CSc 133 Lecture Notes

12 - Introduction to Sound

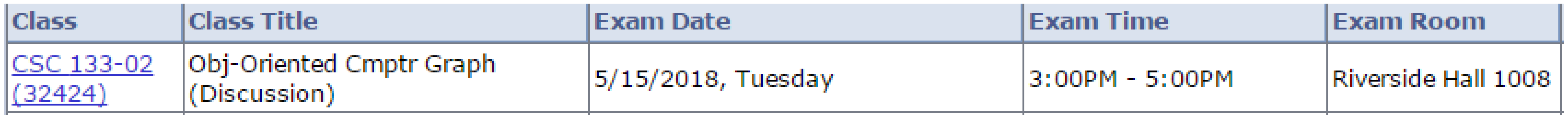
Computer Science Department

California State University, Sacramento

<u>Announcement</u>

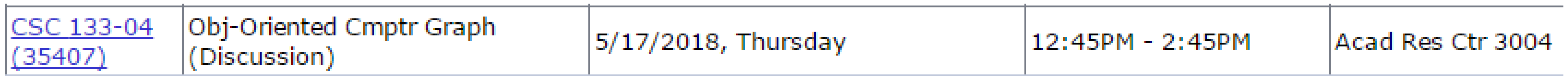
Final exam schedule:

Section 2:



Class	Class Title	Exam Date	Exam Time	Exam Room
CSC 133-02 (32424)	Obj-Oriented Cmptr Graph (Discussion)	5/15/2018, Tuesday	3:00PM - 5:00PM	Riverside Hall 1008

Section 4:

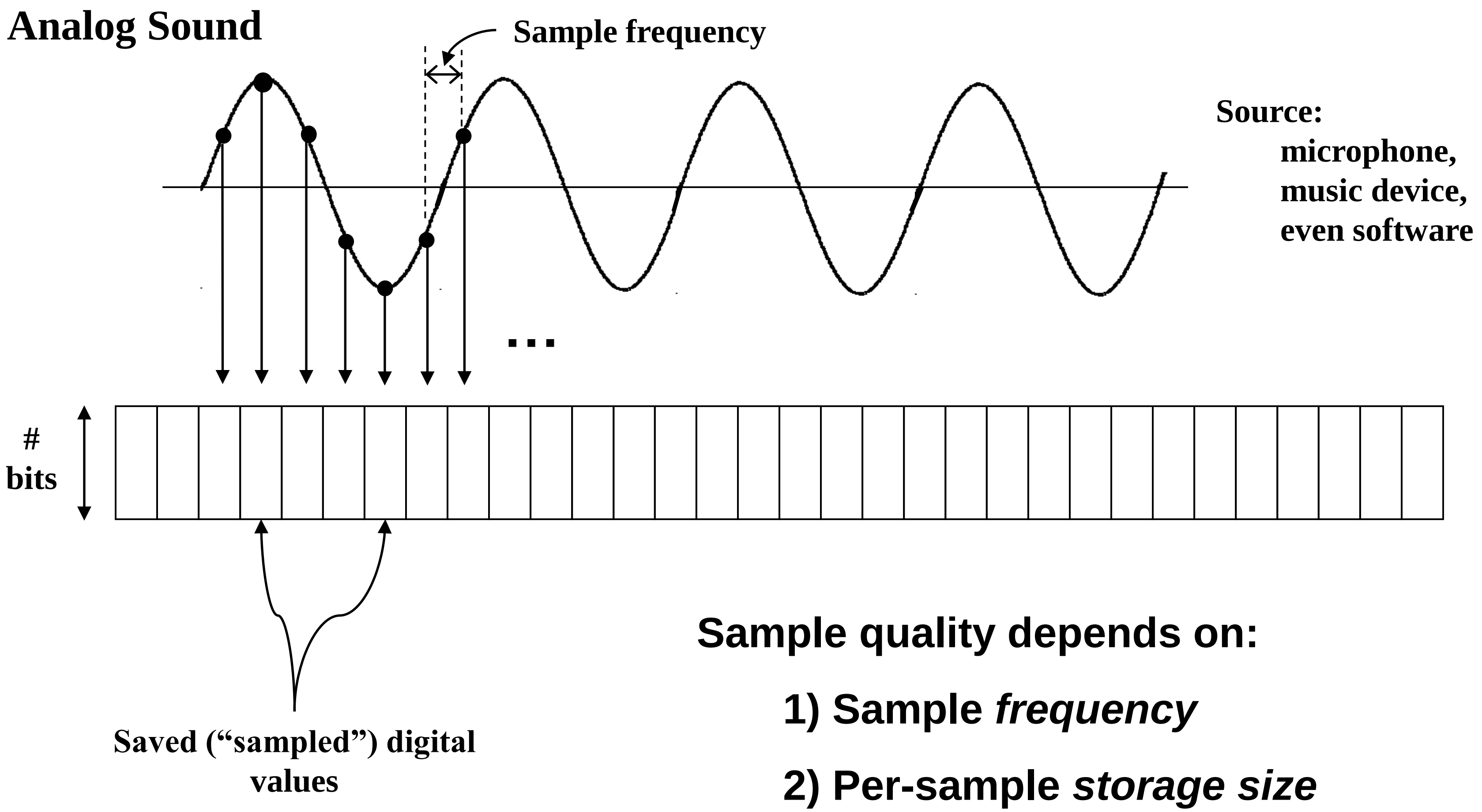


CSC 133-04 (35407)	Obj-Oriented Cmptr Graph (Discussion)	5/17/2018, Thursday	12:45PM - 2:45PM	Acad Res Ctr 3004

Overview

- Sampled Audio
- Sound File Formats
- Popular Sound APIs
- Playing Sounds in CN1
  - Creating background sound that loops

Sampled Audio



Sample quality depends on:

1) Sample *frequency*

2) Per-sample *storage size*

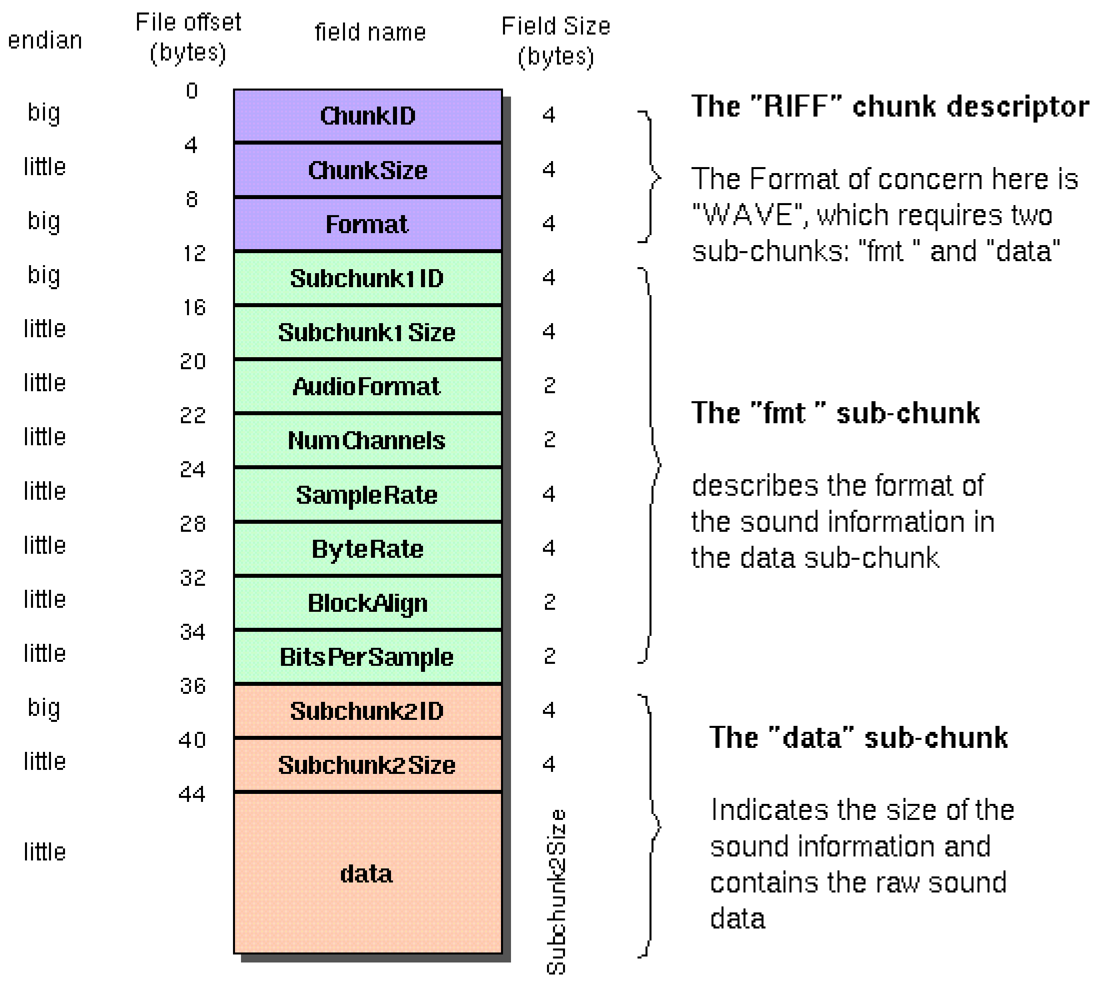
(bits per sample)

Sound File Formats

`.au`	Sun Audio File (Unix/Linux)
`.aiff`	Audio Interchange File Format (Mac)
`.cda`	CD Digital Audio (track information)
`.mpx`	MPEG Audio (mp, mp2, mp3, mp4)
`.mid`	MIDI file (sequenced, not sampled)
`.ogg`	Ogg-Vorbis file (open source)
`.ra`	Real Audio (designed for streaming)
`.wav`	Windows “wave file”

Finding sound files: **`www.findsounds.com`**

Example: WAVE Format



endian	File offset (bytes)	field name	Field Size (bytes)
big	0	ChunkID	4
little	4	ChunkSize	4
big	8	Format	4
big	12	Subchunk1ID	4
little	16	Subchunk1Size	4
little	20	AudioFormat	2
little	22	NumChannels	2
little	24	SampleRate	4
little	28	ByteRate	4
little	32	BlockAlign	2
little	34	BitsPerSample	2
big	36	Subchunk2ID	4
little	40	Subchunk2Size	4
little	44	data	Subchunk2Size

The "RIFF" chunk descriptor

The Format of concern here is "WAVE", which requires two sub-chunks: "fmt " and "data"

The "fmt " sub-chunk

describes the format of the sound information in the data sub-chunk

The "data" sub-chunk

Indicates the size of the sound information and contains the raw sound data

(ASCII “RIFF”, 0x52494646)

(ASCII “WAVE”, 0x57415645)

(ASCII “fmt ”, 0x666D7420)

(ASCII “data”, 0x64617461)

Image credit: http://ccrma.stanford.edu/courses/422/projects/WaveFormat/

Popular Sound API's

- Java `AudioClip` Interface
- JavaSound
- DirectSound / DirectSound3D
- Linux Open Sound System (OSS)
- Advanced Linux Sound Architecture (ALSA)
- OpenAL / JOAL

Java `AudioClip` Interface

- Originally part of web-centric **`Applets`**
- Supports
  - Automatic loading
  - **`play(), loop(), stop()`**
    - No way to determine progress or completion
- Supported sound file types depend on JVM
    - Sun default JVM: *.wav*, *.aiff*, *.au* , *.mid,* others…

Java Sound API

- A *package* of expanded sound support

```
import javax.sound.sampled;
import javax.sound.midi;
```

- New capabilities:
  - Skip to a specified file location
  - Control volume, balance, tempo, track selection, etc.
  - Create and manipulate sound files
  - Support for streaming
- Some shortcomings
  - Doesn’t recognize some common file characteristics
  - Doesn’t support spatial (“3D”) sound

- “Open Audio Library”
  - 3D Audio API (**`www.openal.org`**)
- Open-source
- Cross-platform
- Modeled after OpenGL
- Java binding (“JOAL”):

  `www.jogamp.org`

Playing Sounds in CN1

- **`Import:`**

  `com.codename1.media.Media;`

  `com.codename1.media.MediaManager;`
- **`Media`** object should be created to play sounds.
- **`Media`** objects is created by the overloaded **`creatMedia()`** static method of the **`MediaManager`** class.
- **`createMedia()`** takes in an **`InputStream`** object which is associated to the audio file.
- **`Media, MediaManager`**, and **`InputStream`** are all build-in classes.

Important tips

- You must copy your sound files directly under the **`src`** directory of your project.
- You may need to refresh your project in your IDE (e.g., in Eclipse select the project and hit F5 OR right click on the project and select “Refresh”) for CN1 to properly locate the sound files newly copied to the **`src`** directory.

Creating and playing a sound

```
import java.io.InputStream;
import com.codename1.media.Media;
import com.codename1.media.MediaManager;
...
/** This method constructs a Media object from the
 *  specified file, then plays the Media.
 */
public void playSound (String fileName) {
 try {
   InputStream is = Display.getInstance().getResourceAsStream(getClass(),
                                                                     "/"+fileName);
   Media m = MediaManager.createMedia(is, "audio/wav");
   m.play();
   }
 catch (IOException e) {
   e.printStackTrace();
 }
}
//this method calls playSound() to play alarm.wav copied directly under the src directory
public void someOtherMethod(){
   playSound("alarm.wav")
}
```

Encapsulating the sound

```
/** This class encapsulates a sound file as an Media inside a
 *  "Sound" object, and provides a method for playing the Sound.
 */`
public class Sound {
   private Media m;
   public Sound(String fileName) {
      try{
      InputStream is = Display.getInstance().getResourceAsStream(getClass(),
                                                                  "/"+fileName);
      m = MediaManager.createMedia(is, "audio/wav");
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
   }

   public void play() {
      //start playing the sound from time zero (beginning of the sound file)
      m.setTime(0);
      m.play();
   }
}
```

Encapsulating the sound (cont)

- In the assignments, you should use encapsulated sounds.
- Create a single sound object for each audio file:

```
Sound catCollisionSound = new Sound(“meow.wav”);
Sound scoopSound = new Sound(“scoop.wav”);
```

- Operations that belong to the same type should play this single instance (e.g., make all cat-cat collisions call `catCollisionSound.play()`), instead of creating new instances.

Looping the Sound

- To create a sound which is played in a loop (e.g., the background sound), **`Media`** object **`m`** indicated above should be created differently.
- We must attach a **`Runnable`** object to it which is invoked when the media has finished playing.
- The **`run()`** method of the **`Runnable`** object must play the sound starting from its beginning.

Encapsulating Looping Sound

```
/**This class creates a Media object which loops while playing the sound
 */
public class BGSound implements Runnable{
  private Media m;

  public BGSound(String fileName){
    try{
      InputStream is = Display.getInstance().getResourceAsStream(getClass(),
                                                                   "/"+fileName);
      //attach a runnable to run when media has finished playing
      //as the last parameter
      m = MediaManager.createMedia(is, "audio/wav", this);
    }
    catch(Exception e){
      e.printStackTrace();
    }
  }
  public void pause(){ m.pause();} //pause playing the sound
  public void play(){ m.play();} //continue playing from where we have left off

  //entered when media has finished playing
  public void run() {
    //start playing from time zero (beginning of the sound file)
    m.setTime(0);
    m.play();
    }
}
`
```

Use of Encapsulated Looping Sound

```
/**This form creates a looping sound and a button which pauses/plays the looping sound
 */

public class BGSoundForm extends Form implements ActionListener{

  private BGSound bgSound;
  private boolean bPause = false;
  public BGSoundForm() {
    Button bButton = new Button("Pause/Play");
    //...[style and add bButton to the form]
    bButton.addActionListener(this);
    bgSound = new BGSound("alarm.wav");
    bgSound.play();
  }

  public void actionPerformed(ActionEvent evt) {
    bPause = !bPause;
    if (bPause)
      bgSound.pause();
    else
      bgSound.play();
  }
}
`
```