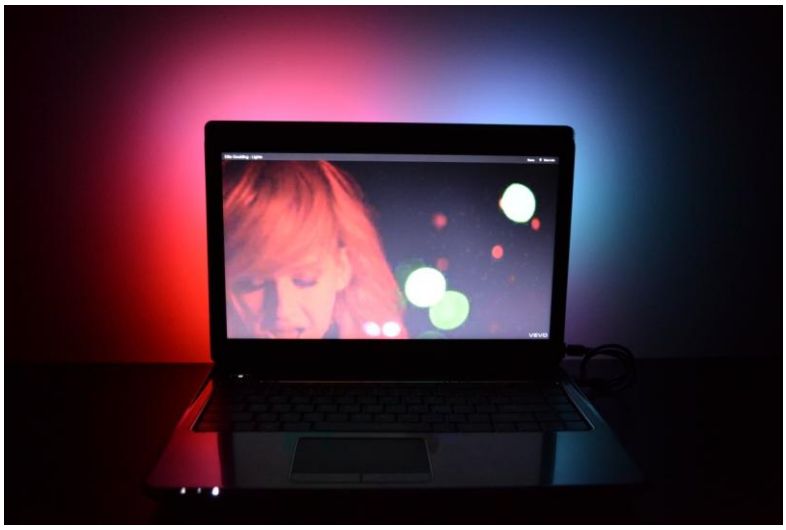
COMPUTER AMBIENT LIGHTING SYSTEM



Rajarshi Roy (roy18)

(roy18@illinois.edu / rajarshiroy91@gmail.com)

Supervised by:

Zuofu Cheng, Lippold Haken

December 16th, 2012

ECE 395: Advanced Digital Projects Lab (2 credits hours)

University of Illinois at Urbana Champaign

Contents:

1. Introduction

Ambient lighting has been introduced in the television market by Koninklijke Philips Electronics Corporation [1]. The concept behind ambient lighting is to light up the wall behind and around a television unit real-time based on the image being displayed on the television at any point on time. These systems provide an immersive viewing experience. Furthermore, these systems reduce eye strain by providing ambient lighting without the distracting element of white or yellow lighting.

Ambient lighting systems are only available for Philips televisions and not for computers. Considering that cinema and television shows are increasingly being viewed in computers, ambient lighting is highly applicable for computers. Furthermore, ambient lighting for computers will allow a more immersive gaming experience.

Due to the advantages of ambient lighting for computers, this project creates a complete ambient lighting system for computers. Some features of the system are that it is supported on multiple operating systems (based on standard USB-Serial drivers and Java runtime environment), can be easily attached to laptops or computer monitors.

The system consists of a wooden rounded-rectangular frame with 22 LEDs and USB port for connectivity to a computer as shown in Figure 1. An ATMEGA 328 based Arduino Mini microcontroller is used to control the LEDs. A FT232 USB-Serial IC is used to communicate between the USB port and the microcontroller. On the computer end, a standard USB-Serial driver is required to use the FT232 IC.

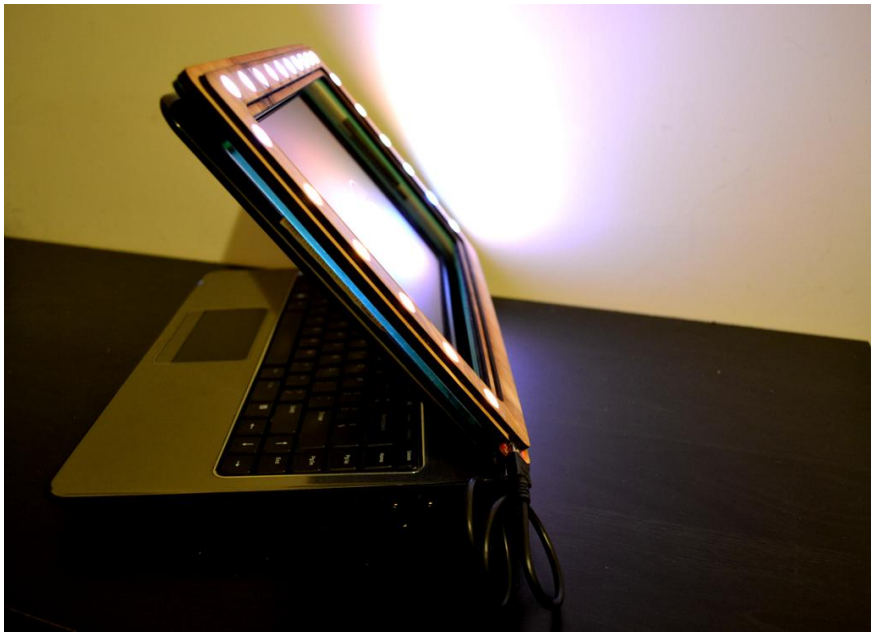
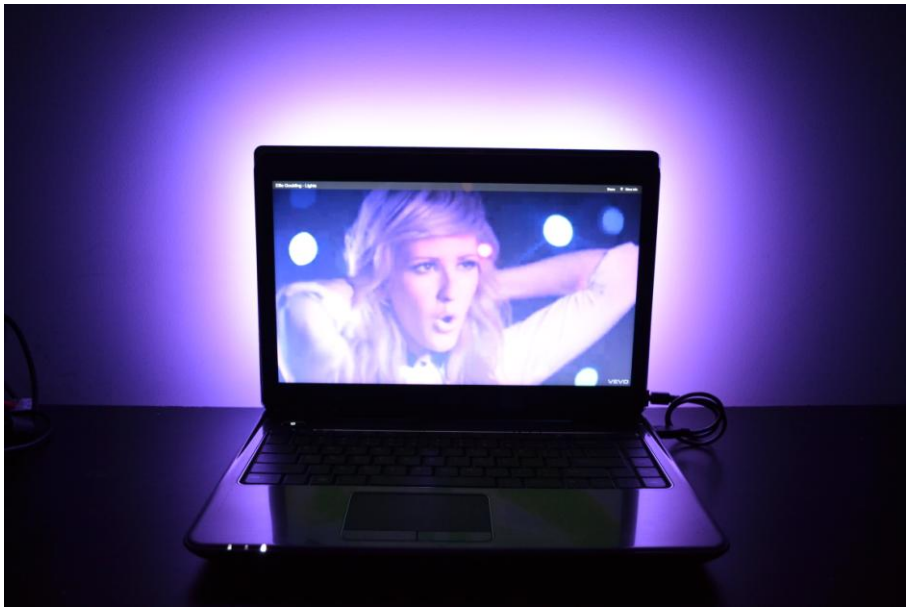


Figure 1: System connected to laptop USB port.

The software supports three lighting modes: “average”, “edge” and “sound”. The average mode controls the color of all the LEDs to be the average color of the image displayed on the computer screen real-time. The edge mode controls the color of each LED individually to be of the average color of the screen region corresponding to the physical location of the LED real-time. The sound mode controls the color of each LED individually based on a sound visualization algorithm that analysis the frequencies of the audio being played on the computer real-time. The sound mode is intended to be used when listening to music. The edge mode is intended for cinema or shows. The average mode is used for situations where the important video content is concentrated towards the center of the screen. Figure 2 shows the user interface of the three modes. The bottom of the window shows the color of the LEDs real-time. The user can cycle through modes simply by clocking anywhere on the window. The behavior of the working system during three modes can be observed in Figure 3.

Figure 2: Screenshots of software in three modes: (left) Average mode (center) Edge mode (right) Sound mode



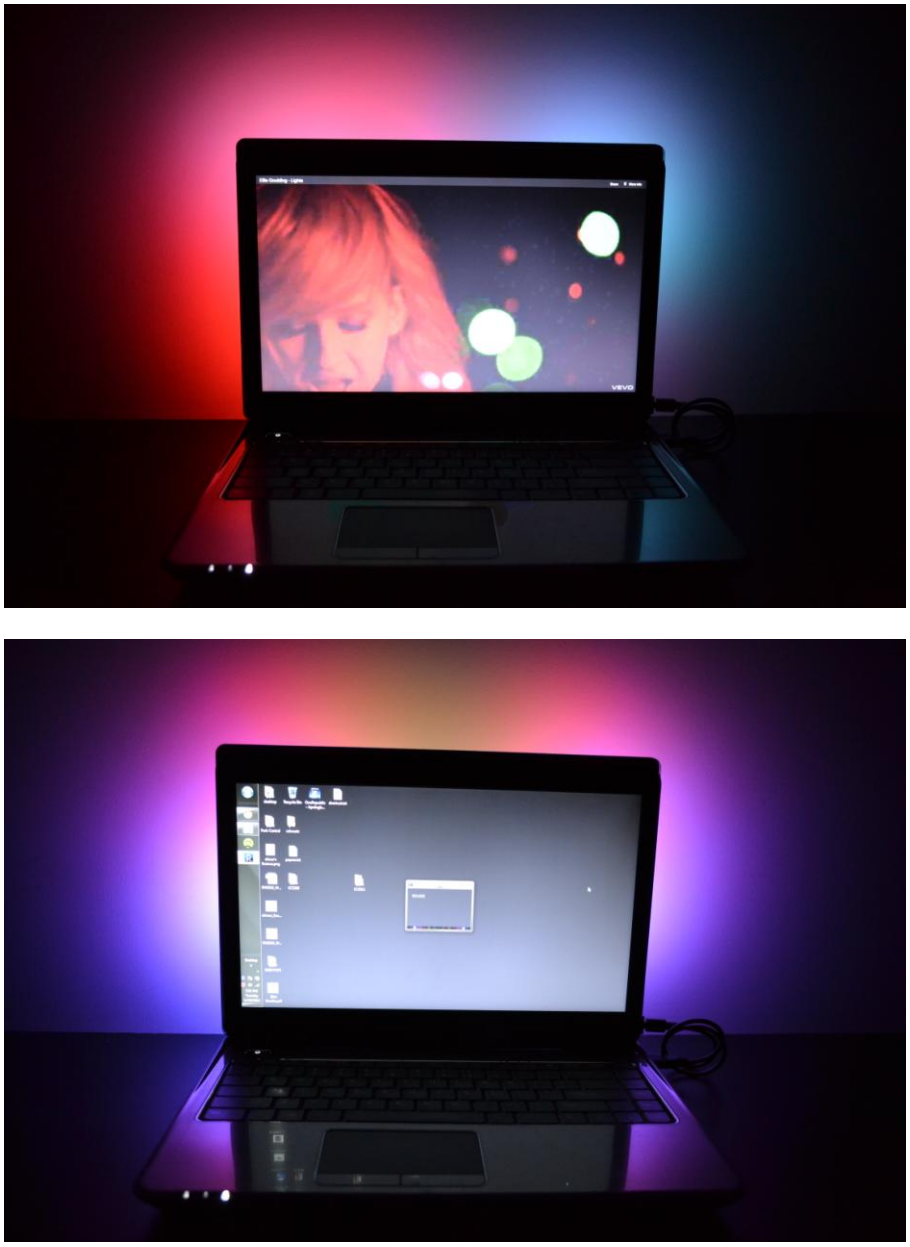


Figure 3: Images of the system in action: (top) Average mode (center) Edge mode (bottom) Sound mode

2. Capturing Screen Data

Screen data is captured using the Java Robot class which is supported on multiple operating systems by Java™ 2 Platform Std. Ed. v1.4.2 [2].

The code for the average and edge mode is only active when sound mode is not selected while the code for the sound mode is only active when the sound mod is selected. This is implemented using if statements around whole code blocks and is intended to prevent FFT calculations from slowing down edge and average mode operations, or screencapture delay from slowing sound mode operations.

A BufferedImage object called screenshot is used to captures a screenshot of the screen.

```
screenshot = robby.createScreenCapture(
  new Rectangle(new Dimension(screenWidth,screenHeight)));
```

The captured screenshot is used for both average and edge mode. For either mode, a for loop goes over every alternate pixel on the captured image (to speed up the process):

```
for(int i =0;i<screenWidth; i=i+2){
  for(int j=0; j<screenHeight;j=j+2){
    pixel = screenshot.getRGB(i,j); //the ARGB integer has the colors of pixel (i,j)
.
.
}
```

Each pixel is of the ARGB 32-bit format where alpha, R, G, B form the four bytes. The R, G, B value is first extracted:

```
r = (int)(255&(pixel>>16)); //add up reds
g = (int)(255&(pixel>>8)); //add up greens
b = (int)(255&(pixel)); //add up blues
```

Followed by which, it is added to a summing bin based on appropriate position constraints:

```
//Bin 0:
screenRed[0]   += r;//(int)(255&(pixel>>16)); //add up reds
screenGreen[0] += g;//(int)(255&(pixel>>8)); //add up greens
screenGreen[0] += b;//(int)(255&(pixel)); //add up blues
if(((5*screenHeight/6)<j)&&(j<(screenHeight))){
  //Bin 1:
  if((0<i)&&(i<(screenWidth/10))){
    screenRed[1]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[1] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[1]  += b;//(int)(255&(pixel)); //add up blues
  }
  //Bin 22:
  if(((9*screenWidth/10)<i)&&(i<(screenWidth))){
    screenRed[22]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[22] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[22]  += b;//(int)(255&(pixel)); //add up blues
  }
}
```

Where screenRed[0], screenGreen[0], screenGreen[0] correspond to the whole screen (no position constraints) and used to calculate the average screen color for the average mode. Bins screenRed[x], screenGreen[x], screenGreen[x] where x is from 1 to 22 correspond to bins for led 1 to 22 with positional constraints based on the screen region corresponding to LED location as shown in Figure 4.

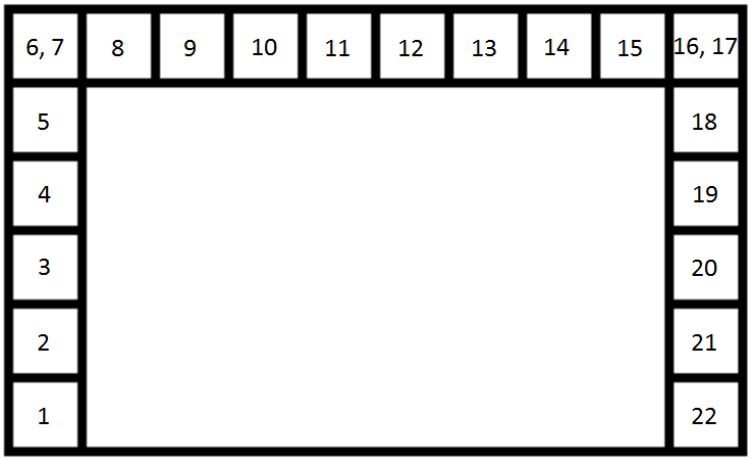


Figure 4: LED number and corresponding screen region.

3. Capturing Sound Data

Throughout the project, several methods were attempted to capture sound data in order to produce a light visualization based on the data. Since the visualization is based on the frequency domain of the sound, all attempts were directed at capturing a FFT spectrum of the sound produced by the computer via the headphone jack. A headphone jack splitter was used to allow the sound to play to headphones or speakers while being analyzed by the system at the same time. Three approaches were attempted for this purpose at circuit, microcontroller and software level.

The first hardware based approach relies on the MSGEQ7 7-channel Graphic Equalizer Display Filter IC [3]. The microcontroller interfaces with the IC using the Strobe, Reset and Output pins as indicated in the datasheet. The audio channel input from the laptop headphone jack is connected to the AudioIn pin via a resistor and capacitor configuration as indicated by the application notes in the datasheet [3].

The protocol for controlling the Reset, Strobe and Output pins were implemented based on the datasheet information.

```
digitalWrite(resetPin, HIGH);
digitalWrite(resetPin, LOW);
for (int i = 0; i < 7; i++)
{
  digitalWrite(strobePin, LOW);
  delayMicroseconds(30); // to allow the output to settle
  spectrumValue[i] = analogRead(analogPin);
  digitalWrite(strobePin, HIGH);
}
```

This approach was tested to be working when the LED circuitry was off. However, turning on the LED lights injected noise into the system via the Ground and Vcc connections. The noise injection resulted in a complete malfunction of the MSGEQ7 based system.

Since slight noise was greatly affecting hardware based frequency analysis, a microcontroller based approach was attempted. The sound was amplified by a LM358AN Op-Amp and analyzed by a second ATMEGA328 microcontroller. The fast FFT library [4] was used to obtain 24 channels of frequency data spaced at 150Hz. This system worked perfectly with the LED lights turned on.

When a powerful speaker was connected to the system via the passive audio jack-splitter mentioned previously, a high frequency sound was audible when no audio was being played from the computer and the LEDs were on. It was observed that this noise was introduced as long as either Audio or GND, or both were connected to the circuit. As a result, attempts to isolate the system to prevent the noise were unsuccessful. The distinctive and constant frequency of the noise suggests that it could be caused by the PWM signals that control each LED.

Finally, since a completely software based approach was attempted. The sound acquisition is performed using the computer soundcard by looping the audio output to the audio input using a standard male-male audio cable.

This approach allows full software control, ability to use any sound input instead of the computer sound output, fast analysis and visualization done in software, and easy customizability of visualization.

For software sound analysis, the Java Minim library [5] is used to acquire sound from the soundcard audio input and perform FFT realtime:

```
void setup()
{
  .
  .
  minim = new Minim(this);
  input = minim.getLineIn();
  fft = new FFT(input.bufferSize(),input.sampleRate());
  .
  .
}
void soundCompute(){
  fft.forward(input.mix);
  .
  .
}
```

The number of bins in the fft is returned by calling `fft.specSize()` and the amplitude of each bin is returned by calling `fft.getBand(i)`. The bins cover a frequency range of 0Hz to 22.05kHz.

4. Sound Visualization

A specific visualization scheme of controlling the 22 LEDs based on the audio data was implemented to demonstrate the system. In the future any visualization scheme can be implemented with ease due to the software structure created by this project.

The frequency spectrum of songs was noticed to be decreasing in a exponential or inverse proportional manner. Thus, the high frequency spectrum amplitude was significantly lower than the low frequency spectrum amplitude. Thus, a linearly proportional multiplier had to be applied to linearize the frequency spectrum.

```
float bandheight = fft.getBand(i)*(i+10)/20; // (i+10)/20 multiplier
```

After the multiplier, the frequency spectrum from 0kHz to 8kHz were binned into 11 bins. The 11 left LEDs and the 11 right LEDs display the same 11 colors for symmetry.

During the binning process, two variables are calculated: arraySum and arrayMax. arraySum calculates the center frequency of the spectrum:

```
for(int i = 0; i < fft.specSize(); i++){
  if(i<110){
    .
    .
    arraySum+=bandheight*i/10; // Find average frequency
  }
}
```

arrayMax calculates the maximum amplitude among the bins:

```
for(int i = 0; i < 11; i++) arrayMax = (soundArray[i]>arrayMax)?
                                        soundArray[i] : arrayMax;
```

The color for each set of 11 LEDs is assigned to a shifting rainbow spectrum that is displaced based on the center frequency arraySum. A rainbow spectrum consists of an infinite loop of colors that transition from Red to Blue to Green to Red again. At any point of time, two-thirds of the rainbow spectrum is assigned to the 11 LEDs in a sliding window manner whereby the displacement of the window is dictated by the center frequency.

Finally, the amplitude of each LED light is assigned to the amplitude of the corresponding frequency bin dynamically scaled to the volume of the music. The dynamic scaling involves a linear scaling by a factor of 255/arrayMax.

```
int amp = soundArray[i]*255/arrayMax;
```

The visualization works best with music genres exhibiting wide ranges of melodies and amplitude.

5. Software-Hardware Communication

As mentioned previously, the communication between the computer and microcontroller happens over the USB serial port. On the software end the Processing Serial library [6] was used. On the microcontroller end, the Arduino Serial library [7] was used.

On the software end, the final data buffers screenRed[], screenGreen[], screenBlue[] are populated with the output buffers from the screen data capture and sound data capture steps based on the current mode (0:average, 1:edge, 2:screen).

```
switch(state){
  case 0:
    for(int i=0; i<23; i++){
      ledRed[i]  =screenRed[0]; //average red
      ledGreen[i]=screenGreen[0]; //average green
      ledBlue[i] =screenBlue[0]; //average blue
    }
    break;
  case 1:
    for(int i=0; i<23; i++){
      ledRed[i]  =screenRed[i]; //average red
      ledGreen[i]=screenGreen[i]; //average green
      ledBlue[i] =screenBlue[i]; //average blue
    }
    break;
  case 2:
    int j;
    for(int i=1; i<23; i++){
      j = (i<12) ? (i-1) : (22-i);
      ledRed[i]  = soundRedArray[j]; //average red
      ledGreen[i]= soundGreenArray[j]; //average green
      ledBlue[i] = soundBlueArray[j]; //average blue
    }
    break;
}
```

Response time was greatly improved by multithreading the software such that the serial communication is handled in a separate thread. The data is sent in packets of five bytes containing the RGB information for one LED in the format:
[0]: 0xff //flag byte
[1]: index //index of corresponding LED (1 to 22)
[2]: R
[3]: G
[4]: B

After the transmission of 22 packets, a 5 millisecond delay is added. The Serial port is set up to a baud rate of 19200. The delay and baud rate were tuned in a trial and error method to determine the shortest delay and highest baud rate possible without losing data. At faster rates or shorter delays, the microcontroller's 80 byte serial buffer was filling up faster than emptying, causing the communication to malfunction.

```
    while (running) {
      byte data[] = new byte[5];
      if(ready){
        for(int index=1; index<23; index++){
          data[0]=(byte)0xff;
          data[1]=(byte)index;
          data[2]=(byte)(ledRed[index]/2);
          data[3]=(byte)(ledGreen[index]/2);
          data[4]=(byte)(ledBlue[index]/2);
          port.write(data);
        }
      }

      //End of thread body
      try {
        sleep((long)(delaytime));
      }
      catch (Exception e) {
      }
    }
```

On the microcontroller side, the packets are decoded in a similar fashion.

```
int index = 0;
char buffer[4];
while(index<22){  //recieve all the led data first
  if (Serial.available()>=5) {
    /*if statement automatically clears wrong values*/
    if(Serial.read() == 0xff){
      Serial.readBytes(buffer,4);
      index = buffer[0];//Serial.read();
      red[index]   = buffer[1];//Serial.read();
      green[index] = buffer[2];//Serial.read();
      blue[index]  = buffer[3];//Serial.read();
    }
  }
}
```

6. LED Control

The Adafruit addressable LED strip [8] is used for this project. The LED strip is powered by 5 volts connected to it's Vcc and GND pins. The LED strip contains a controller LPD8806 IC between LED that function as shift registers to allow individual addressability. The PWM and regulator for each LED is built-in to an IC in each LED. The data is shifted in via the DI pin as the CLK pin is clocked. These operations are handled by the provided LPD8806 library.

```
for (int i=0; i<strip.numPixels(); i++) {
   strip.setPixelColor(i, strip.Color(red[i+1], green[i+1], blue[i+1]));
}
```

The pin connections between the LED strip and microcontroller is illustrated in Figure 5.

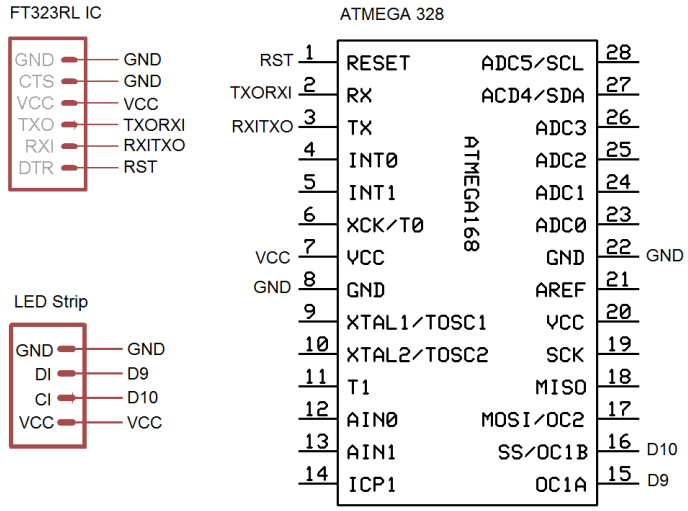


Figure 5: Pin connections between LED strip and microcontroller.

7. Mechanical Design

The front side of the frame is made of plywood for aesthetic value. The back side of the frame is made of acrylic to display components. The front and back side of the frame is shown in Figure 6.

Figure 6: Front and back side of frame.

The frame was laser-cut using the Epilog Helix 350 laser cutter at the Champaign-Urbana Community Fabrication Laboratory. Frosted masking tape was used on the LED outlets as light diffusing element.

8. Conclusion

The system works as it was intended to. However, several improvements can still be made to the system. Firstly, movies ultra-widescreen formats usually display on computers with a black bar on top and bottom. As a result, the top LEDs do not display any useful information. The user interface can be improved to allow users to define a virtual rectangle to represent the screen rather than the actual screen. Secondly, the sound visualization code can be abstracted to a class so that sound visualizations can be easily customized by inheriting from the proposed class or other classes.

I would like to thank the University of Illinois Advanced Digital Projects Lab and the Champaign-Urbana Community Fabrication Laboratory for providing me with the tools and resources that made this project possible. I would also like to thank Zuofu Cheng, Skot P. Wiedmann and Prof.Lippold Haken for their technical insight on this project.

9. Citations

[1] Combined video and audio based ambient lighting control. Erik Nieuwlands. Koninklijke Philips Electronics <http://www.google.com/patents/US20100265414>

[2] Class Robot. JavaTM 2 Platform Std. Ed. v1.4.2 <http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/Robot.html>

[3] Seven Band Graphic Equalizer Datasheet. Micro-Star International Co. <https://www.sparkfun.com/datasheets/Components/General/MSGEQ7.pdf>

[4] Fast Fourier Transform. Arduinoos. <http://www.arduinoos.com/2010/10/fast-fourier-transform-fft/>

[5] Class Minim. Damien Di Fede .<http://code.compartmental.net/minim/javadoc/ddf/minim/Minim.html>

[6] Serial. Processing. <http://processing.org/reference/libraries/serial/index.html>

[7] Serial. Arduino. <http://arduino.cc/en/Reference/serial>

[8] Digital RGB LED Weatherproof Strip. Adafruit. <http://adafruit.com/products/306>

10. Appendix

Final software code:

```
// PC Ambient Lighting System
// Copyright: Rajarshi Roy 2012

// Libraries for screen capture //
import java.awt.Robot; //java library that lets us take screenshots
import java.awt.AWTException;
import java.awt.event.InputEvent;
import java.awt.image.BufferedImage;
import java.awt.Rectangle;
import java.awt.Dimension;

// Libraries for sound processing //
import ddf.minim.analysis.*;
import ddf.minim.*;

// Library for Serial communication with Arduino //
import processing.serial.*;

// Objects for screen capture //
Robot screenImage;
BufferedImage screenshot;

// Objects for sound processing //
Minim minim;
AudioInput input;
FFT fft;

// Objects for Serial communication with Arduino //
Serial port; //creates object "port" of serial class
serialThread serialArduino;

// Arrays for screen data
float screenRed[] = new float[23];
float screenGreen[] = new float[23];
float screenBlue[] = new float[23]; //red, green and blue values

// Arrays for sound data
int[] soundArray = new int[11];
int arrayMax;
int arraySum;
int arrayAvg;
// amplitude*color:
int[] soundRedArray = new int[11];
int[] soundGreenArray = new int[11];
int[] soundBlueArray = new int[11];

// Arrays for data transfer
float ledRed[] = new float[23];
float ledGreen[] = new float[23];
float ledBlue[] = new float[23]; //red, green and blue values
```

```
// State variable that controls mode
// 0: screen average
// 1: screen edge
// 2: music
int state;

void setup()
{
  frame.setTitle("Lumen");
  size(220, 140); // window size
  background(20);
  fill(255);
  text("AVERAGE", 15, 30);
  state = 0; // mode
  // Setup screen capture //
  try //standard Robot class error check
  {
    screenImage = new Robot();
  }
  catch (AWTException e)
  {
    println("Robot class not supported by your system!");
    exit();
  }

  // Setup sound processing //
  minim = new Minim(this);
  input = minim.getLineIn();
  fft = new FFT(input.bufferSize(),input.sampleRate());

  // Setup serial communication //
  port = new Serial(this, Serial.list()[0],19200); //set baud rate
  serialArduino = new serialThread(5,"arduino");
  serialArduino.start();

}

void mousePressed() {
  if(state<2) state++;
  else state=0;
  background(20);
  fill(255);
  switch(state){
    case 0:
      text("AVERAGE", 15, 30);
      break;
    case 1:
      text("EDGE", 15, 30);
      break;
    case 2:
      text("SOUND", 15, 30);
      break;
  }
}
```

```
void draw()
{
  if((state==0)||(state==1)) screenCompute();
  if(state==2) soundCompute();
  serialArduino.ready = false;
  switch(state){
    case 0:
      for(int i=0; i<23; i++){
        ledRed[i]  =screenRed[0]; //average red
        ledGreen[i]=screenGreen[0]; //average green
        ledBlue[i] =screenBlue[0]; //average blue
      }
      break;
    case 1:
      for(int i=0; i<23; i++){
        ledRed[i]  =screenRed[i]; //average red
        ledGreen[i]=screenGreen[i]; //average green
        ledBlue[i] =screenBlue[i]; //average blue
      }
      break;
    case 2:
      int j;
      for(int i=1; i<23; i++){
        j = (i<12) ? (i-1) : (22-i);
        ledRed[i]  = soundRedArray[j]; //average red
        ledGreen[i]= soundGreenArray[j]; //average green
        ledBlue[i] = soundBlueArray[j]; //average blue
      }
      break;
  }
  serialArduino.ready = true;
  for(int i=1; i<23; i++){fill(ledRed[i],ledGreen[i],ledBlue[i]); rect(10*(i-1), 130,
10, 10, 3, 3);}
}



void screenCompute(){
  for(int i =0; i<23; i++) {screenRed[i]=0; screenGreen[i]=0; screenBlue[i]=0;}

  //sets of 8 bytes are: Alpha, Red, Green, Blue
  int pixel,r,g,b; //ARGB variable with 32 int bytes where

  //get screenshot into object "screenshot" of class BufferedImage
  screenshot = screenImage.createScreenCapture(new Rectangle(new
Dimension(screenWidth,screenHeight)));

  //I skip every alternate pixel making my program 4 times faster
  for(int i =0;i<screenWidth; i=i+2){
    for(int j=0; j<screenHeight;j=j+2){
      pixel = screenshot.getRGB(i,j); //the ARGB integer has the colors of pixel
(i,j)
      r = (int)(255&(pixel>>16)); //add up reds
```

```
g = (int)(255&(pixel>>8)); //add up greens
b = (int)(255&(pixel)); //add up blues
screenRed[0]   += r;//(int)(255&(pixel>>16)); //add up reds
screenGreen[0] += g;//(int)(255&(pixel>>8)); //add up greens
screenBlue[0]  += b;//(int)(255&(pixel)); //add up blues

if(((5*screenHeight/6)<j)&&(j<(screenHeight))){
  if((0<i)&&(i<(screenWidth/10))){
    screenRed[1]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[1] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[1]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((9*screenWidth/10)<i)&&(i<(screenWidth))){
    screenRed[22]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[22] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[22]  += b;//(int)(255&(pixel)); //add up blues
  }
}
if(((4*screenHeight/6)<j)&&(j<(5*screenHeight/6))){
  if((0<i)&&(i<(screenWidth/10))){
    screenRed[2]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[2] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[2]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((9*screenWidth/10)<i)&&(i<(screenWidth))){
    screenRed[21]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[21] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[21]  += b;//(int)(255&(pixel)); //add up blues
  }
}
if(((3*screenHeight/6)<j)&&(j<(4*screenHeight/6))){
  if((0<i)&&(i<(screenWidth/10))){
    screenRed[3]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[3] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[3]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((9*screenWidth/10)<i)&&(i<(screenWidth))){
    screenRed[20]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[20] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[20]  += b;//(int)(255&(pixel)); //add up blues
  }
}
if(((2*screenHeight/6)<j)&&(j<(3*screenHeight/6))){
  if((0<i)&&(i<(screenWidth/10))){
    screenRed[4]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[4] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[4]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((9*screenWidth/10)<i)&&(i<(screenWidth))){
    screenRed[19]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[19] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[19]  += b;//(int)(255&(pixel)); //add up blues
  }
}
if(((screenHeight/6)<j)&&(j<(2*screenHeight/6))){
```

```
  if((0<i)&&(i<(screenWidth/10))){
    screenRed[5]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[5] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[5]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((9*screenWidth/10)<i)&&(i<(screenWidth))){
    screenRed[18]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[18] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[18]  += b;//(int)(255&(pixel)); //add up blues
  }
}
if((0<j)&&(j<(screenHeight/6))){
  if((0<i)&&(i<(screenWidth/10))){
    screenRed[6]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[6] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[6]  += b;//(int)(255&(pixel)); //add up blues
    screenRed[7]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[7] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[7]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((screenWidth/10)<i)&&(i<(2*screenWidth/10))){
    screenRed[8]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[8] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[8]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((2*screenWidth/10)<i)&&(i<(3*screenWidth/10))){
    screenRed[9]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[9] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[9]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((3*screenWidth/10)<i)&&(i<(4*screenWidth/10))){
    screenRed[10]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[10] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[10]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((4*screenWidth/10)<i)&&(i<(5*screenWidth/10))){
    screenRed[11]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[11] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[11]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((5*screenWidth/10)<i)&&(i<(6*screenWidth/10))){
    screenRed[12]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[12] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[12]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((6*screenWidth/10)<i)&&(i<(7*screenWidth/10))){
    screenRed[13]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[13] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[13]  += b;//(int)(255&(pixel)); //add up blues
  }
  if(((7*screenWidth/10)<i)&&(i<(8*screenWidth/10))){
    screenRed[14]   += r;//(int)(255&(pixel>>16)); //add up reds
    screenGreen[14] += g;//(int)(255&(pixel>>8)); //add up greens
    screenBlue[14]  += b;//(int)(255&(pixel)); //add up blues
  }
```

```
        if(((8*screenWidth/10)<i)&&(i<(9*screenWidth/10))){
          screenRed[15]   += r;//(int)(255&(pixel>>16)); //add up reds
          screenGreen[15] += g;//(int)(255&(pixel>>8)); //add up greens
          screenBlue[15]  += b;//(int)(255&(pixel)); //add up blues
        }
        if(((9*screenWidth/10)<i)&&(i<(screenWidth))){
          screenRed[16]   += r;//(int)(255&(pixel>>16)); //add up reds
          screenGreen[16] += g;//(int)(255&(pixel>>8)); //add up greens
          screenBlue[16]  += b;//(int)(255&(pixel)); //add up blues
          screenRed[17]   += r;//(int)(255&(pixel>>16)); //add up reds
          screenGreen[17] += g;//(int)(255&(pixel>>8)); //add up greens
          screenBlue[17]  += b;//(int)(255&(pixel)); //add up blues
        }
      }

    }  //end of for loop
  } //end of for loop

  // Normalize screen average //
  screenRed[0]  =screenRed[0]/((screenWidth/2)*(screenHeight/2)); //average red
  screenGreen[0]=screenGreen[0]/((screenWidth/2)*(screenHeight/2)); //average green
  screenBlue[0] =screenBlue[0]/((screenWidth/2)*(screenHeight/2)); //average blue

  // Normalize edge averages //
  for(int i=1; i<23; i++){
    screenRed[i]  =screenRed[i]/((screenWidth/20)*(screenHeight/12)); //average red
    screenGreen[i]=screenGreen[i]/((screenWidth/20)*(screenHeight/12)); //average
green
    screenBlue[i] =screenBlue[i]/((screenWidth/20)*(screenHeight/12)); //average blue
  }
}



void soundCompute(){
  fft.forward(input.mix);

  for(int i = 0; i < 11; i++) soundArray[i] = 0;
  arrayMax = 0;
  arraySum = 0;
  for(int i = 0; i < fft.specSize(); i++){
    float bandheight = fft.getBand(i)*(i*1+10)/20;
    if(i<110){
      soundArray[i/10]+=bandheight;
      arraySum+=bandheight*i/10;  // Find average frequency
    }
  }

  // Compute and place lower limit on array max
  for(int i = 0; i < 11; i++) arrayMax = (soundArray[i]>arrayMax)? soundArray[i] :
arrayMax;
  arrayMax = (arrayMax<20) ? 2 : arrayMax;

  // Compute and place lower limit on array average
  for(int i = 0; i < 11; i++) arrayAvg+=soundArray[i];
```

```
  arrayAvg /= 11;
  arrayAvg = (arrayAvg<10) ? 1 : arrayAvg;

  for(int i=0; i <11; i++){
    int amp = soundArray[i]*255/arrayMax;
    soundRedArray[i] = getRed(511/11*i)*amp/255;
    soundGreenArray[i] = getGreen(511/11*i)*amp/255;
    soundBlueArray[i] = getBlue(511/11*i)*amp/255;
 }
}

int getBlue(int x){
  x = (x-arraySum/50-10) % 768; // color swing
  x = (x>0)? x : 768+x;
  int y;
  if((0 <=x)&&(x<= 255)){y = 255-x;}
  else if((256 <=x)&&(x<= 511)){y = 0;}
  else {y = x-512;}
  return y;
}

int getRed(int x){
  x = (x-arraySum/50-10) % 768; // color swing
  x = (x>0)? x : 768+x;
  int y;
  if((0 <=x)&&(x<= 255)){y = x;}
  else if((256 <=x)&&(x<= 511)){y = 511-x;}
  else {y = 0;}
  return y;
}

int getGreen(int x){
  x = (x-arraySum/50-10) % 768; // color swing
  x = (x>0)? x : 768+x;
  int y;
  if((0 <=x)&&(x<= 255)){y = 0;}
  else if((256 <=x)&&(x<= 511)){y = x-256;}
  else {y = 767-x;}
  return y;
}


/* Sends data to the microcontroller */
class serialThread extends Thread {
  boolean running;           // Is the thread running?  Yes or no?
  int delaytime;             // Enough delay to let data to be ready
  String id;                 // Thread name
  boolean ready;             // True when data is ready

  serialThread (int w, String s) {
    delaytime = w;
    running = false;
    id = s;
        //port1 = new Serial(this, Serial.list()[0],19200); //set baud rate
  }
```

```
  void start () {
    // Set running equal to true
    running = true;
    // Print messages
    println("Starting thread (will execute every " + delaytime + " milliseconds.)");
    // Do whatever start does in Thread, don't forget this!
    super.start();
  }

  void run () {
    while (running) {
      byte data[] = new byte[5];
      if(ready){
        for(int index=1; index<23; index++){
          data[0]=(byte)0xff;
          data[1]=(byte)index;
          data[2]=(byte)(ledRed[index]/2);
          data[3]=(byte)(ledGreen[index]/2);
          data[4]=(byte)(ledBlue[index]/2);
          port.write(data);
        }
      }

      //End of thread body
      try {
        sleep((long)(delaytime));
      }
      catch (Exception e) {
      }
    }
    println(" thread is done!");  // The thread is done when we get to the end of
run()
  }

  void quit() {
    println("Quitting.");
    running = false;  // Setting running to false ends the loop in run()
    // In case the thread is waiting. . .
    interrupt();
  }
}
```

Final Microcontroller code:

```
// PC Ambient Lighting System
// Copyright: Rajarshi Roy 2012

/* Include LED Strip Library and SPI Library */
#include "LPD8806.h"
#include "SPI.h"

/* Array containing LED color information received:
```

```
*  from PC.
*  red[0] , green[0] , blue[0]  contains average color of whole screen
*  red[1] , green[1] , blue[1]  contains average color of left-bottom LED
*  red[22], green[22], blue[22] contains average color of right-bottom LED
*/
int red[23];
int green[23];
int blue[23]; //red, green and blue values

/* Setting up LED library class with parameters*/
// Number of RGB LEDs in strand:
int nLEDs = 22;
// Chose 2 pins for output; can be any valid output pins:
int dataPin  = 9;
int clockPin = 10;
// Create LED strip object called strip
LPD8806 strip = LPD8806(nLEDs, dataPin, clockPin);



void setup()
{
  Serial.begin(19200);
  // Start up the LED strip
  strip.begin();
  // Update the strip, to start they are all 'off'
  strip.show();
}



void loop()
{
  int index = 0;
  char buffer[4];
  while(index<22){  //recieve all the led data first
    if (Serial.available()>=5) {
      /* if statement automatically clears wrong values */
      if(Serial.read() == 0xff){
        Serial.readBytes(buffer,4);
        index = buffer[0];//Serial.read();
        red[index]   = buffer[1];//Serial.read();
        green[index] = buffer[2];//Serial.read();
        blue[index]  = buffer[3];//Serial.read();
      }
    }
  }

  // Set each LED color to each portion of screen
  for (int i=0; i<strip.numPixels(); i++) {
    strip.setPixelColor(i, strip.Color(red[i+1], green[i+1], blue[i+1]));
  }

  strip.show();
}
```