se oli selkeästi erillinen osa Creanex Oy:llä käynnissä olevassa simulaattoriprojektissa. Myös komponentin suunnitteluun ja toteuttamiseen kulunut aika vastasi hyvin tutkintotyöhön vaadittavaa työmäärää. Aihe tuki myös hyvin ohjelmointitaitojeni kehittämistä, erityisesti DirectX-rajapinnan käytön suhteen.

Kirjaston vaatimukset tarkentuivat, kun erilaisten äänirajapintojen tarjoamat mahdollisuudet saatiin selvitettyä. Lopullisia vaatimuksia olivat äänitiedostojen lataaminen ja konfiguroiminen tekstitiedoston perusteella, mahdollisuus käyttää eri äänirajapintoja ilman, että kirjastoa käyttävää ohjelmaa tarvitsee muokata. Ääniin vaikuttavia ominaisuuksia piti pystyä panemaan päälle ja pois, joko välittömästi tai viivästetysti. EAX-rajapintalaajennuksen piti olla käytettävissä, jos käytettävä laitteisto sitä tukee ja komponentin piti pystyä lataamaan myös salattuja äänitiedostoja. Kryptauksen purkuun käytettiin aikaisemmin tekemääni funktiota, joka perustui TEA-algoritmiin.

Kirjaston rajapintafunktioiden tuli olla helppokäyttöisiä, jotta sitä käyttävät sovellukset pysyisivät äänien käytön osalta mahdollisimman selkeinä. Nyt kehitettyä dll-kirjastoa voidaan käyttää myös tulevissa Creanex Oy:n simulaattoriprojekteissa ja pienin muutoksin myös muissa sovelluksissa.

2 TUTKITUT TEKNIIKAT

2.1 Taustaa

Tietokoneohjelmassa äänimaailman muodostavat minimissään kuuntelija ja yksi äänilähde. Äänilähteitä voi olla useita, mutta kuuntelijoita vain yksi jokaista ohjelmaa kohti. Kuuntelijaa edustaa pelissä oleva pelaaja ja äänilähdettä taas vaikka ase, joka tuottaa ääntä esimerkiksi ammuttaessa. Samanaikaisesti soivien äänien määrää rajoittavat käytettävä laitteisto ja äänirajapinta. Ensimmäisen sukupolven äänirajapinnat mahdollistivat äänien soittamisen vain samassa paikassa, missä kuuntelija oli. Seuraavan sukupolven äänirajapinnat mahdollistivat äänilähteiden sijoittamisen eri paikkaan, missä kuuntelija sijaitsi. Näin saatiin luotua huomattavasti aidommalta kuulostava ympäristö. Nykyään käytössä on laajennuksia, jotka muuttavat soitettavien äänien ominaisuuksia ympäristön mukaan. Mahdollistaen näin entistäkin aidommalta kuulostavan äänimaailman.

2.2 OpenAL

OpenAL on Loki Entertainment Softwaren ja Creative Labsin kehittämä äänirajapinta, joka suunniteltiin helppokäyttöiseksi ja alustasta riippumattomaksi. /1/ Nimi OpenAL on lähtöisin kehittäjien kunnioituksesta OpenGL-rajapintaa kohtaan, ja lisäksi rajapinnan funktioiden toteutuksessa on vaikutteita OpenGL:n ohjelmointikäytännöistä ja -tavoista: OpenAL:n syntaksi muistuttaa OpenGL:ssä käytettävää syntaksia. Tämän ansiosta OpenGL:ää aiemmin käyttänyt ohjelmoitsija hallitsee myös OpenAL:n käytön hyvin pienellä opettelulla.

Koska OpenAL on ensisijaisesti tarkoitettu 3D-äänimaailman luomiseen, ei perinteistä stereoäänimaailmaa suoraan tueta. OpenAL on yhteensopiva IASIG 3D level 1:n ja level 2:n laajennuksien kanssa, jotka määrittelevät äänilähteen suunnan ja etäisyyden vaikuttaman vaimennuksen. Laajennukset määrittelevät myös dopplerilmiön ja ympäristöefektien, kuten heijastuksien, esteiden, johtumisen ja kaikujen käyttäytymisen.

Ohjelmoijalle OpenAL on kokoelma käskyjä, jotka mahdollistavat äänilähteiden ja kuuntelijan paikan määräämisen kolmessa ulottuvuudessa. Nämä yhdistettynä käskyihin, jotka määräävät, kuinka äänilähteet renderöidään lähtöpuskurissa, mahdollistavat laadukkaan äänimaailman luomisen. Käskyjen vaikutus ei ole välttämättä välitön, vaan viive riippuu käytetystä toteutuksesta, mutta yleensä tämä viive ei ole käyttäjän huomattavissa.

2.2.1 Ympäristön alustaminen

Tyypillinen ohjelma, joka käyttää OpenAL-rajapintaa alkaa kutsulla, jolla avataan yhteys äänilaitteeseen, joka käsittelee uloslähtevän äänen ja hoitaa sen soittamisen kytketyssä laitteistossa. Seuraavana on esimerkki kyseisestä koodista.

Ensimmäisellä rivillä avataan yhteys oletuslaitteeseen käyttämällä *alcOpenDevice*-funktiota. Mikäli halutaan käyttää järjestelmän oletusäänilaitetta, funktiolle annetaan parametrina arvo *NULL*. Yhteyden avaamisen jälkeen luodaan ohjelmassa käytettävä konteksti kutsumalla *alcCreateContext*-funktiota. Parametreina funktiolle annetaan osoitin valittuun äänilaitteeseen ja arvo NULL. Lopuksi luotu konteksti valitaan käyttöön ohjelmassa *aclMakeContextCurrent*-funktiolla.

```
ALCdevice* Device = alcOpenDevice( NULL );
ALCcontext* Context;
if ( Device ) {
    Context = alcCreateContext( Device, NULL );
    alcMakeContextCurrent( Context );
}
```

2.2.2 Äänipuskurien luonti

Äänipuskurien luominen olisi hyvä tehdä heti ympäristön alustamisen jälkeen, varsinkin, jos on tiedossa ohjelman tarvitsemien äänipuskurien määrä. Seuraavana on esimerkki äänipuskurin luonnista.

Puskurit luodaan kutsumalla *alGenBuffers*-funktiota, jolle annetaan parametreina *numberOfBuffers*-muuttuja, joka ilmaisee, kuinka monta äänipuskuria halutaan luoda, ja *buffers*-taulukko, joka tulee sisältämään luotujen puskureiden nimet.

```
ALsizei numberOfBuffers = 2;
ALuint buffers[2];
alGenBuffers( numberOfBuffers, buffers );
```

2.2.3 Äänimateriaalin lataaminen

Onnistuneen äänipuskurien luomisen jälkeen seuraavana ohjelmassa yleensä ladataan äänimateriaali ja kopioidaan se luotuihin äänipuskureihin. OpenAL tukee tällä hetkellä äänimateriaalin lataamista wav- ja Ogg vorbis-tiedostomuodoista. Operaatio voidaan jakaa kolmeen vaiheeseen, joista seuraavana ovat esimerkit.

Ensimmäisenä vaiheena on äänimateriaalin lataaminen tiedostosta käyttäen *loadWAVFile*-funktiota. Ennen funktion kutsumista pitää mahdolliset virhetilanteet nollata kutsumalla *alGetError*-funktiota. Tämän jälkeen kutsutaan *loadWAVFile*-funktiota, jolle annetaan parametreina äänitiedoston nimi ja osoittimet muuttujiin, joihin ladattavan äänimateriaalin tiedot sijoitetaan. *If*-lauseessa tarkastetaan, onnistuiko lataus ja ilmoitetaan käyttäjälle mahdollisista virheistä.

```
alGetError();
loadWAVFile( _T("test.wav"), &format, &data, &size, &freq, &loop );
if ( alGetError() != AL_NO_ERROR )
    AfxMessageBox( _T("Error while loading file.");
```

Lataamisen jälkeen on vuorossa äänimateriaalin kopioiminen äänipuskuriin. Ensimmäinen vaihe on virhetilanteen nollaaminen *alGetError*-funktiota kutsumalla. Äänimateriaalin kopioiminen tapahtuu kutsumalla *alBufferData*-funktiota. Funktiolle annetaan parametreina äänipuskuri, johon äänimateriaali kopioidaan, kopioitava äänimateriaali ja tämän tiedot. *If*-lause tarkastaa virhetilanteet ja ilmoittaa niistä käyttäjälle ja vapauttaa varatut resurssit.

```
alGetError();
alBufferData( g_Buffers[0], format, data, size, freq );
if ( (error = alGetError()) != AL_NO_ERROR ) {
    AfxMessageBox( _T("Error while copying data!") );
    alDeleteBuffers( numberOfBuffers, buffers );
}
```

Kopioimisen jälkeen vapautetaan äänimateriaalin varaamat resurssit kutsumalla *unloadWAV*-funktiota. Parametreina funktiolle annetaan äänimateriaali ja äänimateriaalin tiedot.

```
alGetError();
unloadWAV( format, data, size, freq );
if ( (error = alGetError()) != AL_NO_ERROR )
    DisplayALError( _T("Error while unloading file!") );
```

2.2.4 Äänilähteen luominen

Edellä esitetyillä koodiriveillä on luotu yhteys käytettävään äänilaitteeseen, luotu äänipuskureita ja niihin on ladattu äänimateriaalia. Tämä ei kuitenkaan vielä riitä äänien soittamiseen. Jotta soittaminen onnistuu, on ohjelmoijan vielä luotava ainakin yksi äänilähde. Äänilähde edustaa paikkaa, missä siihen liitetty äänipuskuri soi 3D-maailmassa. Äänilähde tarjoaa myös useita funktioita, joiden avulla soitettavan äänen ominaisuuksia voidaan muokata. Esimerkiksi äänenkorkeutta, voimakkuutta, kuuluvuus etäisyyttä ja äänilähteen nopeutta voidaan muuttaa eri rajapintafunktioiden avulla.

Äänilähteen luominen aloitetaan nollaamalla virhetilanteet *alGetError*-funktiolla. Tämän jälkeen kutsutaan *alGenSources*-funktiota, joka saa parametreinaan luotavien äänilähteiden määrän ja taulukon, johon luotavat äänilähteet sijoitetaan.

```
alGetError();
ALuint sources[2];
ALsizei numberOfSources = 2;
alGenSources( numberOfSources, sources );
if ( alGetError() != AL_NO_ERROR)
    AfxMessageBox( _T("Error while creating sources!") );
```

2.2.5 Äänen soittaminen

Kun ohjelmassa halutaan soittaa ääntä, pitää soitavaksi haluttu äänipuskuri liittää johonkin ohjelmassa olevaan äänilähteeseen. Liittäminen tapahtuu kutsumalla *alSourcei*-funktiota. Funktiolle annetaan kolme parametria: äänilähde, johon haluttu äänipuskuri liitetään. *AL_BUFFER*-vakio, joka ilmaisee, että funktiota käytetään äänipuskurin liittämiseen äänilähteeseen ja liitettävä äänipuskuri.

```
alGetError();
alSourcei( sources[0], AL_BUFFER, buffers[0] );
if ( alGetError() != AL_NO_ERROR )
    AfxMessageBox( _T("Error while attaching sound!") );
```

Nyt äänipuskuri on liitetty äänilähteeseen ja soittaminen voidaan aloittaa kutsumalla *alSourcePlay*-funktiota. Parametrina funktiolle annetaan soitettavaksi haluttu äänilähde.

```
alGetError();
alSourcePlay( source[0] );
if ( alGetError() != AL_NO_ERROR )
    AfxMessageBox( _T("Error while playing sound!") );
```

2.2.6 Resurssien vapauttaminen

Kun OpenAL-rajapintaa äänien toistamiseen käyttämä ohjelma on sammumassa, on sen vapautettava mahdollisesti rajapinnan kautta varaamansa resurssit. Edellä esitetyssä tapauksessa varatut resurssit vapautettaisiin seuraavasti: Ensimmäisellä rivillä haetaan osoitin käytettyyn laiteyhteyteen, toisella rivillä haetaan osoitin käytettyyn laitteeseen. Tämän jälkeen nollataan käytetty yhteys ja seuraavaksi tuhotaan ohjelman alussa luotu laiteyhteys ja lopuksi suljetaan yhteys käytettyyn äänilaitteeseen.

```
Context = alcGetCurrentContext();
Device = alcGetContextsDevice( Context );
alcMakeContextCurrent( NULL );
alcDestroyContext( Context );
alcCloseDevice( Device );
```

Kuten esimerkeistä huomaa, on äänien soittaminen OpenAL-rajapintaa käyttäen helppoa. Voidaankin todeta, että rajapinnan vahvuuksia ovat juuri helppokäyttöisyys ja alustasta riippumattomuus. Heikkoutena on tällä hetkellä rajoittunut ajurituki joiltakin valmistajilta ja vielä puutteellinen dokumentaatio.

2.3 DirectSound

DirectSound on Microsoftin kehittämä äänirajapinta Windows-käyttöjärjestelmälle. /2/ DirectSound on tällä hetkellä paljon käytetty äänirajapinta nykyohjelmistoissa. Tämä johtuu suurelta osin hyvästä laitetuesta, koska rajapinta löytyy jokaisesta Windows-käyttöjärjestelmää käyttävästä tietokoneesta. Ensimmäiset DirectSoundin versiot eivät olleet yhtä monipuolisia kuin markkinoilla sillä hetkellä olleet parhaat äänirajapinnat. Kovan kehitystyön ja hyvän laitetuen ansiosta DirectSound on kui-

tenkin tällä hetkellä äänirajapintojen parhaimmistoa niin käytettävyydeltään kuin ominaisuuksiltaankin. Varsinkin Microsoftin hyvä dokumentaatio rajapinnan käytöstä on varmasti auttanut suosion lisääntymistä.

Yhtenä suurena erona OpenAL:ään verrattuna DirectSound tukee efektien lisäämistä ääniin. Efektien ansiosta samaa ääntä voidaan käyttää eri tilanteissa, hieman sitä muokkaamalla. DirectSound tukee tällä hetkellä seuraavia efektejä.

- Chorus
- Compression
- Distortion
- Echo
- Environmental Reverberation
- Flange
- Gargle
- Parametric Equalizer
- Waves Reverberation

Kaikki DirectSoundin tukemat efektit lasketaan ohjelmallisesti, minkä vuoksi niiden käyttäminen lisää ohjelman käyttämää prosessoriaikaa. Nykykoneilla tämä kuitenkin vastaa vain noin 1-2%:n lisäkuormitusta, joten eroa ei varsinaisesti huomaa.

2.3.1 IDirectSound-objektin luonti

Ensimmäinen vaihe DirectSoundin käyttöönotossa on IDirectSound-objektin luominen ja cooperative-tason asettaminen. Cooperative-taso vaikuttaa siihen, miten äänet käyttäytyvät, kun käyttäjä vaihtaa ohjelmasta toiseen. IDirectSound-objekti luodaan kutsumalla *DirectSoundCreate8*-funktiota, joka luo ja alustaa objektin, joka on IDirectSound8-rajapinta. Ensimmäinen funktiolle annettava parametri määrää käytettävän äänilaitteen. Jos halutaan käyttää oletusäänilaitetta, voidaan parametrille antaa arvo *NULL*. Toinen parametri on osoitin *IDirectSound8*-tyyppiseen osoittimeen, jonka funktio asettaa osoittamaan luotuun objektiin. Viimeisen parametrin pitää olla arvoltaan *NULL*.

```
IDirectSound8* pIDirectSound;
if( FAILED(DirectSoundCreate8( NULL, &pIDirectSound, NULL )) ) {
    AfxMessageBox( _T("Failed to create IDirectSound8 object!") );
}
```

Objektin luomisen jälkeen pitää asettaa ohjelman käyttämä cooperative-taso. Taso määrää, kuinka ohjelma toimii muiden yhtä aikaa DirectSoundia käyttävien ohjelmien kanssa. Esimerkissä annettava *DSSCL_EXCLUSIVE*-taso tarkoittaa sitä, että kun ohjelma on aktiivinen, kaikki muut, paitsi ohjelman omat äänet, hiljennetään. Tason asettaminen tapahtuu kutsumalla luodun objektin *SetCooperativeLevel*-metodia. Metodille ensimmäisenä annettava parametri on ohjelman pääikkunan kahva. Toisena parametrina annetaan halutun tason määräävä vakio.

```
if( FAILED(pIDirectSound->SetCooperativeLevel( AfxGetMainWnd()-
>GetSafeHwnd(), DSSCL_EXCLUSIVE )) ) {
    AfxMessageBox(_T("Failed to set Cooperative level!"));
}
```

2.3.2 Ensisijaisen puskurin luonti

IDirectSound-objektin luomisen jälkeen ohjelmaan pitää luoda ensisijainen puskuri (primary buffer) ja määrätä käytettävä formaatti. Tähän puskuriin miksataan kaikki ohjelman soittama äänimateriaali ja tämän puskurin dataa luetaan suoraan käytettävään äänilaitteeseen. Ensisijainen puskuri edustaa DirectSound-ympäristössä kuuntelijaa.

Ensisijainen puskuri luodaan kutsumalla IDirectSound-objektin *CreateSoundBuffer*-metodia. Metodille annettavat parametrit ovat osoite *DSBUFFERDESC*-tietueeseen, *IDirectSoundBuffer*-tyyppisen osoittimen osoite ja arvo *NULL*. Tietue määrää, minkätyyppinen äänipuskuri luodaan ja mitä ominaisuuksia se tukee. Ensisijaista puskuria luotaessa tärkeimmät asetukset ovat *DSBCAPS_PRIMARYBUFFER*- ja *DSBCAPS_CTRL3D*-liput. Ensimmäinen lippu määrää, että luotava äänipuskuri on ensisijainen, ja toinen lippu mahdollistaa 3D-äänien käyttämisen.

```
IDirectSoundBuffer* pDSBPrimary = NULL;

DSBUFFERDESC dsbd;
```

```
// Nollataan tietue
ZeroMemory( &dsbd, sizeof(DSBUFFERDESC) );

// Asetetaan tietueen koko.
dsbd.dwSize        = sizeof(DSBUFFERDESC);

// Asetetaan liput, jotka määräävät, minkätyyppinen äänipuskuri luodaan.
dsbd.dwFlags       = DSBCAPS_PRIMARYBUFFER | DSBCAPS_CTRL3D;

// Asetetaan äänipuskurin koko.
dsbd.dwBufferBytes = 0;

// Asetetaan äänipuskurin formaatti.
dsbd.lpwfxFormat   = NULL;

if( FAILED(pIDirectSound->CreateSoundBuffer( &dsbd, &pDSBPrimary,
NULL )) ) {
    AfxMessageBox( _T("Failed to create primary soundbuffer!") );
}
```

Kun ensisijainen puskuri on luotu, sille täytyy sille asettaa käytettävä formaatti. Tämä tapahtuu kutsumalla puskurin *SetFormat*-metodia. Metodille annetaan parametrina osoite *WAVEFORMATEX*-tietueeseen, joka määrää tarkemmin käytettävän formaatin. Esimerkissä käytettävä äänidata on 16-bittistä monoääntä 44.1kHz:n näytteenottotaajuudella.

```
WAVEFORMATEX wfx;

// Nollataan tietue
ZeroMemory( &wfx, sizeof(WAVEFORMATEX) );

// Asetetaan käytettävä äänidataformaatti.
wfx.wFormatTag      = WAVE_FORMAT_PCM;

// Asetetaan käytettävien kanavien määrä.
wfx.nChannels       = 1;

// Asetetaan käytettävä näytetaajuus.
wfx.nSamplesPerSec  = 44100;

// Asetetaan tieto, kuinka monta bittiä on yhdessä näytteessä.
wfx.wBitsPerSample  = 16;

// Asetetaan tieto, kuinka monta tavua yksi näyte on.
wfx.nBlockAlign     = 2; //(nChannels * wBitsPerSample) / 8

// Asetetaan tieto keskimääräisestä tiedon koosta (tavua/s).
wfx.nAvgBytesPerSec = 88200; // nSamplesPerSec * nBlockAlign

if( FAILED( pDSBPrimary->SetFormat(&wfx) ) ) {
    AfxMessageBox( _T("Failed to set format to primary buffer") );
}
```

2.3.3 Kuuntelijarajapinnan kysyminen

Mikäli ohjelmassa halutaan käyttää kaikkia DirectSoundin tarjoamia ominaisuuksia, on ensisijaiselta puskurilta kysyttävä erillinen kuuntelijarajapinta. Rajapinta tarjoaa kaikki kuuntelijan 3D-äänimaailman määrittämiseen tarvittavat metodit. Rajapinta kysytään kutsumalla ensisijaisen puskurin *QueryInterface*-metodia. Metodille annetaan ensimmäisenä parametrina IDirectSound3DListener8:n määräävä *GUID*-arvo. Toisena parametrina annetaan *IDirectSound3DListener8*-tyyppisen osoittimen osoite.

```
IDirectSound3DListener8* pIDirectSound3DListener;
if(FAILED(pDSBPrimary->QueryInterface( IID_IDirectSound3DListener8,
(void**)&pIDirectSound3DListener ))) {
    AfxMessageBox(_T("Failed to get IDirectSound3DListener inter-
    face"));
}
```

2.3.4 Toissijaisen puskurin luonti

Toissijaista puskuria (secondary buffer) luotaessa ensimmäinen vaihe on *DSBUFFERDESC*-tietueen luominen ja alustaminen halututuilla arvoilla. Tärkeimmät kentät ovat *dwFlags*, *guid3DAlgorithm*, *lpwfxFormat* ja *dwBufferBytes*. *DwFlags*-kenttä määrää luotavalle puskurille tämän tukemat ominaisuudet. Kaikille luotaville puskureille ei kannata asettaa jokaista tukea päälle, koska mitä enemmän ominaisuuksia tuetaan, sitä enemmän puskurin käsittely vaatii laskentatehoa. Lisäksi joitain ominaisuuksia ei voi käyttää yhtä aikaa samassa puskurissa. *Guid3DAlgorithm* määrää algoritmin laadun, jolla äänen 3D-paikka määritetään. Jos puskurille ei ole asetettu 3D-ominaisuutta, kentän pitää saada arvo *NULL*. Ladattavan äänimateriaalin formaattitiedot sijoitetaan *lpwfxFormat*-kenttään, joka on osoitin *WAVEFORMATEX*-tietueeseen. Tieto luotavan äänipuskurin koosta sijoitetaan *dwBufferBytes*-kenttään. Kentän arvo kannattaa asettaa ladattavan äänimateriaalin kokoiseksi.

Esimerkissä esiintyvä *pvRes* on osoitin, joka on alustettu osoittamaan ladattuun äänimateriaaliin. Esimerkissä tiedoston luku on jätetty pois; se voidaan tehdä normaaleilla tiedoston lukurutiineilla, mutta toteutus riippuu käytetystä tiedostomuodosta.

```
// Luodaan tietue.
DSBUFFERDESC dsBD;
```

```
// Nollataan tietue.
ZeroMemory( &dsBD, sizeof(DSBUFFERDESC) );

// Sijoitetaan tietueen koko tietueeseen.
dsBD.dwSize = sizeof(DSBUFFERDESC);

/* Asetetaan luotavalle äänipuskurille seuraavat ominaisuudet: taajuuden muuttaminen, äänenvoimakkuuden muuttaminen, 3D-äänen käyttäminen ja efektien käyttötuki. */
dsBD.dwFlags = DSBCAPS_CTRLFREQUENCY | DSBCAPS_CTRLVOLUME |
DSBCAPS_CTRL3D | DSBCAPS_CTRLFX;

/* Asetetaan tieto kuinka tarkkaa algoritmia käytetään 3D-paikan laskemisessa. */
dsBD.guid3DAlgorithm = DS3DALG_HRTF_LIGHT;

/* Otetaan äänimateriaalin formaattitiedot talteen. pdwFormat on osoitin ladatun äänimateriaalin formaattitietoon. /*
dsBD.lpwfxFormat = reinterpret_cast<LPWAVEFORMATEX>( pdwFormat );

/* Tallennetaan äänimateriaalin alkamisosoite. pdwData on osoitin ladattuun äänimateriaaliin. */
pbWaveData = reinterpret_cast<LPBYTE>( pdwData );

/* Tallennetaan äänimateriaalin koko. (dwLenght muuttujaan on aiemmin asetettu tieto materiaalin koosta.)*/
dsBD.dwBufferBytes = dwLength;
```

Kun tietueen kentät on alustettu halutuiksi, on seuraavana vaiheena uuden puskurin luominen. Puskuri luodaan samalla tavalla, kuin ensisijainen puskuri, eli luomiseen tarvitaan osoitin, joka osoittaa IDirectSound-objektiin. Esimerkissä osoitin on nimeltään *pIDirectSound*.

```
IDirectSoundBuffer* pDsb = NULL;
if( FAILED(pIDirectSound->CreateSoundBuffer(&dsBD, &pDsb, NULL)) )
    AfxMessageBox( _T("Failed to create sound buffer") );
```

2.3.5 Äänimateriaalin lataaminen

Toissijaisen puskurin luomisen jälkeen on seuraavana vaiheena täyttää se ladatulla äänimateriaalilla. Puskurin täyttäminen aloitetaan lukitsemalla puskuri kutsumalla puskurin *Lock*-metodia. Se alustaa puskurin vastaanottamaan materiaalia ja lukitsee sen pois käytöstä. Metodille annetaan ensimmäisenä parametrina lukitsemisen aloituskohta puskurissa. Toisena parametrina annetaan tieto, kuinka monta tavua aloituskohdasta eteenpäin puskuria lukitaan. Kolmas parametri on osoittimen osoite, joka asetetaan osoittamaan puskurin lukitun osion alkua. Neljäs parametri on muuttujan osoite, johon kirjoitetaan lukitun osion koko. Viides ja kuudes parametri eivät ole tarpeen kyseissä esimerkissä, joten niille annetaan arvot *NULL*. Viimeisellä pa-

rametrilla kerrotaan metodille, että koko puskuri halutaan lukita. Sen vuoksi toisen parametrin arvo voidaan asettaa nollaksi.

```
LPVOID lpvWrite;
DWORD dwLenght;
pDsb->Lock(0,0,&lpvWrite,&dwLength,NULL,NULL,DSBLOCK_ENTIREBUFFER )
```

Kun puskuri on lukittu ja osoite puskurin lukitun osion alkuun saatu, voidaan itse äänimateriaali kopioida puskuriin käyttämällä *memcpy*-funktiota. Funktiolle annetaan ensimmäisenä parametrina *Lock*-metodin palauttama lukitun osion osoite. Toisena parametrina annetaan puskurin luomisen yhteydessä alustettu *pbWaveData*-osoitin, joka on asetettu osoittamaan ladattuun äänimateriaaliin. Kolmas parametri on *Lock*-metodin palauttama lukitun osion koko.

```
memcpy( lpvWrite, pbWaveData, dwLength );
```

Materiaalin kopioimisen jälkeen on kutsuttava puskurin *Unlock*-metodia, joka vapauttaa puskurin lukituksen ja sallii näin ollen puskurin käyttämisen. Ensimmäisenä parametrina metodille annetaan *Lock*-metodin palauttama lukitun osion osoite ja toisena parametrina lukitun osion koko. Kolmas ja neljäs parametri eivät ole tässä tapauksessa tarpeellisia.

```
pDsb->Unlock( lpvWrite, dwLength, NULL, 0 );
```

2.3.6 Kehittyneempien rajapintojen hakeminen

Mikäli puskuriin halutaan lisätä vaikka DirectSoundin tarjoamia efektejä, on puskurille haettava IDirectSoundBuffer8-rajapinta. Rajapinta korvaa tavallisen IDirectSoundBuffer-rajapinnan ja näin ollen rajapinnan hakemisen jälkeen alkuperäinen rajapinta voidaan vapauttaa. Uusi rajapinta haetaan kutsumalla vanhan rajapinnan *QueryInterface*-metodia. Metodille annetaan ensimmäisenä parametrina IDirectSoundBuffer8-rajapinnan määrittävä *GUID*-arvo. Toisena parametrina annetaan *IDirectSoundBuffer8*-tyyppisen osoittimen osoite. Kun rajapinta on saatu, voidaan vanha rajapinta vapauttaa. Tämä tapahtuu kutsumalla rajapinnan *Release*-metodia.

```
IDirectSoundBuffer8* pISoundBuffer;
```

```
if( FAILED(pDsb->QueryInterface(IID_IDirectSoundBuffer8, reinter-
pret_cast<void**>(&pISoundBuffer))) ) {
    AfxMessageBox( _T("Failed to get IDirectSound interface");
}
pDsb->Release();
```

2.3.7 Äänen soittaminen

Kun järjestelmään on luotu onnistuneesti ensisijainen ja toissijainen puskuri, ja toissijaiseen puskuriin on ladattu äänimateriaalia, voidaan puskuriin ladattua äänimateriaalia soittaa. Ennen soittamisen aloittamista on kuitenkin tarpeellista tarkistaa, että soitettava puskuri on kunnossa. Tämä tapahtuu kutsumalla puskurin *GetStatus*-metodia. Metodille annetaan parametrina osoitin muuttujaan, johon puskurin tila palautetaan. Metodin kutsumisen jälkeen voidaan sen muuttaman muuttujan avulla testata, onko puskuri vielä käytössä. Tämä tapahtuu tekemällä *AND*-operaatio muuttujan ja *DSBSTATUS_BUFFERLOST*-lipun kanssa. Puskuri on voinut poistua käytöstä, jos käytössä on samanaikaisesti monta DirectSoundia käyttävää ohjelmaa. Tarkistuksen jälkeen voidaan kutsua puskurin *Play*-metodia, jolle annetaan esimerkkitapauksessa kahtena ensimmäisenä parametrina arvo nolla. Viimeisenä parametrina annetaan tieto siitä, että puskurin halutaan soivan toistuvasti.

```
DWORD dwStatus;
pISoundBuffer->GetStatus( &dwStatus );
if ( dwStatus & DSBSTATUS_BUFFERLOST )
    AfxMessageBox( _T("Error: BufferLost!") );
pISoundBuffer->Play( 0, 0, DSBPLAY_LOOPING)
```

2.3.8 Resurssien vapauttaminen

Kun DirectSoundia käyttävä ohjelma sulkeutuu, sen on vapautettava kaikki varaamansa resurssit. Tämä tapahtuu kutsumalla jokaisen haetun rajapinnan *Release*-metodia.

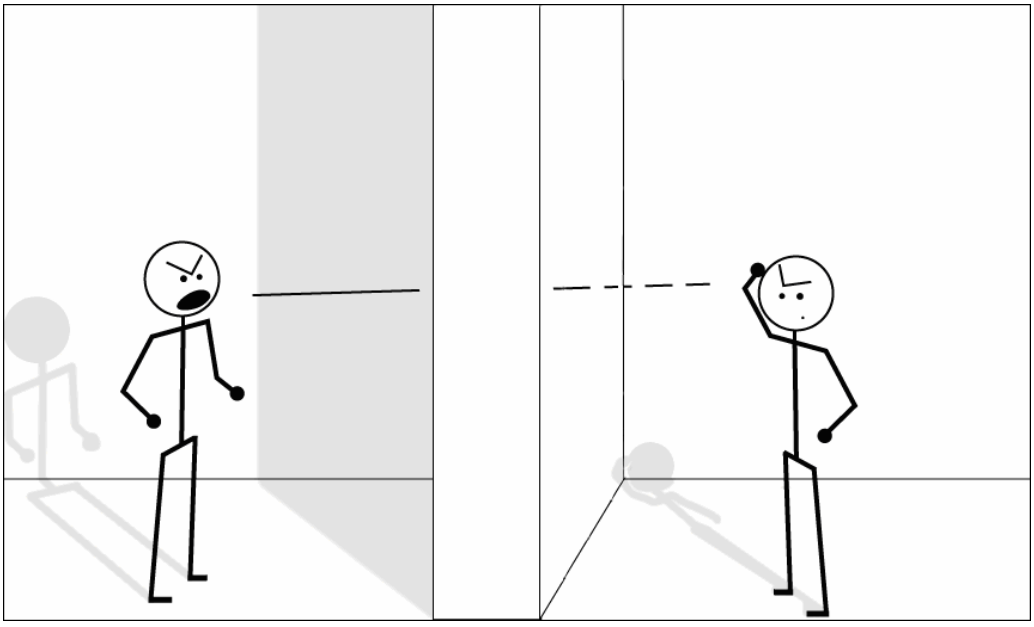
2.4 EAX

EAX eli Enviromental Audio Extensions on Creative Labsin kehittämä äänirajapintalaajennus. /3/ Rajapinnan avulla ohjelmoijat voivat luoda realistisen kuuloisen

äänimaailman lisäämällä normaalien rajapintojen tarjoaman 3D-paikkatiedon päälle ympäristöstä aiheutuvan muutoksen soitettavaan ääneen. Ensimmäinen versio EAX:stä julkaistiin vuonna 1998, uuden SoundBlaster Live!-äänikortin mukana. EAX oli ensimmäinen rajapinta, joka mahdollisti reaaliaikaisten tilaäänien käyttämisen.

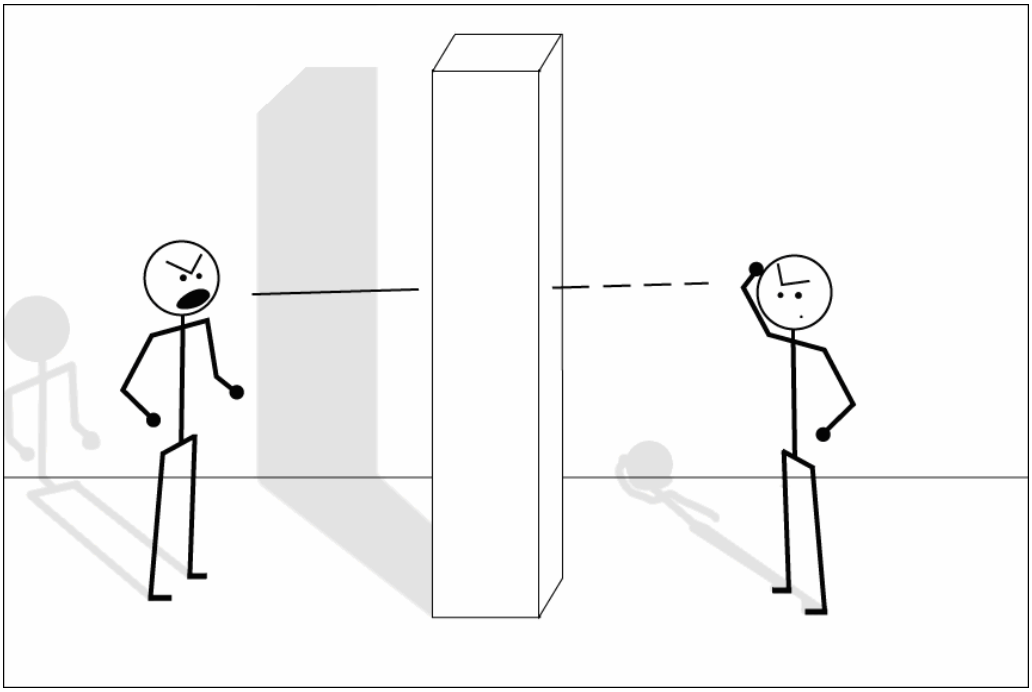
Rajapinnan ensimmäinen versio toi mukanaan 26 esimääritettyä ympäristöä, kuten esimerkiksi kylpyhuone, halli tai luola. Lisäksi rajapinta tarjosi mahdollisuuden muuttaa kaiun parametreja ja näin mahdollisti myös itse määritellyn ympäristön käyttämisen. Ensimmäinen versio mahdollisti vain kahdeksan yhtäaikaisen äänen soittamisen tilavaikutelmassa, mikä oli selvä rajoitus, koska sen hetkinen DirectSound rajapinta mahdollisti jo 32 äänilähteen käyttämisen.

Rajapinnan seuraava versio julkaistiin vuonna 1999 ja se sisälsi huomattavia parannuksia. Samanaikaisten äänien määrä oli muun muassa saatu nostettua 8:sta 32:een. Kaikuun vaikuttavia parametreja lisättiin ja rajapintaan tuotiin automaatiota, joka muutti ääntä sen etäisyyden mukaan mallintamalla näin entistä paremmin suljetuissa tiloissa tapahtuvia äänenmuutoksia. Tärkein uudistus liittyi kuitenkin esteiden aiheuttamaan vaimennukseen. Tämän avulla pystyttiin suoraan hallitsemaan kahta peleissä useasti tarvittavaa tilannetta, joista käytetään nimityksiä *occlusion* ja *obstruction*. Seuraavana ovat havainnollistavat kuvat.



Kuva 1 Occlusion

Kuvassa 1 vasemmanpuoleinen ihmishahmo esittää äänen lähdettä ja oikealla oleva ihmishahmo edustaa kuuntelijaa. Koska hahmot ovat eri tiloissa, vasemmanpuoleisesta hahmosta lähtevästä äänestä muokataan sekä suoraan että heijastumalla tulevia ääniä. Muokkaus tapahtuu tavallisesti ajamalla äänet alipäästösuodattimen läpi, eli korkeita taajuuksia suodatetaan pois. Suotimen parametreihin vaikuttavat tässä tapauksessa esteen paksuus ja materiaali.



Kuva 2 Obstruction

Kuvassa 2 vasemmanpuoleinen ihmishahmo on äänilähde ja oikealla puolella oleva ihmishahmo kuuntelija. Nyt molemmat hahmot ovat samassa tilassa, mutta niillä ei ole näköyhteyttä. Tämä aiheuttaa sen, että vasemmanpuoleisesta hahmosta lähtevästä äänestä muokataan vain suoraan tulevaa ääntä, mutta heijastukset tulevat ilman muokkausta oikeanpuoleisen hahmon luo. Suoraan tulevaa ääntä muokataan taas alipäästösuodattimella, johon vaikuttavat samat parametrit kuin aikaisemmassakin tilanteessa.

Ensimmäisen version tapaan Creative Labs julkisti myös EAX 2.0-version määritelmät ja antoi mahdollisuuden myös muille valmistajille tukea uutta rajapintaa. Osin tämän avoimuuden kautta rajapinnasta tuli keskeinen osa tulevaa IASIG level 2 3D-äänimääritelmää.

Tällä hetkellä uusin versio rajapinnasta on nimeltään EAX Advanced HD 5.0, jonka ominaisuuksiin kuuluu muun muassa tuki 128 yhtäaikaiselle äänelle, mahdollisuus käyttää mikrofonia, johon puhuttu ääni muokataan ääniympäristöön sopivaksi ja mahdollisuus määrätä, mistä kaiuttimesta ääni halutaan soitettavan.

2.4.1 EAX ja OpenAL

EAX sisältää kaksi erilaista ominaisuus joukkoa. Ensimmäinen näistä on kuuntelija ominaisuus joukko, jota käytetään OpenAL:n kuuntelija objektin kanssa ja näin ollen määrittää minkälaisessa ympäristössä kuuntelija sijaitsee. Toinen ominaisuus joukko on nimeltään äänilähde, jota käytetään OpenAL:n lähteiden kanssa ja näin ollen se määrittää, miltä ääni kuulostaa käytetyssä ympäristössä. Kun EAX on käytössä OpenAL ympäristössä, lisää EAX tilavaikutelman käyttämättä OpenAL:n rajapintaa. Tämän ansiosta äänilähteiden alkuperäinen 3D-paikkavaikutelma pysyy muuttumattomana. Seuraavana esimerkki kuinka EAX:ää käytetään OpenAL:n kanssa.

Ensimmäisenä varmistetaan, että laitteisto tukee EAX:ää, ja haetaan osoittimet EAXGet- ja EAXGet-funktioihin. Tuen tarkistaminen tapahtuu kutsumalla *alIsExtensionPresent*-funktiota, jolle annetaan parametrina halutun laajennuksen nimi, joka on tässä tapauksessa EAX. Kun tuki on varmistettu, on seuraavana vuorossa osoittimien hakeminen edellä mainittuihin funktioihin. Tämä tapahtuu kutsumalla *alGetProcAddress*-funktiota, jolle annetaan parametrina halutun funktion nimi. Tässä tapauksessa *EAXGet* ja *EAXSet*.

```
PropSetFunction pfPropSet = NULL;
PropGetFunction pfPropGet = NULL;
if (alIsExtensionPresent( (ALubyte *)"EAX" ) == AL_TRUE ) {
    pfPropSet = alGetProcAddress( (ALubyte *)"EAXSet" );
    pfPropGet = alGetProcAddress( (Alubyte *)"EAXGet" );
}
```

Seuraavana vaiheena on tavallisesti ympäristön määrittäminen. Ympäristön voi määrittää, joko käyttämällä EAX:n mukana tulevia valmiita ympäristöjä, tai vaihtoehtoisesti voi luoda kokonaan uuden ympäristön määrittelemällä jokaisen parametrin itse. Kummassakin vaihtoehdossa ympäristön määrittämiseen käytetään

edellä hankittua osoitinta EAXSet-funktioon, esimerkissä nimeltään *pfPropSet*. Funktion ensimmäinen parametri kertoo, ollaanko muokkaamassa ympäristöä vai äänilähdettä. Tässä tapauksessa muokkauksen kohteena on ympäristö, joten parametri on *PROPSETID_EAX_ListenerProperties*-vakio, joka on määritelty EAX:n mukana tulevassa otsikkotiedostossa. Toinen parametri kertoo, mitä asetuksia ollaan muokkaamassa. Neljäs parametri on osoitin tietoon, jota ollaan lisäämässä ja viimeisenä on tieto siitä, kuinka iso lisättävä tieto on tavuina.

```
unsigned long ulEAXVal;
if ( pfPropSet != NULL ) {
    ulEAXVal = EAX_ENVIRONMENT_BATHROOM;
    pfPropSet( PROPSETID_EAX_ListenerProperties,
    DSPROPERTY_EAXLISTENER_ENVIRONMENT, 0, &ulEAXVal,
    sizeof(unsigned long) );
}
```

Vaihtamalla funktion toisen parametrin arvoa, voidaan samalla tavalla vaihtaa myös yksittäisiä ympäristöarvoja. Tämä on tarpeen, jos halutaan käyttää itse luotua ympäristöä.

Äänilähteiden ominaisuuksien määrittämiseen käytetään samaa funktiota kuin edellä. Tällä kertaa ensimmäisenä parametrina funktiolle annetaan *PROPSETID_EAX_SourceProperties*-vakio, koska muutoksen kohteena on äänilähde. Toisena parametrina annetaan muutettavan ominaisuuden *ID* ja kolmantena parametrina äänilähteen *ID*, johon muutos halutaan kohdistuvan. Neljäntenä parametrina on arvo, joka sijoitetaan muutettavaan parametriin ja viimeisenä on arvoparametrin koko tavuina.

```
long lEAXVal;
if (pfPropSet != NULL) {
    lEAXVal = 0;
    pfPropSet( PROPSETID_EAX_SourceProperties,
    DSPROPERTY_EAXBUFFER_ROOM, 0, &lEAXVal, sizeof(long) );
}
```

Normaalisti ohjelman ajon aikana tulee myös tilanteita, jolloin tarvitaan tietoa, mitkä ovat kyseisellä ajan hetkellä voimassa olevat arvot kuuntelijalla ja äänilähteillä. Arvot saadaan käyttämällä *pfPropGet*-funktiota, johon haettiin osoitin jo aikaisemmassa esimerkissä. Funktiota käytetään täysin samalla tavalla kuin paramet-

ria asetettaessa. Ainoa erona on se, että funktiokutsun jälkeen arvoparametrin arvo on muuttunut käytössä olevaan arvoon. Esimerkissä haetaan kaikki parametriarvot käytössä olevasta ympäristöstä.

```
EAXLISTENERPROPERTIES EAXVal;
if (pfPropGet != NULL) {
    pfPropGet( PROPSETID_EAX_ListenerProperties,
    DSPROPERTY_EAXLISTENER_ALL, 0, &EAXVal,
    sizeof(EAXLISTENERPROPERTIES) );
}
```

2.4.2 EAX ja DirectSound

EAX:n rooli DirectSound-ympäristössä eroaa hieman sen roolista OpenAL-ympäristössä. DirectSound-ympäristössä EAX toimii laajennuksena ja näin ollen sen käyttö tukeutuu DirectSound-rajapintaan. Tästä johtuen sen käyttö DirectSound-ympäristössä eroaa jonkin verran sen käytöstä OpenAL-ympäristössä. Seuraavassa esimerkissä oletetaan, että ohjelmassa on jo aiemmin alustettu DirectSound-ympäristö.

Ensimmäisenä vaiheena on EAX tuen tarkistaminen ja ominaisuus joukon rajapinnan hakeminen kuuntelijalle, joka siis edustaa käytettävää ääniympäristöä. Vaikka kyseessä on kuuntelija ominaisuus joukon rajapinnan hakeminen, niin rajapintaa ei voida hakea ensisijaisesta äänipuskurista, koska DirectSound ei anna ohjelman käyttää ominaisuus joukko rajapintoja suoraan ensisijaisen puskurin kanssa. Sen sijaan rajapinta haetaan jostakin ohjelmassa olevasta toissijaisesta puskurista. Rajapinta saadaan kutsumalla toissijaisen puskurin *Queryinterface*-funktiota, joka esimerkissä on nimeltään *pDsb*. Funktio saa ensimmäisenä parametrina *IID_IKsPropertySet GUID*:n, joka määrää, mitä rajapintaa ollaan hakemassa. Toisena parametrina funktiolle annetaan *IKsPropertySet*-osoitin, johon halutun rajapinnan osoite haetaan.

```
IKsPropertySet* pEAXListener;
if ( FAILED( pDsb->QueryInterface( IID_IKsPropertySet, reinter-
pret_cast<void**>(&pEAXListener) ) ) ) {
    AfxMessageBox(_T("Failed to get Property Set interface!") );
}
```

Seuraavassa vaiheessa kutsutaan *pEAXListener*-osoittimen kautta *QuerySupport*-funktiota. Funktiolle annetaan ensimmäisenä parametrina *GUID*-arvo, joka määrittää, minkä ominaisuus joukon tukea ollaan kysymässä. Toisena parametrina annetaan vakio, joka määrittää, mitä tukea halutun ominaisuus joukon sisältä ollaan kysymässä. Viimeisenä parametrina annetaan osoitin muuttujaan, johon funktio sijoittaa tiedon tuesta.

```
ULONG support = 0;
if (FAILED( pEAXListener->QuerySupport( DSPROP-
SETID_EAX_ListenerProperties, DSPROPERTY_EAXLISTENER_ALLPARAMETERS,
&support) ) ) {
    AfxMessageBox( _T("EAX 2.0 not supported!") );
}
```

Viimeisenä vaiheena on *QuerySupport*-funktiossa käytetyn muuttujan tutkiminen, että saadaan selville tukeeko laitteisto kysyttyä ominaisuus joukkoa. Tämä tapahtuu helpoiten suorittamalla *AND*-operaatio muuttujan ja *KSPROPERTY_SUPPORT-_GET* ja *KSPROPERTY_SUPPORT_SET* lippujen kanssa ja vertaamalla operaatiosta saatua tulosta lippujen *OR*-operaation tuloksen kanssa.

```
if ( (support & (KSPROPERTY_SUPPORT_GET | KSPROPERTY_SUPPORT_SET) )
!= (KSPROPERTY_SUPPORT_GET | KSPROPERTY_SUPPORT_SET) ) {
    AfxMessageBox( _T("EAX 2.0 not supported!"));
}
```

Kun varmuus tuesta on saatu, voidaan aloittaa ääniympäristön määrittäminen. Helpoiten tämä onnistuu käyttämällä EAX:n mukana tulevia ympäristömalleja. Ääniympäristön tilaa muokataan käyttämällä edellä haetun osoittimen kautta *Set*-funktiota. Funktiolle annetaan ensimmäisenä parametrina ominaisuus joukon *GUID*, jonka tilaa ollaan muokkaamassa. Toisena parametrina annetaan vakio, joka määrittää, mitä arvoa ollaan muuttamassa. Tämän parametrin kanssa voidaan myös antaa tieto, toteutetaanko muutos välittömästi vai ei. Kolmas ja neljäs parametri eivät ole käytössä EAX:n yhteydessä. Viidentenä parametrina annetaan haluttu arvo ja viimeisenä parametrina annetaan arvomuuttujan koko tavuina.

```
DWORD envId = EAX_ENVIRONMENT_BATHROOM;
m_pEAXListener->Set(DSPROPSETID_EAX_ListenerProperties, DSPROP-
ERTY_EAXLISTENER_ENVIRONMENT, NULL, 0, &envId, sizeof(DWORD))
```

Yleensä tämän jälkeen muokataan valmista ympäristömallia halutunlaiseksi. Tämä tapahtuisi vain muuttamalla toista parametria, sekä mahdollisesti kahta viimeistä parametria.

Ääniympäristön määrittämisen jälkeen on vuorossa käytettyjen äänilähteiden muokkaaminen. EAX parametrien muokkaamista varten on jokaiselta muokattavalta äänilähteeltä kysyttävä oma ominaisuus joukko rajapinta. Tämä tapahtuu aivan samalla tavalla, kuin rajapinnan hakeminen ääniympäristölle.

```
if ( FAILED( m_pISoundBuffer->QueryInterface( IID_IKsPropertySet,
reinterpret_cast<void**>(&m_pEAXBuffer)) ) ) {
    AfxMessageBox( _T("Failed to get EAX interface") )
}
```

Kun osoitin rajapintaan on haettu, voidaan aloittaa äänilähteen EAX-parametrien muokkaaminen. Tämä tapahtuu kutsumalla osoittimen kautta *Set*-funktiota. Ensimmäisenä parametrina funktiolle annetaan EAX-äänilähde ominaisuus joukon *GUID*. Toisena parametrina on vakio, joka määrittää, mitä äänilähteen EAX-parametria halutaan muuttaa. Kolmas ja neljäs parametri eivät ole käytössä EAX:n yhteydessä. Neljäntenä parametrina annetaan haluttu arvo ja viimeisenä parametrina arvomuuttujan koko tavuina.

```
LONG room = 0;
m_pEAXBuffer->Set( DSPROPSETID_EAX_BufferProperties,
DSPROPERTY_EAXBUFFER_ROOM, NULL, 0, &room, sizeof(LONG) )
```

Get-funktiolla voidaan kysyä äänilähteellä kulloinkin käytössä olevat arvot. Arvoja voidaan kysyä yksitellen tai kaikki kerralla. Esimerkissä kysytään äänilähteeltä suoraan kuuntelijalle tulevan äänen voimakkuutta. Funktiolle annettava ensimmäinen parametri on EAX-äänilähde ominaisuus joukon *GUID*. Toisena parametrina annetaan vakio, joka määrittää, mitä arvoa ollaan kysymässä. Kolmas ja neljäs parametri eivät ole käytössä. Viidentenä parametrina annetaan osoitin muuttujaan, joka saa kysytyn arvon. Kuudes parametri on muuttujan koko ja viimeinen parametri on osoitin muuttujaan, joka saa tiedon, kuinka monta tavua tietoa kirjoitettiin arvomuuttujaan.

```
LONG direct;
```

```
ULONG revsize;
m_pEAXBuffer->Get( DSPROPSETID_EAX_BufferProperties,
DSPROPERTY_EAXBUFFER_DIRECT, NULL, 0, &direct, sizeof(LONG),
&revsize );
```

3 ÄÄNENTOISTO KOMPONENTTI

3.1 Määrittely

Dll-kirjaston toteuttamisen ensimmäinen vaihe oli määritellä, mitä ominaisuuksia kirjaston tulisi sisältää. Perusvaatimuksina olivat tietenkin äänien lataaminen ja soittaminen. Näiden lisäksi määrittelyvaiheessa vaatimuksiin kuuluivat myös seuraavat asiat:

- Kirjastoa piti pystyä käyttämään monisäikeisessä sovelluksessa, eli kirjaston piti olla säiesuojattu
- Kirjasto piti pystyä alustamaan tekstitiedostoa käyttämällä
- Soitettaviin ääniin piti pystyä lisäämään erilaisia efektejä
- Kirjaston piti pystyä luomaan 3D-ääniympäristö
- Kaikkia ääniin liittyviä toimintoja piti pystyä viivästämään. Siten että kaikki asetetut toiminnot suoritetaan vasta pyydettäessä
- Kirjaston rakenne piti olla sellainen, että sen käyttämä äänirajapinta voi vaihtua, ilman että kirjastoa käyttävää sovellusta tarvitsee muokata
- Kirjaston piti mahdollistaa EAX-rajapintalaajennuksen käyttö, mikäli käytettävä laitteisto sitä tukee
- Kirjaston piti kirjoittaa log-tiedostoon alustuksessa tapahtuvista vaiheista ja virhetilanteissa
- Kirjaston piti pystyä lataamaan ääniä normaalin wav-tiedoston lisäksi itse määritellystä salatusta tiedostosta, jolle käytetään laajennetta cra

Määrittelyvaihe ei ollut selvä kokonaisuus projektin alussa, vaan se oli koko projektin keston aikainen prosessi. Tämä johtui siitä, että projektin alkaessa ei ollut tarkkaa tietoa siitä, minkälaisia ominaisuuksia kirjastossa käytettävät tekniikat tukevat. Lisäksi toteutuksen edetessä ja taitojen parantuessa mukaan päätettiin lisätä vaikeammin toteutettavia ominaisuuksia.

3.2 Suunnittelu

3.2.1 Säiesuojaus

Kirjaston piti olla säiesuojattu, koska sitä tullaan käyttämään monisäikeisessä sovelluksessa. Ilman säiesuojausta kirjaston käyttäminen kyseisissä sovelluksissa voisi aiheuttaa tiedon katoamista tai korruptoitumista, koska kirjastoa käyttää sovelluksessa useampi kuin yksi säie. Suojauksen ansiosta, vain yksi säie kerrallaan voi käyttää kirjastossa olevia luokkia. Mikäli useampi säie kutsuu samanaikaisesti kirjaston sisältämiä metodeja, suoritetaan kutsut niiden saapumisjärjestyksessä. Kutsun suorittamisen ajan muut säikeet ovat pysähtyneinä ja odottavat vuoroaan kutsumansa metodin alussa.

3.2.2 Alustaminen

Kirjasto piti pystyä alustamaan käyttökuntoon tekstitiedostoon kirjoitettujen määreiden mukaisesti. Tekstitiedoston muotona käytettiin Windowsista tuttua ini-tiedosto muotoa. Tiedosto piti sisällään seuraavat ääniympäristöön vaikuttavat tiedot:

- Hakemiston, missä ladattavat äänet sijaitsevat
- Ladattavien äänien määrän ja tiedostojen nimet
- Tarkempi kuvaus jokaisesta ladattavasta äänestä
- Tieto, alustetaanko ääni soimaan toistuvasti
- Tieto, mille äänenvoimakkuudelle ääni alustetaan oletusarvoisesti
- Tieto, mille taajuudelle ääni alustetaan oletusarvoisesti
- Tieto, sallitaanko äänen taajuudenvaihto
- Tieto, halutaanko äänen tukevan efektejä
- Tieto, halutaanko äänen tukevan 3D-paikkaa

Tavoitteena oli, että kirjastoa käyttävän sovelluksen sisältämien äänien perusparametreja pystyttäisiin muokkaamaan helposti. Konfigurointi-tiedoston käyttöä tukee myös seikka, että sen avulla pystytään helposti määräämään, mitä ominaisuuksia

eri äänien halutaan tukevan. Tämä auttaa alentamaan kirjaston käytön vaatimaa prosessoriaikaa valitsemalla tuetuiksi vain ne asiat, joita kyseisen äänen käyttäminen vaatii.

3.2.3 Efektit

Kirjaston tukemien efektien pohjalle valittiin, DirectSoundin tukemat efektit. Koska kirjaston piti pystyä vaihtamaan käyttämäänsä äänirajapintaa, ilman että sitä käyttävää sovellusta tarvitsee muokata. Efektien käyttöä varten suunniteltiin oma rajapinta, jonka kautta efektien muokkaaminen tapahtuisi. Kirjaston sisäinen efektien käsittely tapahtui tätä toimintoa varten suunnitellun manageriluokan kautta.

Efektejä varten suunniteltiin abstrakti kantaluokka, josta periytettiin jokaiselle efektityypille oma luokka. Kantaluokan ansiosta jokainen efekti pystyttiin laittamaan päälle yhtä metodia käyttämällä. Tämän lisäksi yhteisen kantaluokan käyttämisen ansiosta manageriluokalle oli mahdollista luoda taulukko, joka sisälsi kaikki äänellä olevat efektit.

3.2.4 3D-ääniympäristö

Suunnitteluvaiheessa päätettiin, että kirjaston ensimmäinen versio ei tue kaikkia 3D-ääniympäristössä muokattavia ominaisuuksia. Tähän päädyttiin siitä syystä, että projektissa, jota varten kirjasto kehitettiin, ei ollut tarvetta kaikille ominaisuuksille. Pois jätetyt ominaisuudet on kuitenkin helppo lisätä myöhemmin kirjaston tukemiin ominaisuuksiin. Tämä vaatii vain metodien lisäämistä kirjaston sisällä oleviin luokkiin. Kirjaston ensimmäiseen versioon tehtiin tuki seuraaville ominaisuuksille:

- Kuuntelijan ja äänilähteen paikan asettaminen 3D-maailmassa
- Kuuntelijan ja äänilähteen paikan hakeminen 3D-maailmasta
- Äänilähteen minimi ja maksimi etäisyyden asettaminen
- Äänilähteen minimi ja maksimi etäisyyden kysyminen
- Kuuntelijan orientaation asettaminen

3.2.5 Tapahtumien viivästäminen

Tuki tapahtumien viivästämiselle oli perusteltua, koska 3D-äänimaailman parametrien vaihtuessa, uuden ympäristöarvojen laskeminen saattaa viedä huomattavasti aikaa. Tästä voi olla haittaa, jos muutoksen tehnyt säie on hyvin aika kriittinen.

Kirjaston sisälle päätettiin luoda myös säie, jota voisi halutessaan käyttää viivästettyjen muutoksen asettamiseksi voimaan. Säie voidaan alustaa joko päivittämään muutokset tietyn ajan välein tai tekemään päivitys erikseen pyydettäessä. Kirjaston sisäisen säikeen prioriteetti voidaan valita sovelluksen tarpeen mukaan, siten että se ei vie suoritusaikaa tärkeämmiltä säikeiltä.

3.2.6 Kirjaston luokkarakenne

Koska kirjasto piti suunnitella siten, että se voisi käyttää eri rajapintoja ääniympäristön luomiseen, oli luokkarakenteen suunnittelu haasteellista. Ongelma ratkaistiin käyttämällä periyttämistä kaikkien sellaisten luokkien yhteydessä, joista piti tehdä erilaisia toteutuksia eri rajapintojen yhteydessä.

Kirjaston sisälle päätettiin toteuttaa yksi manageriluokka, jonka tehtävänä olisi ääniympäristön hallinnoiminen ja peruskäyttöön tarvittavien rajapintafunktioiden toteuttaminen. Luokka olisi myös vastuussa äänien tarvitsemien resurssien varaamisesta ja vapauttamisesta. Näiden tehtävien lisäksi luokan tehtävänä oli toimia ääniympäristössä kuuntelijana, tämä sopi luokalle hyvin, koska se oli muutenkin keskeinen osa käytettävää ääniympäristöä.

Seuraavana suunnittelun alla oli yksittäisen äänen tiedot sisältävä luokka. Koska äänien käsittely poikkeaa eri rajapintojen välillä, oli kyseisen luokan oltava kantaluokka. Kantaluokasta päätettiin tehdä abstrakti, joten se määrittelee vain rajapinnan ja siitä periytyvien luokkien olisi siten pakko tehdä kyseiselle rajapinnalle.

Ääniin lisättävien efektien toteuttamiseksi oli myös efekteille suunniteltava oma kantaluokka. Efektien käyttämisen helpottamiseksi kirjaston sisälle suunniteltiin efektejä varten manageriluokka. Managerin tarkoitus oli luoda helposti käytettävä

rajapinta kirjaston sisälle, joka ei myöskään olisi riippuvainen käytettävästä rajapinnasta.

3.2.7 EAX-tuki

Kirjaston käyttämäksi EAX versioksi valittiin 2.0, koska tätä uudemmat versiot tarvitsevat toimiakseen Creative Labsin valmistaman äänikortin ja tämä olisi liikaa rajoittanut laitteistovaihtoehtoja. Lisäksi uudempien versioiden kehityspakettien saamiseksi olisi rekisteröidyttävä Creative Labsin kehitysyhteisöön.

3.2.8 Tapahtumaraportin tekeminen

Kirjaston haluttiin kirjoittavan logia, kirjaston latautuessa ja äänien latauksen yhteydessä. Logiin kirjautuu tieto sekä tapahtumien onnistumisesta ja virhetilanteesta. Tiedostoon kirjoittaminen tapahtui tätä toimintoa varten kehitetyn dll-kirjaston kautta. Perusidea oli seuraava: sovelluksen pääikkuna alusti kirjaston käyttämään tiettyä tiedostoa, ja kaikki sovelluksen sisältämät osaset käyttivät kirjaston sisältämiä rajapintafunktioita tietojen lisäämiseen logiin. Rakenteen ansiosta login kirjoittaminen vain yhteen tiedostoon on helppoa ilman vaaraa, että esimerkiksi kaksi eri säiettä yrittäisi varata tiedostoa samanaikaisesti.

3.2.9 Tuetut tiedostoformaatit

Kirjaston piti tukea äänien lataamista perinteisestä wav-tiedostosta, mutta myös aikaisemmin yrityksessä kehittämästäni cra-tiedostosta. Cra-tiedosto sisältää normaalia wav-muodossa olevaa ääntä, mutta se on salattu käyttämällä TEA-algoritmia. Tiedostomuoto haluttiin tuetuksi, koska sovelluksessa, jota varten kirjasto kehitettiin, käytetään yrityksen toimesta nauhoitettuja ääniä ja nämä haluttiin suojata kopioinnilta.

Kirjaston tukemiin ääniformaatteihin liittyy muutamia rajoituksia. Sisäisestä toteutuksesta johtuen ladattavan äänitiedoston täytyy olla mono-ääntä. Tämä johtuu suurelta osin siitä, että 3D-äänimaailmassa käytettävien äänien pitää olla tässä muodossa. Toisena rajoitteena on ladattavan äänitiedoston pituus. Mikäli ladattavalla

äänellä halutaan käyttää efektejä, on sen oltava vähintään 150 ms pitkä. Tämä rajoitus johtuu siitä, että DirectSound ei pysty laskemaan tätä lyhyemmälle äänelle efektejä. Ladattava ääni ei saisi myöskään olla liian pitkä. Tämä rajoitus johtuu siitä, että kirjasto ei tue äänien soittamista pienempinä paloina (streaming).

3.3 Toteutus

Kehitysympäristönä kirjaston toteutuksessa toimi *Microsoft Visual Studio .NET 2003* ohjelmisto. Lisäksi käytössä olivat *Microsoft DirectX 9.0c SDK* ja *EAX 2.0 extensions SDK*:t.
Ennen kirjaston toteutusta päätettiin, että kirjaston ensimmäinen versio tukee vain DirectSound-äänirajapintaa. Päätös tehtiin, koska kirjaston toteuttamiselle varattu aika oli rajallinen. Syy, miksi ainoaksi tuetuksi rajapinnaksi valittiin DirectSound eikä OpenAL, oli suurelta osin efektien puuttuminen OpenAL-rajapinnasta.

3.3.1 Säiesuojaus

Suojaus toteutettiin käyttämällä *Win32 SDK*:n sisältämää *Critical Section*-objektia. Objekti on suunniteltu nimenomaan käytettäväksi säiesuojauksen toteuttamiseen, monisäikeisessä sovelluksessa. Säiesuojauksen käyttö päätettiin paketoida erillisen luokan sisälle, jonka ansiosta suojauksesta tuli selvästi erillinen osa kirjastossa. Luokka piti sisällään *Critical Section*-objektin, joka alustettiin luokan rakentajassa. Lisäksi luokan sisällä oli erillinen apuluokka, joka varasi rakentajassaan *Critical Section*:n ja vastaavasti purkajassaan vapautti *Critical Section*:n varauksen. Apuluokka toteutettiin, koska sitä käyttämällä ei ollut vaaraa siitä, että jossain tapauksessa *Critical Section*:n varaus jäisi vapauttamatta.

Koska suunnitteluvaiheessa päätettiin, että vain yksi säie kerrallaan voisi käyttää kirjaston sisäisiä luokkia, luokkaa joka sisälsi *Critical Section*:n, piti päästä kutsumaan jokaisesta kirjastosta ulospäin näkyvästä metodista. Ongelma ratkaistiin antamalla jokaiselle kirjastossa olevalle luokalle, osoitin luokkaan, joka sisälsi *Critical Section*:n. Tämän ansiosta jokainen kirjaston sisältämä metodi varasi saman *Critical Section*-objektin, josta johtuen vain yksi säie kerrallaan pystyi käyttämään kirjaston tarjoamia metodeja.

3.3.2 Alustus tekstitiedostolla

Tekstitiedoston formaatiksi valittiin Windowsin asetustiedostoista tuttu ini-formaatti. Valinta tehtiin, koska tietojen lukemiseen tästä formaatista on olemassa valmiit funktiot. Lisäksi ini-formaatissa olevaa tiedostoa on helppo muokata millä tahansa tekstieditorilla ja formaatin tarjoamat ominaisuudet riittivät hyvin kirjastossa käytettäviin ominaisuuksiin. Seuraavana esimerkki käytettävästä tiedostosta:

```
// Hakemisto missä äänet sijaitsevat
[General]
SoundDirectory = D:\Temp

// Ladattavien äänien määrä ja nimet
[SimulationSounds]
NumberOfSounds = 1
Sound_1 = Testi

// Yksittäisen äänen konfigurointitiedot
[Testi]
FileName = Test
Description = Testi ääni
Continuous = 1
Volume = 100
Frequency = 55000
UseEffects = 1
AllowFrequencyChange = 1
Use3D = 1
```

Tiedot luettiin ini-tiedostosta käyttämällä *GetPrivateProfileString*-funktiota, jolle annetaan parametreina lohkon ja avaimen nimi, arvo joka palautetaan, jos haluttua haluttua avainta ei löydy. Merkkijonotaulukko, johon halutun avaimen sisältämä tieto sijoitetaan ja merkkijonotaulukon koko. Sekä tiedosto, josta lohkoa ja avainta haetaan.

Kirjaston alustamiseksi tekstitiedoston avulla managerille luotiin *ConfigureFromInifile*-metodi, jolle annetaan tekstitiedoston polku, jonka avulla kirjasto alustetaan. Ensimmäisessä vaiheessa metodi lukee annetusta tiedostosta ladattavien äänien määrän ja varaa niitä varten tilaa vektoriin. Seuraavana tiedostosta käydään lukemassa tieto, missä ladattavat äänitiedostot sijaitsevat. Tämän jälkeen käydään lukemassa tiedostosta jokaisen äänen parametrit yksitellen ja ladataan halutut äänet kirjaston sisälle.

Lataamisen etenemisestä kirjoitetaan myös koko ajan tietoa, sovelluksessa käytössä olevaan log-tiedostoon

3.3.3 Efektit

Koska kirjaston efektien pohjaksi oli valittu DirectSound-rajapinnan tarjoamat efektit, toteutettiin kirjaston efektit täysin samanlaisiksi, kuin mitä ne DirectSoundissa ovat.

Efektien abstraktiin kantaluokkaan toteutettiin metodit, joiden avulla voitaisiin siitä periytetyltä luokalta kysyä sen tyyppi. Tätä metodia käyttämällä voitiin olla varmoja, että kirjaston ajon aikana tapahtuvien *dynamic_cast* operaatioiden aikana ei tule poikkeuksia. *Dynamic_cast* operaatioita jouduttiin käyttämään, koska manageri, joka sisälsi äänen käyttämät efektit, talletti vektoriinsa vain efektien kantaluokan osoittimia. Lisäksi luokalle toteutettiin metodit, joiden avulla voitiin efektiltä kysyä, onko se käytössä ja ovatko parametrit muuttuneet.

Efektien päälle laittaminen tapahtui kutsumalla efektejä hallinnoivan managerin metodia ja antamalle sille parametrina efekti, joka haluttiin päälle. Manageri hoiti tämän jälkeen sisäisesti efektin alustamisen ja tilan tallentamisen. Koodi esimerkkinä efektin asettaminen oli seuraavanlainen.

```
// Luodaan gargle-efekti
CSndEffectGargle effectGargle;

// Asetetaan efektin parametrit halutuiksi
effectGargle.m_rateHz = 20;
effectGargle.m_waveShape = DSFXGARGLE_WAVE_TRIANGLE;

// Pyydetään haluttu ääni managerilta ja annetaan luotu efekti äänelle.
soundManager.GetSound( 0 )->pSound->SetEffect( &effectGargle );
```

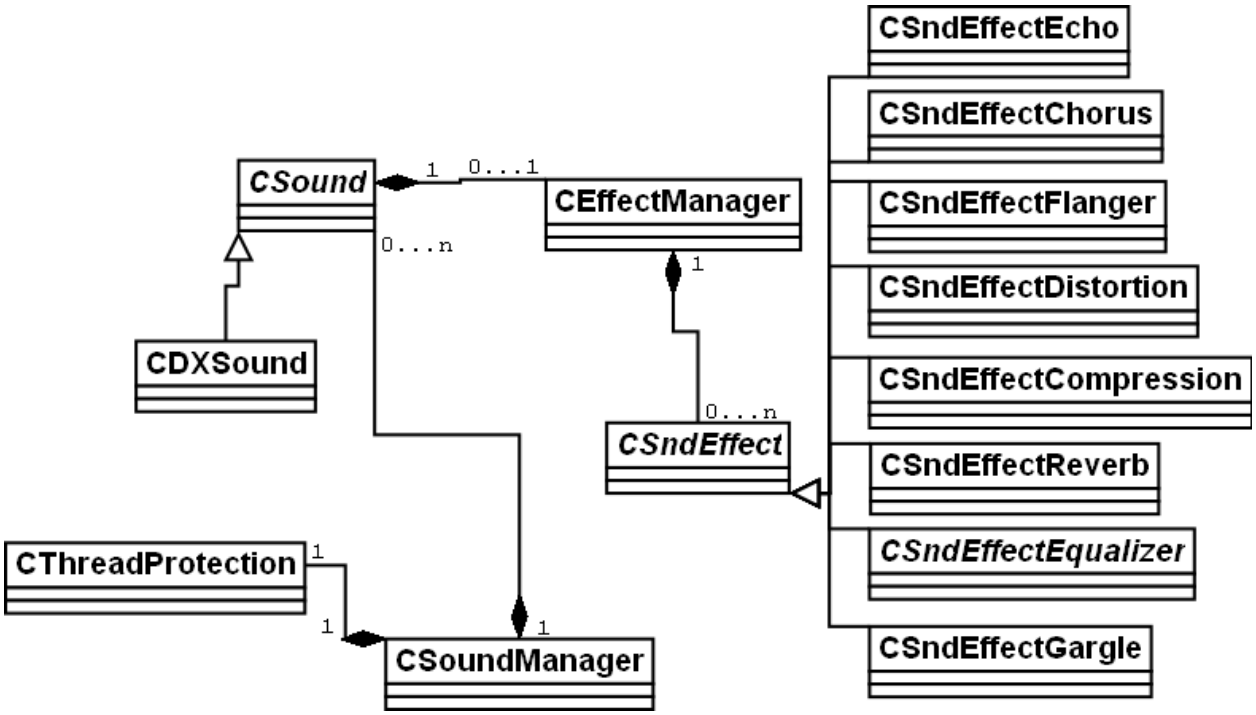
Kuten esimerkistä huomataan, kirjastoa käyttävä sovellus ei näe kirjaston sisällä olevaan efektimanageria ja efektin päälle asettaminen on helppoa. Efektin asettaminen tapahtuisi täysin samalla tavalla, oli kirjaston käyttämä äänirajapinta, mikä tahansa.

3.3.4 3D-ääniympäristö

3D-ääniympäristön luomisen toteuttaminen kirjastoon oli helppoa, koska erillistä manageriluokkaa ei tarvittu niiden käsittelemiseen. Paikan asettaminen kävi helposti, kutsumalla vain yhtä metodia ja antamalla tälle parametreina halutun paikan koordinaatit 3D-maailmassa. Tärkein asia, mistä piti huolehtia 3D-maailmaa hallittaessa, oli, ettei sovellus yritä asettaa 3D-paikkaa sellaiselle äänelle, joka sitä ei tue. Koodissa paikan asettaminen äänelle tapahtui seuraavasti.

```
// Pyydetään ääni managerilta ja asetetaan sen paikka halutuksi ( x, y, z )
soundManager.GetSound( 0 )->pSound->SetPosition( 0, 1.0, 0 );
```

3.3.5 Luokkarakenne



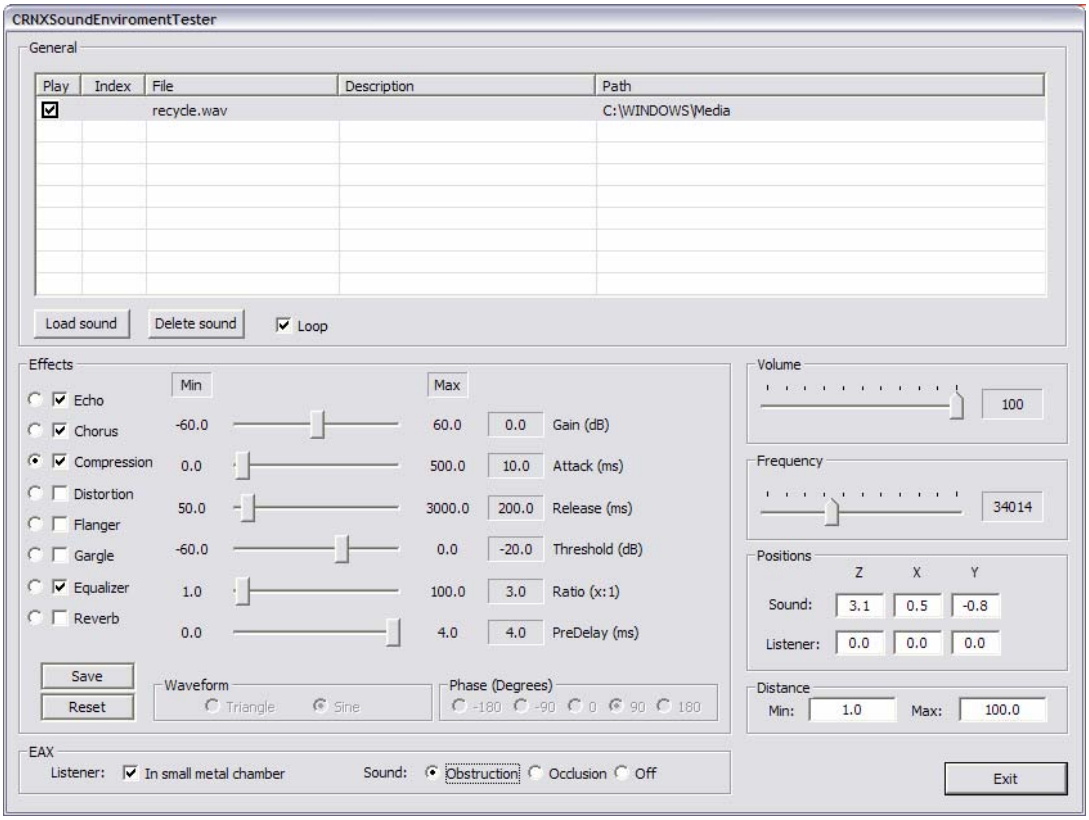
Kuva 3 Kirjaston luokkarakenne

Kuvassa 3 näkyy kirjaston lopullinen luokkarakenne. Kun kirjastoon halutaan lisätä tuki uudelle rajapinnalle, tarvitsee periyttää CSound-luokasta uusi luokka ja tehdä toteutus luokan vaatimille metodeille. Lisäksi CSoundManager-luokkaan on hieman lisättävä koodia. Koodin lisäykset liittyvät siihen, että manageri osaisi jatkossa luoda myös uutta rajapintaa tukevia luokkia.

4 TESTAUS KÄYTTÖLIITTYMÄ

Kirjaston testaamista varten päätettiin toteuttaa myös testaus käyttöliittymä, jonka avulla kirjaston ominaisuuksia voitiin testata kirjaston kehittämisen aikana, omana kokonaisuutena irrallaan varsinaisesta simulaattoriprojektista. Tämä osoittautui varsin hyödylliseksi projektin organisoinnin kannalta ja samaa sovellusta voidaan käyttää testaus- ja kehitysalustana jatkossakin. Pääsääntöisesti käyttöliittymän kehittäminen eteni samassa rytmissä, kuin itse kirjastonkin. Eli, kun kirjastoon lisättiin tuki uudelle toiminnolle, toteutettiin myös käyttöliittymään testaamista varten uusi ominaisuus.

Käyttöliittymän kehittyessä huomattiin, että sitä voisi käyttää myös muuhunkin kuin testaukseen, joten käyttöliittymän käytöstä pyrittiin tekemään mahdollisimman selkeää. Käyttötarkoituksen laajentuessa lisättiin esimerkiksi ominaisuus, jolla oli mahdollista helposti tallentaa valitulla äänellä käytössä ovat efektit ja näiden parametrit. Ominaisuuden avulla parametritieto voidaan helposti siirtää itse sovellusohjelmaan.



Kuva 4 Testidialogi

Kuvassa 4 näkyy testidialogi, kuvasta voidaan huomata, että äänissä käytettäviä ominaisuuksia on pyritty ryhmittelemään käytön selkeyttämiseksi.

Ylimpänä dialogissa olevassa listassa näkyy tällä hetkellä kirjastoon ladatut äänet, sekä mahdolliset tiedot äänistä. Listan alapuolella olevilla painikkeilla on mahdollisuus lisätä ja poistaa ääniä kirjastosta. Painikkeiden oikealla puolella olevalla loop-valinnalla ääni voidaan asettaa soimaan vain kerran tai soimaan toistuvasti.

Vasemmassa reunassa olevan Effects-alueen sisäpuolella olevat kontrollit asettavat ja muokkaavat listassa valittuna olevan äänen efektejä. Reunimmaisena olevat pyöreät valintapainikkeet määräävät, minkä efektin arvot näkyvät alueen keskellä olevissa kontrolleissa. Efektin nimen vieressä olevilla valinta ruuduilla kytketään kyseinen efektiin käyttöön ja pois, valitulle äänelle. Äänellä voi olla useita efektejä yhtäaikaisesti käytössä ja niitä voi laittaa päälle ja pois, ilman, että ääntä tarvitsee pysäyttää välillä. Näiden valintojen alapuolella oleva save-painike avaa uuden käyttöliittymän, jossa näkyvät valitulla äänellä käytössä olevat efektit parametreineen. Reset-painike nollaa valitun efektin parametrit oletusarvoihinsa.

Dialogin oikeassa reunassa ovat liukusäädöt muuttavat valitun äänen äänenvoimakkuutta ja taajuutta. Näiden alapuolella olevassa Positions kentissä sijaitsevat valitun äänen sekä kuuntelijan paikkakoordinaatit. Distance-alueen sisällä oleviin kenttiin voidaan syöttää valitulle äänelle sen kuuluvuusalueen minimi- ja maksimietäisyydet.

Dialogin alareunassa olevan EAX-alueen sisäpuolella olevat kontrollit muokkaavat käytössä olevaa ääniympäristöä. Nämä kontrollit päätettiin toteuttaa rajoitetusti, koska niiden täydellinen hallinta käyttöliittymän kautta, olisi vaatinut paljon tilaa. Jatkossa EAX-ominaisuuksille voidaan tehdä tarvittaessa oma käyttöliittymä.

5 YHTEENVETO

Kirjaston suunnittelu ja toteuttaminen tapahtui niille varatun aikataulun mukaisesti. Tähän päästiin valitsemalla toteutettavaksi vain sellaiset ominaisuudet, jotka olivat tarpeellisia kirjaston ensimmäisessä käyttökohteessa.

Kirjaston suunnittelun ja toteuttamisen aikana tuli opittua monia uusia asioita ja antoi näin hyvää kokemusta työelämän varalle. Uusista asioista varsinkin monisäikeisessä sovelluksessa toimivan komponentin suunnittelu, oli hyvää opetusta.

Missään kirjaston toteuttamisen vaiheessa ei ollut suuria ongelmia, kiitos hyvän ohjauksen valvojan taholta. Työn tuloksena oli halutunlainen äänentoistokomponentti, joka heti käyttöön simulaattoriprojektissa, jota varten se toteutettiin.

Seuraavana asiana kirjaston kehittämisessä on lisätä tuki kaikille DirectSound rajapinnan tarjoamille 3D-ääniparametrien muuttamiselle. Tulevaisuudessa on myös mahdollista toteuttaa tuki OpenAL-rajapinnan käytölle, jos komponenttia halutaan käyttää muussa kuin Windows-käyttöjärjestelmässä.

LÄHTEET

1 Hiebert, Garin. OpenAL 1.1 Programmer's Guide. [pdf-tiedosto]. [Viitattu 6.5.2006]. 107 s. Saatavissa: http://developer.creative.com/

2 DirectX Documentation for C++. [chm-tiedosto] [Viitattu 6.5.2006]. Saatavissa: http://www.microsoft.com/directx

3 Jot, Jean-Marc - Boom, Michael. Environmental Audio Extensions: EAX 2.0 version 1.3. [pdf-tiedosto]. 88s. [Viitattu 6.5.2006]. Saatavissa: http://developer.creative.com/