CSCI 441 - Lab 04
Friday, September 25, 2015
LAB IS DUE BY **FRIDAY OCTOBER 2 11:59 PM**!!

Today, we'll spice up our OpenGL applications by adding sound using OpenAL. A couple of notes before we get started:

- It is highly, highly, and strongly recommended to use the lab machines as OpenAL and ALUT is compiled for you. Use non-lab time to get your personal machines set up. I will help you outside of class if you need it.
- This write up is a bit longer than some of the others and has a lot of TODO points. But each TODO is very small. We need to put things in many different places to get it all working, hence the extra steps. It should not take you an extra long amount of time to get it working. Once the set up is done, it's just calling the same functions over and over in different places.
- If you download your own WAV files to use, not all WAV files are created equal. Unfortunately, I am not versed well enough in WAV formats but some files will not play with OpenAL. Other WAV files will play, but have no sound decay based on the proximity of the listener and source. Finally, some WAV files behave properly – I've included well behaved WAVs for you to use.
- The Makefile has been restructured once again. At the very top, are some path variables to where the include, lib, and bin files are located if using your personal machine. If building in the lab, ignore these three paths. There is a variable to denote if you building in the lab or not. Finally, there are a set of flags to denote what libraries we are using. You can modify these values as needed to build different projects. We'll add more library flags as we get further into the course.

Please answer the questions as you go inside your README.txt file.

Step 1 – Getting Our Code To Compile

THIS IS AN INDIVIDUAL TASK.

We need to do a little setup first. You must follow these instructions exactly for the latest Makefile to work. We'll be performing similar steps like this in future labs so be sure to follow directions exactly.

1. Navigate to the root of your Z: drive
2. Change into **CSCI441GFx** (it should look like: Z:/CSCI441GFx)
3. From this lab zip file, copy the contents of the include and lib folders into the respective folders you just made.
4. Make a new folder called **bin** (it should look like Z:/CSCI441GFx/bin)
5. From this lab zip file, copy the contents of the bin folder into this folder you just made.

You should now have the following file structure:

- Z:
  - CSCI441GFx
    - bin
      - libalut.dll
      - OpenAL32.dll
    - include
      - AL
        - al.h
        - alc.h
        - alext.h
        - alut.h
      - GL
        - glui.h
    - lib
      - libalut.dll.a
      - libglui.a
      - libOpenAL32.dll.a

Alright cool, we've got OpenAL set up. Make sure you set everything up correctly, switch into the openalTest folder and type `make` to build this example program.

Ok, back to our lab.

Step 2 – Initializing OpenAL & ALUT

From this point forward in this lab, you may work with another student. HOWEVER, you each MUST submit your OWN submission. IF you work with another student, you BOTH MUST make note in your README who you had worked with.

You have the option of using the included lab04.cpp, which is my finished Lab02 for the Flight Simulator, OR you may use your Assignment3 / Assignment4 code to add sound to that. In the Flight Simulator, we are using a Free Cam while your assignments use an ArcBall Cam. The Camera implementation does not matter as we'll handle all cases the same. I will reference some different areas in the lab04.cpp code so if you choose to use your own Assignment, then you will need to place these calls in the appropriate place.

Let's look what has been added first. At the top of our program, we have the headers for OpenAL and ALUT. Since we are good programmers we add this for OS X or other OSes. We now have a series of global OpenAL variables to keep track of our device, context, buffers, and sources. You may decide to add more later on.

Scroll down to the main() function. This looks a little different and may a little cleaner. We have two helper functions, `initializeOpenGL()` and

`initializeOpenAL()`. These will take care of all of our one time setup. You can look at `initializeOpenGL()` to get an idea of what is going on. We'll use `initializeOpenAL()` to setup OpenAL. Find `TODO #01`.

The first steps we need to take are to initialize ALUT and create our device/context for OpenAL. Refer to Slide #12 from Wednesday's lecture to accomplish this setup with four lines of code. (We already have global variables set up for these steps and we don't need to worry about getting the current context or device.)

We also need to make sure we clean up our context and device when we exit our program. Find `TODO #02` and notice we've introduced a new callback called `atexit()`. This callback takes a function and executes that function right before the program exits. We are allowed to add multiple functions and they will get put onto a stack to be called. Register a callback for our cleanupOpenAL() function.

Now go to `TODO #03` and we will close our session by adding the lines from Slide #13.

Perfect! At this point, you should be able to compile and run without any errors.

Step 3 – Placing Our Listener

In order to hear anything, we need to have a listener to absorb all the sounds. We're going to need to create some helper functions and then finally place our listener. `TODO #04` is inside a helper function that places the listener into our world. This function is designed to operate in a similar manner to `gluLookAt()`. The function takes nine arguments. The first three denote the world position of our listener. The next three denote the direction the listener is facing. The final three denote the up vector for our listener. These nine parameters correlate to how our camera is position and oriented as well. Slide #17 has our OpenAL listener commands. We will need to set two properties: `AL_POSITION` and `AL_ORIENTATION`.

First, we'll take care of `AL_POSITION`. We will need to call

```
alListenerfv( AL_POSITION, listenerPosition );
```

where `listenerPosition` is of type `ALfloat[]`. The array has a length of three and the three components correspond to the X, Y, and Z position of our listener.

Next, we'll set the `AL_ORIENTATION`. We will call the same function but with different values now:

```
alListenerfv( AL_ORIENTATION, listenerOrientation );
```

where `listenerOrientation` is again of type `ALfloat[]`. This time the array has a length of six. The first three components correspond to the X, Y, and Z direction the listener is facing. The final three components correspond to the X, Y, and Z of the up vector.

Now that we have our helper function in place, let's use it! `TODO #05` is inside our render scene function. We want our listener to always be wherever our camera is and facing the same direction as our camera. Whenever our camera moves, our listener should move with the camera. Can we reuse some of our existing global variables to accomplish this task?

You can compile and run and should not get any errors has you move around your world. It's hard to know at this point if anything is working because we can't hear anything. Let's add our first sound.

Step 4a – Generating Buffers & Sources

Now let's populate our buffers. First, we need to ask OpenAL for a number of handles to buffer objects. Slide #15 tells us how to accomplish this. We have global variables set up so at `TODO #06` (back in our `initializeOpenAL()` function, since we only need to do this step once) we need to call:

```
alGenBuffers( NUM_BUFFERS, buffers );
```

where the two arguments are the number of buffers to create and then a pointer to an array of buffers to store these handles in. We will do the same thing to create our sources from Slide #16.

```
alGenSources( NUM_SOURCES, sources );
```

where the two arguments are the number of sources to create and a pointer to an array of sources to store these handles in.

We want to make sure we clean up our memory, so back in `cleanupOpenAL()` or `TODO #07`, we want to delete our buffers and sources before the program exits. The two calls we need to add are

```
alDeleteBuffers( NUM_BUFFERS, buffers );
alDeleteSources( NUM_SOURCES, sources );
```

Hopefully, after looking at how the two are generated, the deletion should be self-explanatory as to how they work.

Step 4b – Positioning Our First Stationary Source

Slide #15 shows us how to populate a buffer. Since we are good programmers, we will make sure our code is cross-platform worthy and handle the OS X API and the other APIs. Add the following lines of code at `TODO #08`.

```
#ifdef __APPLE__
	alutLoadWAVFile( (ALbyte*)"<filename>", &format, &data, &size, &freq );
#else
	alutLoadWAVFile( (ALbyte*)"<filename>", &format, &data, &size, &freq, &loop );
#endif
	alBufferData( buffers[0], format, data, size, freq );
	alutUnloadWAV( format, data, size, freq );
```

A couple of notes what this is doing. First we check if our operating system is an Apple system or not. If it is, we'll use the Mac version of `alutLoadWAVFile()`. Otherwise we'll use the Windows/Linux version. You need to replace `<filename>` with the WAV file you want to play. You can find one from the internet, or use the included `"wavs/siren.wav"` file. We are passing a string as an argument but ALUT wants this as a byte array so we can type cast to `ALbyte*`. When the WAV file is loaded, the remaining arguments are populated. I hope the names of the variables explain what each does. Next, we tell OpenAL to put the file data into our buffer. Finally, we unload the WAV file from memory to free up some storage space. Ok, our buffer is now populated. Let's get a source going.

We'll reference Slide #16 to set up our source. We only need to set two parameters at this time, `AL_BUFFER` and `AL_LOOPING`. The first associates a sound buffer with this sound source, so the source knows what sound to play. The second tells the source to continually loop the buffer over and over. The two calls are then

```
alSourcei( sources[0], AL_BUFFER, buffers[0] );
alSourcei( sources[0], AL_LOOPING, AL_TRUE );
```

We are telling our first source to be associated with our first buffer and to loop this buffer. Now we need to position our source somewhere, well we have another helper function for that.

`TODO #09` is inside this helper function. It takes four arguments: the source handle and the XYZ location to position this source in our world. This operates similar to positioning our listener, we will make use of the AL_POSITION property.

```
alSourcefv( src, AL_POSITION, srcPosition );
```

where `srcPosition` is a `ALfloat[]` array just like the `listenerPosition` was. Let's go ahead and place our first source. We never want this source to move

so we will call it once inside `initializeOpenAL()` at `TODO #10`. We'll call our helper function and place `sources[0]` at the origin.

Grool (I meant to type cool and then I started to type great). We have a buffer with our source, our source is positioned in our world, our listener is moving around our world, we just need some sound to play and let's hear it!

At `TODO #11` we will start our source before we get into the main GLUT loop. Slide #18 holds the key to this:

```
alSourcePlay( sources[0] );
```

We tell OpenAL which source to start playing.

That's it! Compile and run. Start flying around until you can find the source where the siren is coming from. (HINT: There's a giant Teapot at the origin. Find where the teapot is drawn in the code for some extra programming tips.)

Q1: Can you initially hear the siren and then it gets quieter?
Q2: After all that setup, does the siren get louder as you get closer?

If you answered YES to Q1, there's a perfectly valid reason why that is happening. We set the position of our source, we start the source playing, then in our main loop we position our listener. By default, the listener is at the origin (with the source!) and then we start playing. So we hear the sound, then we move our listener further away and it gets softer. At `TODO #12`, let's place our listener where our camera is before we play the sound to avoid this issue.

Compile and run again.

Q3: Do you now have to search for the source before you start hearing it?

Fabulous, let's add a couple more sources.

Step 5: Making Our Plane Go

The `TODOs` are going to stop at this point so you will need to figure out where to place everything. Repeat **Step4B** to create a second buffer and source using the `Running.wav` sound file. This will go into `buffers[1]` and `sources[1]` and we want this to loop as well. Every frame after we place our listener, we want to place this source where our plane is.

We only want this sound to play when our plane is flying. So when we press the 'w' or 's' keys to fly forwards/backwards we want to start this source (HINT: This will

go into your cases for `normalKeysDown()`). And when we stop flying forwards/backwards, we need to stop our source with

```
alSourceStop( sources[1] );
```

where we tell OpenAL which source to stop (HINT: This goes in `normalKeysUp()`)..

Compile and run.

Q4: Do you hear the propeller sound only when you fly?
Q5: If you zoom in/out (Ctrl+Left Click), does the sound get louder/softer?
Q6: As you continue to fly forwards, what does it sound like is happening?

What is happening, every time we press 'w' we are telling OpenAL to start playing the source so it continually starts over. What we really want is to only start the source if it is not playing already. We can ask OpenAL for this information. Before we call

```
alSourcePlay( sources[1] );
```

let's check the state of the source. We'll use an enum for this:

```
ALenum sourceState;
alGetSourcei( sources[1], AL_SOURCE_STATE, &sourceState );
```

Here are asking OpenAL to return to use the current value of the state properly on this source. Now we just need to check if the state is not `AL_PLAYING`. If it's not, then we'll start our source. If the state is `AL_PLAYING`, then we will do nothing as the sound is already going.

Compile and run.

Q7: Does it sound better now?

Step 6: Adding Some Background Music

Let's create one final buffer and source. This time we'll use the "`background.wav`" sound file that goes into `buffers[2]`, `sources[2]`, and loops.

We want this source to be our background music, so we want it positioned with our listener every frame. We'll start this source before we get into our main GLUT loop, so make sure we position it with our listener before we start the source.

Compile and run.

Q8: Do you hear some kicking background music?

Fly around as you look for the siren.

Q9: Are you able to hear all three sounds together? As you circle the teapot, do you hear the siren move from the left channel to the right channel?

CONGRATS! You did it! You made your flight simulator (or assignment) come to life. Feel free to repeat these steps with your future assignments for an added cool factor.

Q10: Was this lab fun? 1-10 (1 least fun, 10 most fun)
Q11: How was the write-up for the lab? Too much hand holding? Too thorough? Too vague? Just right?
Q12: How long did this lab take you?
Q13: Any other comments?

To submit this lab, zip together your source code, WAV files, Makefile, screenshot, and README.txt with questions. Name the zip file <HeroName>_L4.zip. Upload this on to Blackboard under the Lab04 section.

LAB IS DUE BY **FRIDAY OCTOBER 2 11:59 PM**!!