OpenAL

EECS 4462 - Digital Audio

November 6, 2018

YORK UNIVERSITÉ UNIVERSITY

OpenAL

- Cross-platform Audio API
  - Can be used for games, and other audio applications
- Similar style to OpenGL (G for graphics)
- One of many options for audio middleware
  - FMOD
  - Wwise
  - Fabric (for Unity)

OpenAL overview

- Each game scene is called a context
  - OpenAL class: ALCcontext
- Each context has
  - Several Buffers that contain audio data
  - Several Sources (points that emit sound)
  - Exactly one Listener (the position where the Sources are heard)
- Audio rendering is always done from the point of view of the Listener

Important Class: ALCdevice

- Models an audio device in the host machine
  - Typically, your sound card
- OpenAL allows you to get a list of audio devices and select the one you want to use
- Or you can just use the default audio device with

```
ALCdevice *device;
device = alcOpenDevice(NULL);
```

- You must do this before doing anything else audio-related

Important Class: ALCcontext

- Models an audio scene in the game
- You can create a default context and make it the current context with

```
ALCcontext *cxt;
cxt = alcCreateContext(device, NULL);
alcMakeContextCurrent(cxt);
```

- You must do this before as soon as you have a device
- The current context will apply to all the Buffers and Sources you will create next

Important Method: alGetError();

- Any call to an **`al*()`** function may cause an error
- You can check if an error has occurred as below

```
ALenum error;
error = alGetError();
if (error != AL_NO_ERROR) exit(2);
```

- Common error codes

```
AL_NO_ERROR
AL_INVALID_NAME
AL_INVALID_ENUM
AL_INVALID_VALUE
AL_INVALID_OPERATION
AL_OUT_OF_MEMORY
```

Error handling example

```
ALCcontext *context;
context =
        alcCreateContext(device, NULL);
if (!alcMakeContextCurrent(context))
{
  printf("%s",
          alGetString(alGetError()));
  exit(2);
}
```

At shut down…

- When audio functionality is not needed any more, we must destroy the context and close the audio device

```
context = alcGetCurrentContext();
device = alcGetContextsDevice(context);
alcMakeContextCurrent(NULL);
alcDestroyContext(context);
alcCloseDevice(device);
```

Creating Sources

- A Source is a source of audio that has a particular position in the 3D space, as well as a particular velocity
- Sources cannot be created directly
- You must use the **`alGenSources`** function
- Each Source has a “name”, which is actually an integer

```
ALuint source[2];
alGenSources(2,source);
```

- The above creates two Sources that you can refer to with **`source[0]`** and **`source[1]`**

Customizing Sources

- A set of alSource*() functions can be used to set the attributes of the various sources
- See the specification for a complete list of parameters

```
ALuint s;
alGenSources(1,&s);
alSourcef(s, AL_PITCH, 1);
alSourcef(s, AL_GAIN, 1);
alSource3f(s, AL_POSITION, 0, 0, 0);
alSource3f(s, AL_VELOCITY, 0, 0, 0);
alSourcei(s, AL_LOOPING, AL_FALSE);
```

YORK UNIVERSITÉ UNIVERSITY

Creating Buffers

- A Buffer is an object that holds audio data that can be played when associated with a Source
- Buffers cannot be created directly
- You must use the **`alGenBuffers`** function
- Each Buffer has a “name”, which is actually an integer

```
ALuint buffer[2];
alGenBuffers(2,buffer);
```

- The above creates two Sources that you can refer to with **`buffer[0]`** and **`buffer[1]`**

Loading data into a Buffer

- The alut library provides functions to read various formats into a buffer
- Use the **alutLoadWAVFile** function for WAV files

```
ALsizei size, freq;
ALenum format;
ALvoid *data;
alutLoadWAVFile("bark.wav", &format,
 &data, &size, &freq);
alBufferData(buffer, format,
 data, size, freq);
```

Playing Sound

- First, associate a source with a buffer

```
alSourcei(source, AL_BUFFER, buffer);
```

- Then, play!

```
alSourcePlay(source);
```

Making sure a source is finished

- Sources play audio is separate threads
- Before exiting, you might want to ensure that the audio thread is finished

```
ALint source_state;
alGetSourcei(source, AL_SOURCE_STATE,
  &source_state);
while (source_state == AL_PLAYING) {
  alGetSourcei(source, AL_SOURCE_STATE,
    &source_state);
}
```

Deleting Sources and Buffers

- When sources and buffers are not needed any more, they can be deleted

```
alDeleteSources(1, &source);
alDeleteBuffers(1, &buffer);
```

Customizing the Listener

- The Listener is created and destroyed automatically
- It can be customized in a manner similar to Sources

```
alListener3f(AL_POSITION, 0, 0, 1.0f);
alListener3f(AL_VELOCITY, 0, 0, 0);

ALfloat listenerOri[] =
{0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f};

alListenerfv(AL_ORIENTATION,
  listenerOri);
```

Source and Listener attributes

- Sources and Listener have a number of common attributes that can be customized

  `AL_POSITION`

  `AL_VELOCITY`

  `AL_GAIN`

- The first two require X,Y,Z coordinates (see next slide) while **`AL_GAIN`** requires a positive float
  - **`AL_GAIN`** of 1 is no attenuation
  - **`AL_GAIN`** of 0.5 id 6dB quieter
  - **`AL_GAIN`** of 0 is silence
  - **`AL_GAIN`** of more than one is possible but the sound engine may restrict it to avoid clipping

Coordinate system

- OpenAL uses a right-handed Cartesian coordinate system
  - X points right
  - Y points up
  - Z points towards the viewer
- Default position for listener and all sources is {0,0,0}
- Examples
  - {-2,0,0}: Left of the listener
  - {2,0,2}: Right and behind the listener

AL_POSITION

- Specifies the 3D position of a source (or the listener)
- By default, independent of the position of the listener, but it can be toggled to relative by setting
  `AL_SOURCE_RELATIVE` to **`AL_TRUE`**
- Used to calculate attenuation for the sound emanating from the source
  - The closer the source to the listener, the louder it should sound
- OpenAL has a number of distance models to implement this

Distance models

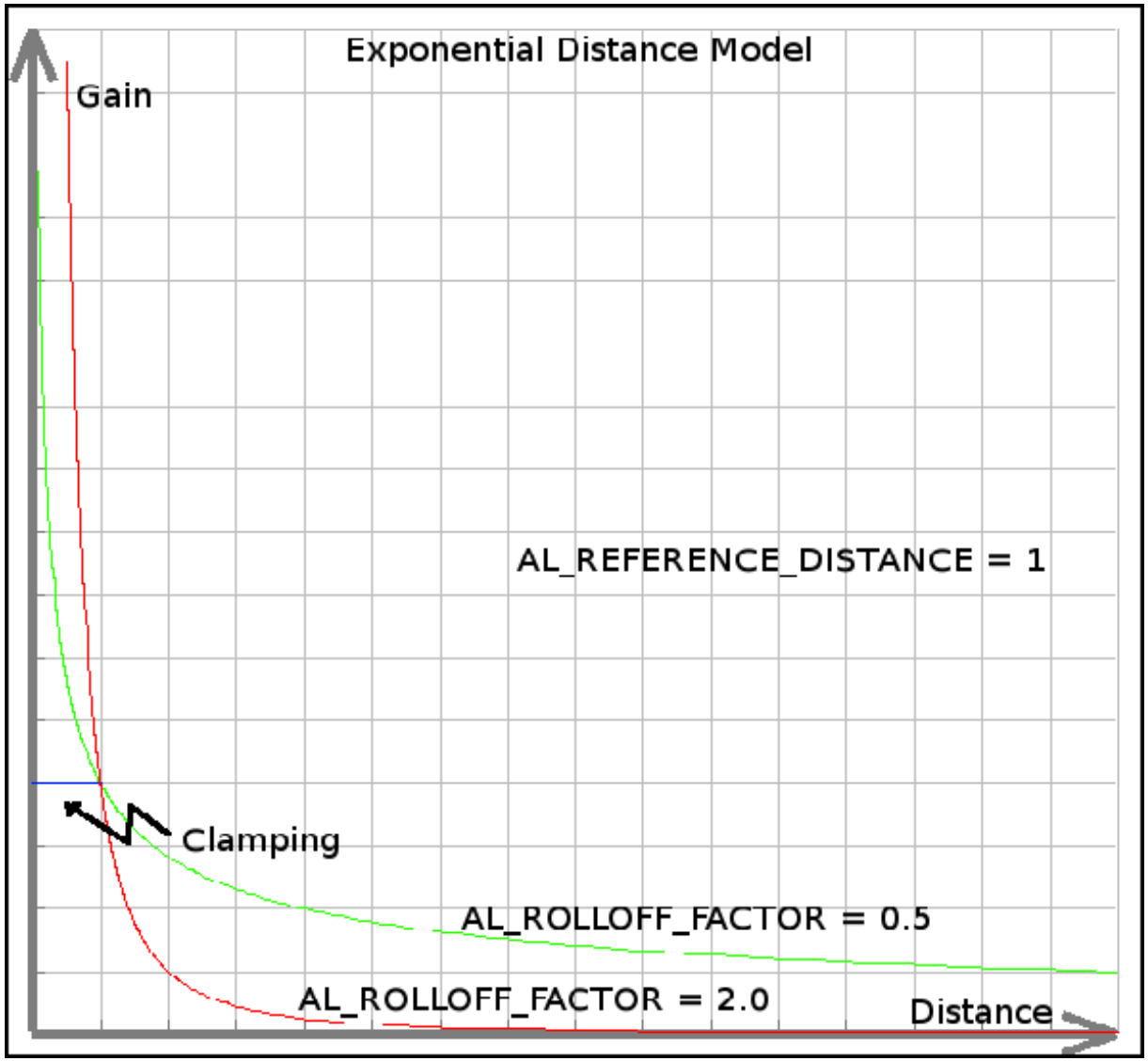
- The default distance model is **`AL_INVERSE_DISTANCE_CLAMPED`**
- **`INVERSE`** means that attenuation follows the inverse square law
- **`CLAMPED`** means that once the distance becomes smaller than a threshold (set by **`AL_REFERENCE_DISTANCE`**), gain does not increase any more, i.e. gain is clamped

Changing distance models

- The distance model behaviour can be changed by setting the value of **`AL_ROLLOFF_FACTOR`**
  - Larger values ➔ more drastic attenuation
- You can also set a completely different distance model with **`void alDistanceModel(ALenum m);`**
- Possible values include

```
AL_NONE
AL_INVERSE_DISTANCE
AL_LINEAR_DISTANCE
AL_EXPONENT_DISTANCE
```

Exponential Clamped DM



Calculating overall gain

- Calculating the overall gain for a particular sound is complicated. It depends on
  - The listener position and orientation
  - The source position
  - The source directionality (discussed later)
  - The distance model, rolloff factor and reference distance
  - The source gain
    - Sources can also set **`AL_MIN_GAIN`** and **`AL_MAX_GAIN`**
  - The listener gain

AL_VELOCITY

- Specifies the speed and direction of a source
- Independent of **`AL_POSITION`**
  - Changes to one do not affect the other
- Used to synthesize the Doppler effect
  - If the source is moving towards the listener, the frequencies in its sound increase
  - If the source is moving away from the listener, the frequencies in its sound decrease
  - https://www.youtube.com/watch?v=h4OnBYrbCjY

Doppler effect in OpenAL

- Calculated automatically
- Can exaggerate or deemphasize with

```
void alDopplerFactor(ALfloat df);
```

  - Default value is 1

- Can also change the speed of sound which affects the magnitude of the Doppler effect

```
void alSpeedOfSound(ALfloat speed);
```

  - Default value is 343.3

Directional Sources

- By default, Sources are omni-directional, i.e. they get attenuated in the same way in all directions
- Many sound sources are directional though
  - If a character is facing away from the listener, their gain should be attenuated
- To make a source directional, set **`AL_DIRECTION`** to the X,Y,Z coordinates of their direction, e.g.

```
alSource3i(src, AL_DIRECTION, 1,1,1);
```

Cones

- A directional source must define an inner and outer cone
- **AL_CONE_INNER_ANGLE** defines the angle of the inner cone inside which no directional attenuation will take place
- **AL_CONE_OUTER_ANGLE** defines an outer cone, outside of which, gain will be attenuated by **AL_CONE_OUTER_GAIN**
- Attenuation between the inner and outer cones is interpolated