**Computer Graphics Course Project:**

**Music synchronised Stage Lights**

*Authored by*

|  |  |
| --- | --- |
| Divyanshu Solanki | 160100070 |
| Pragun Badhan | 160100114 |
| Rohit Sencha | 160100068 |
| Tanya Gupta | 150040010 |

*Guided by*

**Prof S.S Pande**



Department of Mechanical Engineering  
Indian Institute of Technology Bombay

Mumbai 400076

5 November 2019

**Abstract**

The work starts with a motto of cutting of the need of a light controlling person in any public concert event or indoor party. The task done by Lightman includes varying the colour, intensity, direction. To automate this process, we first extract characteristics of sound and then send the output to light controlling device.

The audio analysis is done on Python which involves decomposing the audio into harmonic and percussive components and then using these components to determine the beats and the tempo. The chromogram and spectrogram are used to visualise various pitches and instruments in the audio and their onset along with the beats.

The results of the audio analysis can be watched in any rendering software. This project makes use of **Unreal Engine for stage modelling and rendering**. In rendering, sound is divided in packets of small-time intervals (0.1 sec) and then are given to an FFT spectrum analyser which outputs an array of amplitudes of different frequencies.

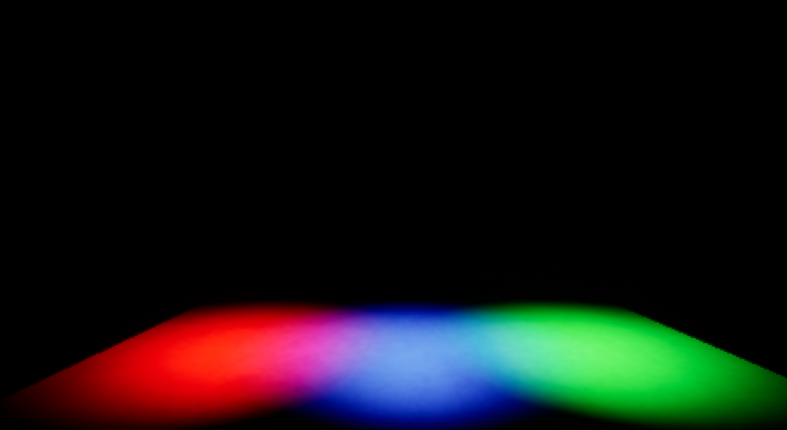
**Introduction**

It is important to apply and understand the theoretical knowledge learnt from the course and how it applies practically in real life. The Computer Graphics and Product Modelling course has been important for us to understand how a designer can interactively synthesize various product shapes, visualise them in different settings and analyse their functional performance.

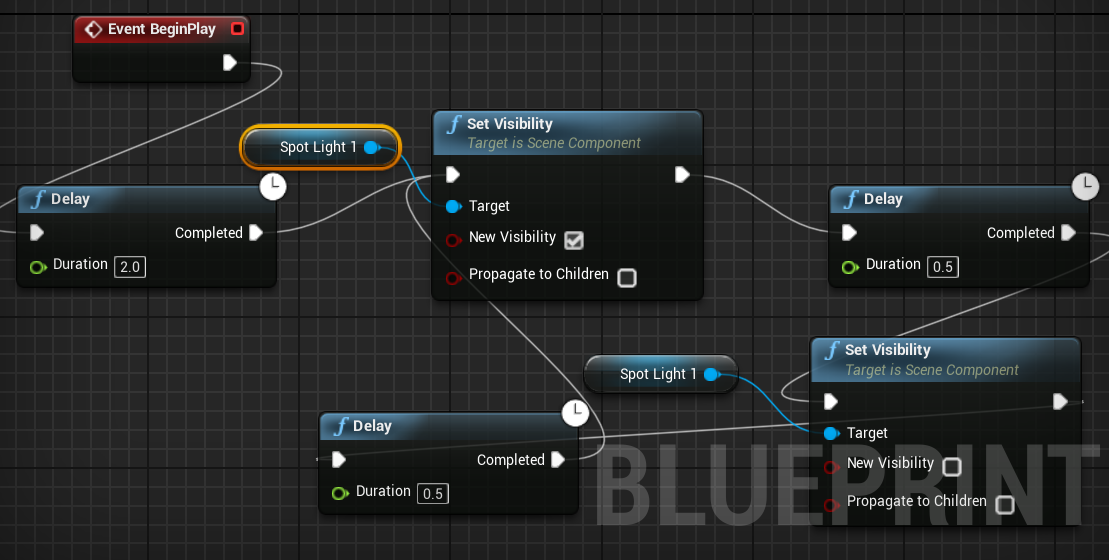
When it comes to making the lights dance in sync with the music, the current scenario is that there is a person who controls the lights, its movement and its intensity in association with the music. We would like to remove this human part and automate the process. This will help reduce costs and remove errors.

The basic difficulties for analysing audio for our problem statement is the extremely complex waveform of signals in a song. The differentiation of the instruments, detection of beats and identifying various tempos hence becomes a daunting task. Extensive literature exists in this domain which reports their working on these by decomposing and analysing the harmonic and percussive components of audio. The FFTs of these components and their spectrograms have been reported to give good estimates of beats and tempos. Few libraries in Python exist which computationally utilise this literature to find the beats and tempos. Librosa library of Python was used in this project for the analysis.

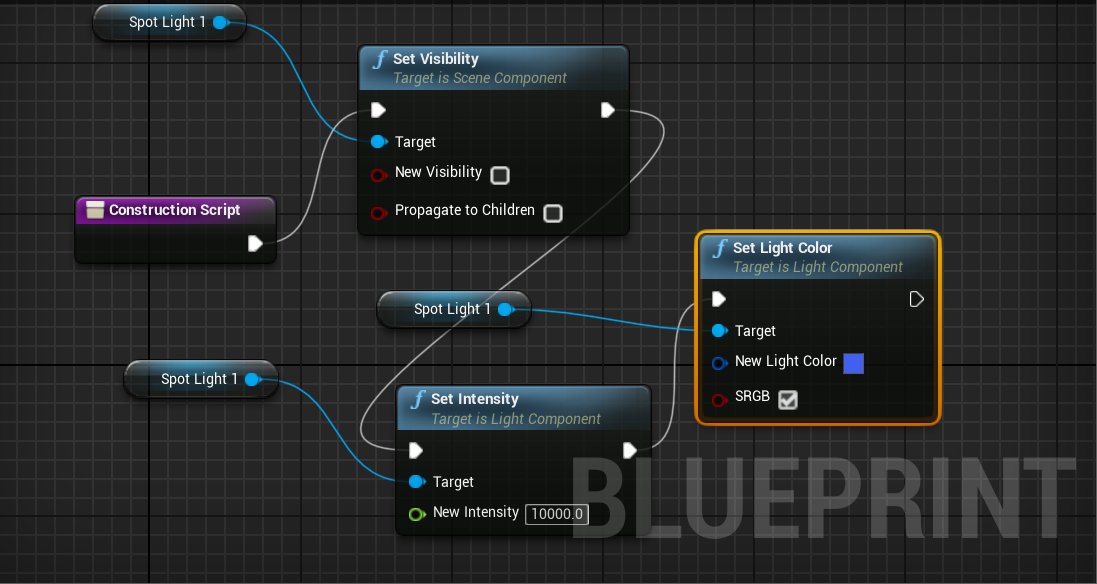
First, we started with a basic understanding of the software Unreal Engine itself and implemented a basic stage light control using time delay. This would have to be predefined for every audio file we play, and hence it is cumbersome and time consuming. Three spotlights of red, blue green colour are selected.



The blueprint for a single spotlight is shown below. We can see it starts playing 2 seconds after we start. The lights play at 0.5 second interval of each other in a loop. Each light remains on for 0.5 seconds and then off for 1 second. These values have been in respect to the audio file chosen, since the time characteristics will vary between slow/fast songs, etc.



In the construction script we define the characteristics of the spotlight, the one shown below is for setting the intensity of light and the colour of the light.



Blueprints are similarly designed for green and red spotlights and the resulting environment is run, but unfortunately as predicted we see that as soon as we change the audio file, the current settings become invalid and we need to redefine the time settings. Thus, we see the need for music synchronised lights and have consequently done the required modelling.

**Project Design**

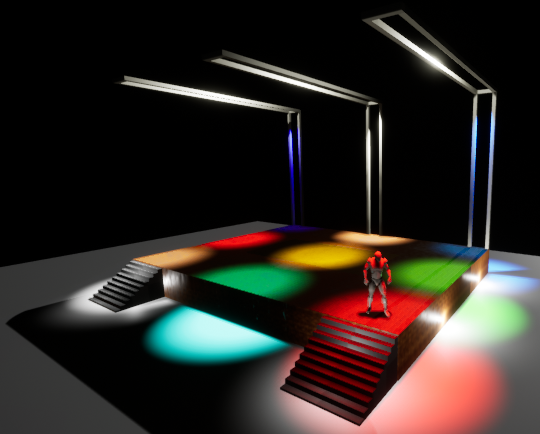
1. **Stage Modelling**

Environment (Level) was modelled in Unreal Engine 4.23. Level unit consists of a floor for audience, a raised stage for singer and 3\*3 spotlight grid whose intensity changes in real time as per characteristics of music. A singer was imported from existing templates and has basic movements like walking and jumping.

Other components of stage include:

* Three overhead ceiling bars to support spotlights
* Static illumination lights around the platform and on ceiling for environment illumination
* Two entry staircases

The stage looks as shown below



To reduce graphics rendering load on GPU, **LightMass Importance Volume** utility of Unreal was used. The Lightmass Importance Volume controls the volume which requires details lighting. Anything outside of this volume is poorly rendered. Good rendering means multiple light reflections are calculated and on the other hand, poor rendering means only one reflection is considered significant.

1. **Lights**

Unreal Engine allows us to use different lights according to our needs and control its position and settings accordingly.

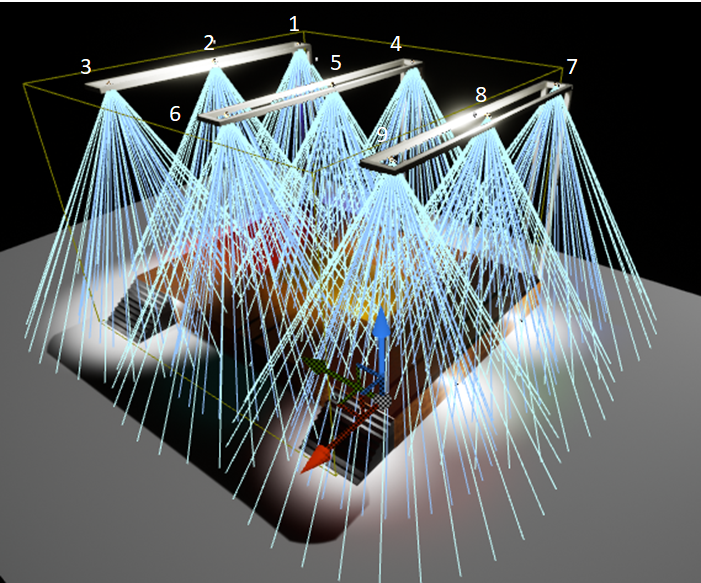
* It provides for 3 types of lights. From left to right they are: **Point Light, Spot Light,** and **Directional Light**



* **Intensity** determines how much energy the light outputs into the scene.
* **Light Color** will adjust the color of the light and the sprite that represents the light in the editor will change its color to match.
* **Attenuation Radius** of the light sets the reach of the light and defines what objects it will affect
* **Source Radius** and **Source Length** define the size of specular highlights on surfaces.

The word ‘static lights’ used before refers to those lights whose intensity remains constant throughout the simulation. Using static lights saves lots of computing load. When a light is declared static, the detailed lighting of environment is computed and then all that information of illumination is hard coded on texture of other materials giving a feel as if there is real light scattering. So calculation for static light rendering is done once only during simulation in unreal editor. So, when we load the program there is no need of real time light rendering calculation for static lights.

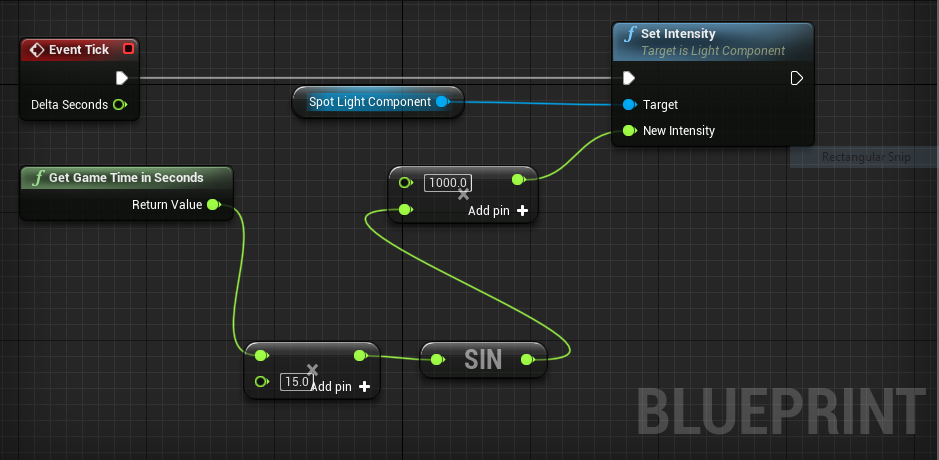
Below image shows the outer and inner cones of spotlight. The surrounding yellow coloured box is LightMass Importance Volume box. Outer cone and inner cone angles of spotlights refers to the solid angle at which they emit lights.



1. **Spotlight Animation Design**

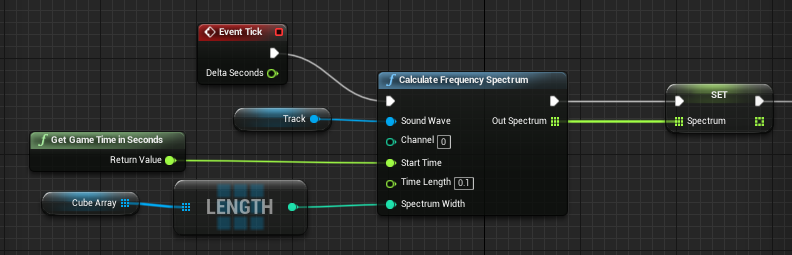
Blueprint allows for visual scripting of simulation events.

Image represents blueprint for a blinking spotlight.

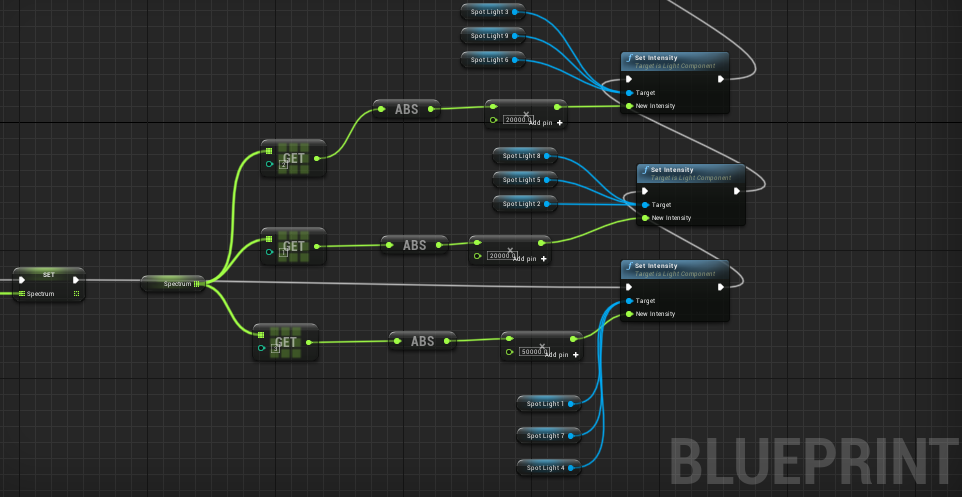


At each instant, **Event Tick** module asks **Set Intensity** module to set intensity of spotlight based on inputs to **New Intensity** value. **Get Game Time in Seconds** sends game time as input to a **multiplier** module. The output is then fed to **sin** function and then to another **multiplier**. **Sin** function enables fluctuating nature of light and multipliers alters frequency of fluctuations and boost intensity of light.

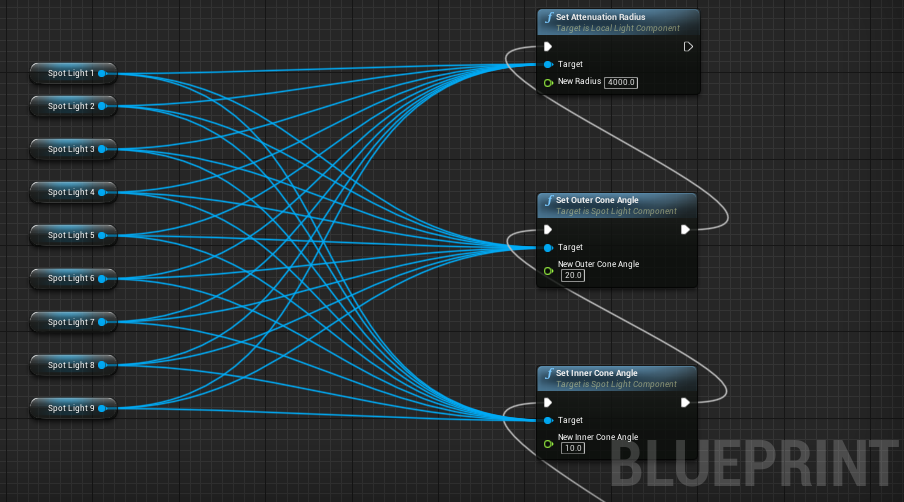
To achieve their intensity control based on song, following scripting was done



The **SET** command stores the calculated FFT spectrum data in an 2d array. We then send this data to intensity of different spotlights in real time. The scripting shown below is in continuation with above image.



The **Outer cone angle** and **Inner cone angle** of spotlights is also controlled with below scripting using spotlight properties. Below shown image is continuation of above image.



The white line running through most of modules shows the functions which run during each Event Tick. We call them primary functions. Event Tick can be understood as series of pulsating input which asks other function to execute their task periodically. Other functions attached to these primary functions are called secondary function and gets called as per requirement of primary functions

**Audio Analysis/ Interpretation**

The analysis of audio was performed in MATLAB and then, due to limited functionalities of MATLAB and open-source nature of Python, using the Librosa library of Python. The audio analysis was done using the various functionalities of Librosa and established techniques of sound/audio processing. Details of each of these is given below.

**Mel Spectrogram**

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. A Mel scale is a widely used scale in music which converts the frequencies (Hz) to mels as:

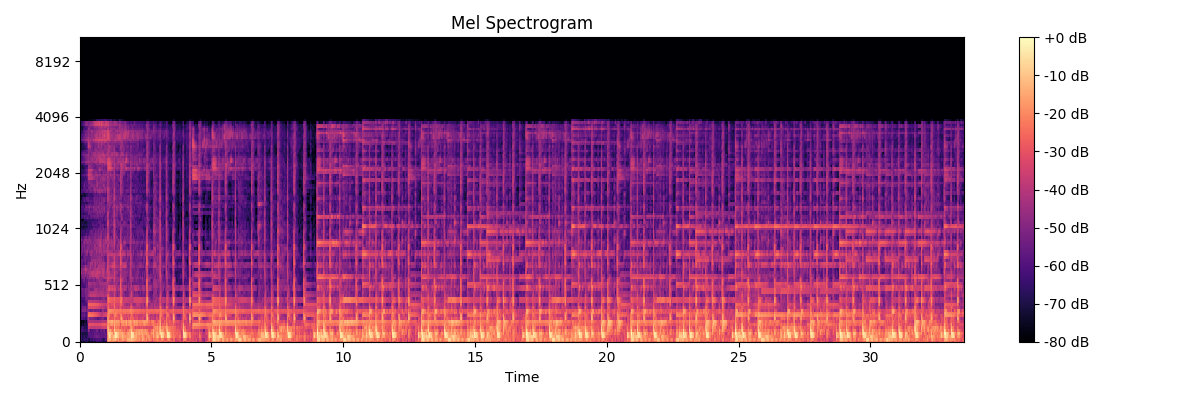
A Mel Spectrogram shows which frequencies occur in the audio and at which time with what power/intensity. Hence it is a 3D graph and Librosa uses colour gradients to show the power values. The commands below create the Mel Spectrogram for an audio file with Librosa and Matplotlib libraries.

audio\_path = 'sample.wav'

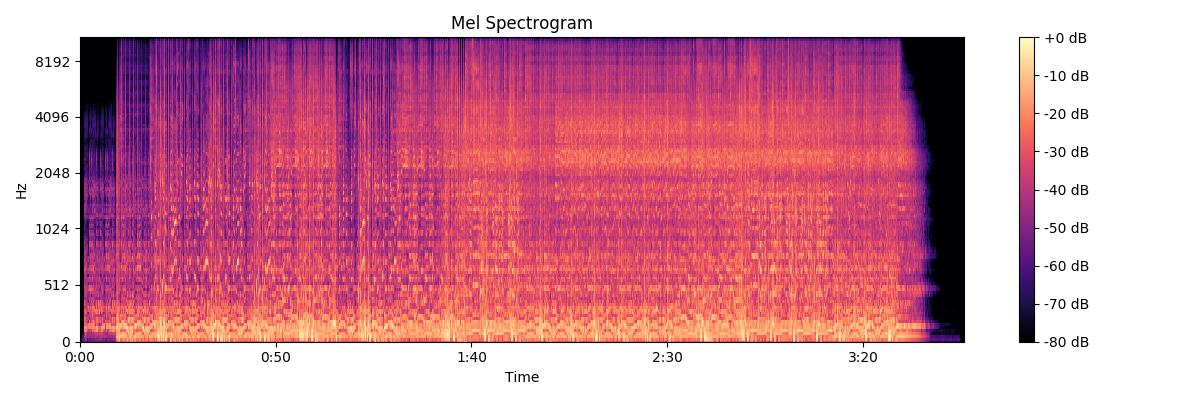
y, sr = librosa.load(audio\_path)

S = librosa.feature.melspectrogram(y, sr=sr, n\_mels=128)

The spectrogram plot for a 35 second sample audio file generated is shown.



The spectrogram plot for Nirvana’s 1992 song “Come As You Are” is shown below.



**Separating Harmonic and Percussion components of audio**

The total spectrogram of the audio can be divided into two components, namely harmonic and percussive components. The percussive components of the audio correspond to the sound from two colliding objects, for example, drums. Percussive sounds do not have a pitch but a very clear localization in time. Harmonic sounds are perceived to have a certain pitch, for example a violin sound. Usually, a note played on an instrument (say piano) has onset of percussion (hammer strung on piano strings) followed by a harmonic tone (from the vibration of the strings)

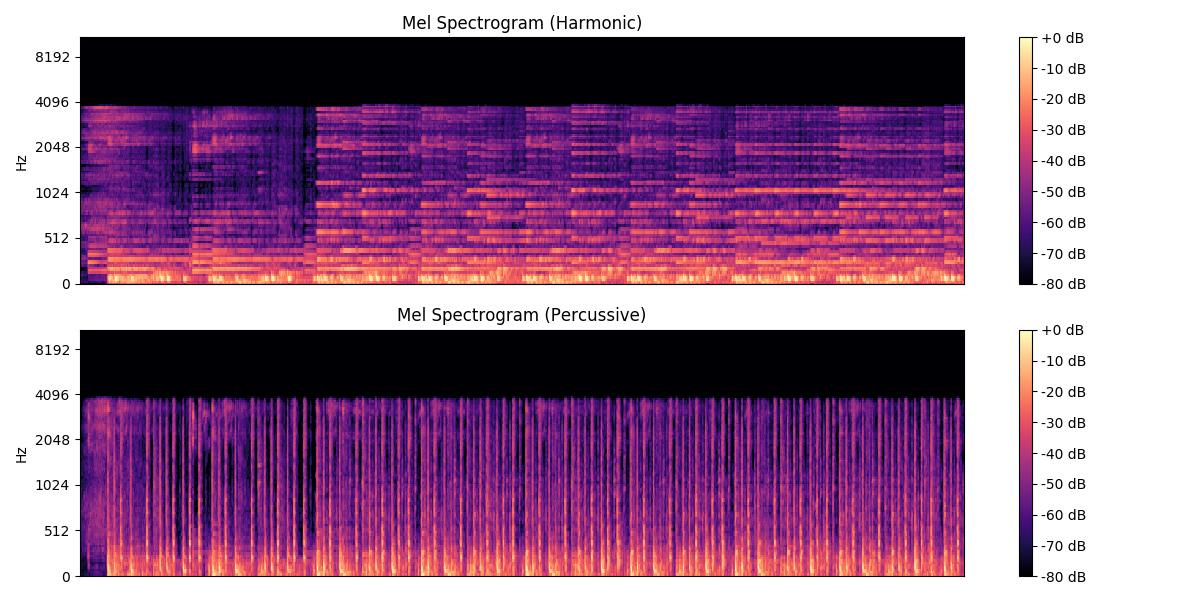
In a spectrogram, harmonic sounds form horizontal structures (in time direction) while percussive sounds form vertical structures (in frequency direction). The analysis of these decomposed components of the audio can be effectively utilised for determining the beats, tempo and when a certain pitch (or instrument) starts. The library Librosa provides for decomposition of the audio in the two components. The commands for creating spectrograms for the two are given below.

y\_harmonic, y\_percussive = librosa.effects.hpss(y)

S\_harmonic = librosa.feature.melspectrogram(y\_harmonic, sr=sr)

S\_percussive = librosa.feature.melspectrogram(y\_percussive, sr=sr)

The spectrograms for harmonic and percussions (for the 35 sec sample audio) are shown below.



**Beat tracking and Tempo**

Beat tracking simply means finding the time stamps where a listener “taps feet”, which is either the onset from a musical instrument which is regularly spaced as a part of the musical rhythm. The library Librosa used in our code detects the beat and tempo by i.) measuring the onset strength ii.) estimating tempo from correlating the onset strength iii.) pick peaks approximately in the tempo and checking the strength of peak. Since the onset determines the beat, the percussive components are used for tracking the beat.

The command for beat tracking and tempo is given below.

tempo, beats = librosa.beat.beat\_track(y=y\_percussive, sr=sr)

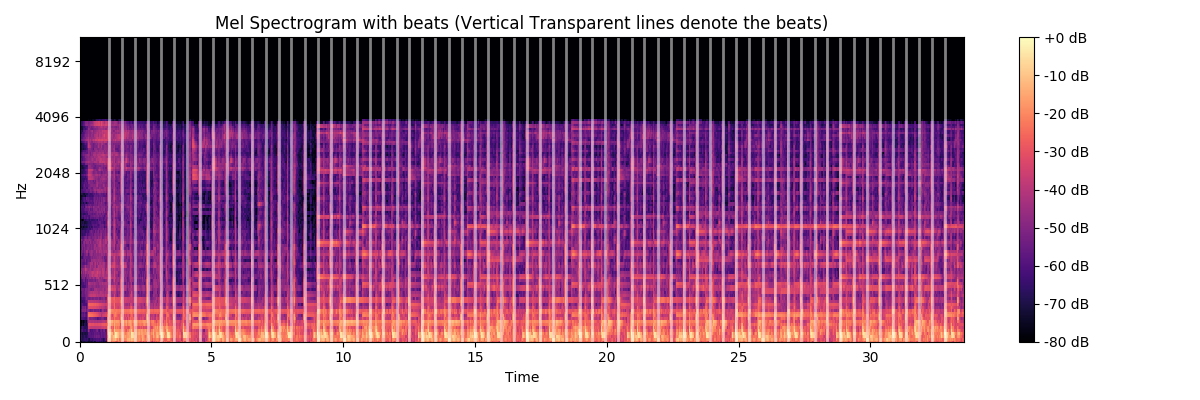
The output for the sample file comes out as:

Estimated tempo: 123.05 BPM

First 5 beat frames: [ 48 69 90 112 133]

First 5 beat times: [1.11455782 1.60217687 2.08979592 2.60063492 3.08825397]

The spectrogram with beats shown (transparent vertical lines) is generated and shown below.



**Chromagram**

A chromagram represents what pitch classes belong in a sound and its various components. A pitch class is the set of all pitches which are a whole number of octaves apart. A chromagram is a 3-D graph showing the power/intensity of each pitch class at a time instant in the audio sample. This is particularly useful for us as the onset of an instrument can be tracked in the chromogram and correspondingly the light can be moved and the duration for which it is lit.

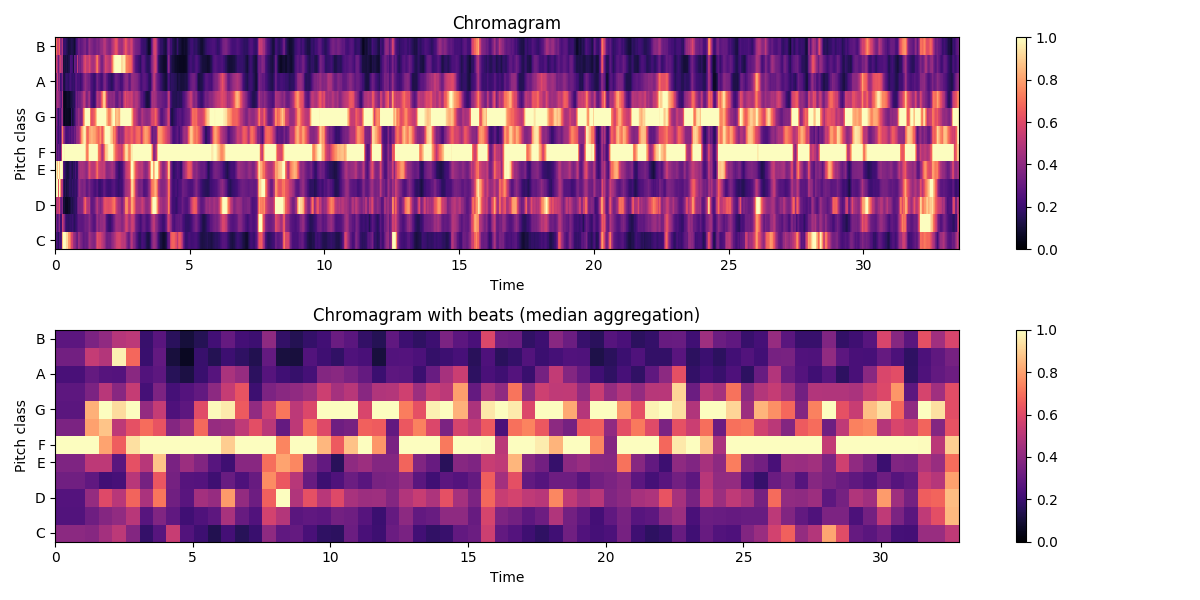
In the code the following command is used for creating a chromogram.

C=librosa.feature.chroma\_cqt(y=y\_harmonic,sr=sr, bins\_per\_octave=36)

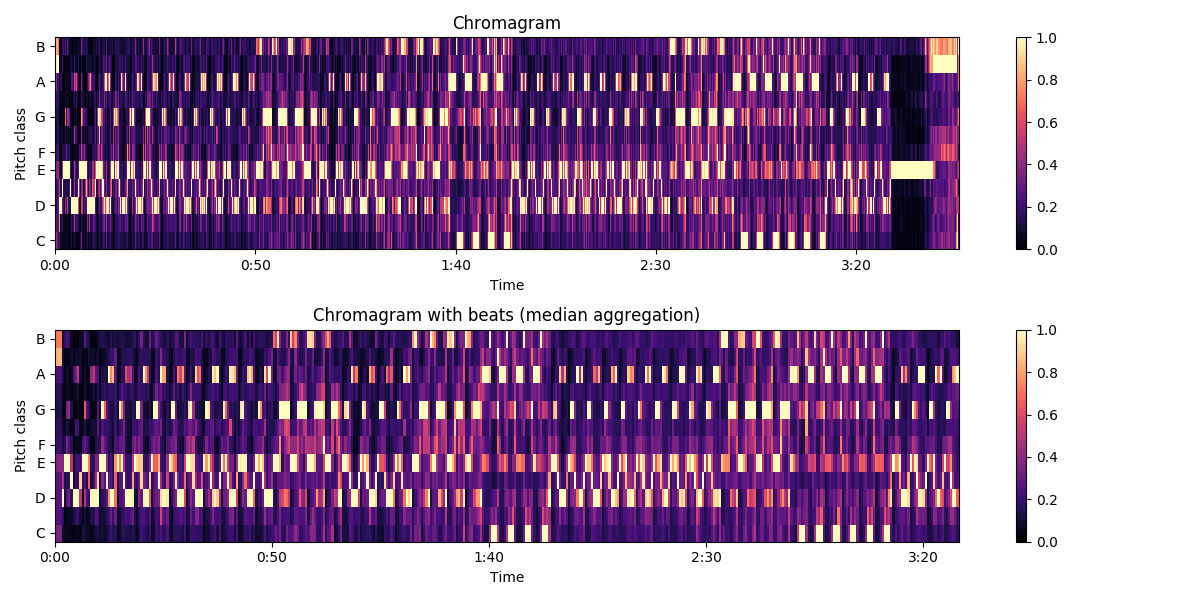
“Median aggregation” of the chromagram shows the summarized beats and reduces the dimensionality.

C\_sync = librosa.util.sync(C, beats, aggregate=np.median)

The chromogram for the sample audio is generated and shown below.



The chromogram generated for Nirvana’s 1992 song “Come As You Are” is shown below.

For illustration, the background guitar riff is in the pitch classes E, E#, D throughout the song and can be seen in the chromagram as well along with other pitch classes.

**Spotlight Control**

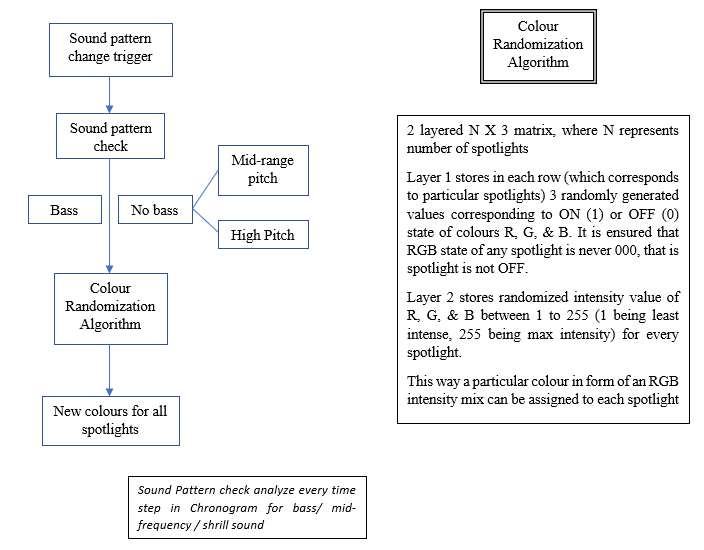
To make the concert environment dynamic, we randomize the colours of each spotlight, their intensity, flickering rates, and fade to liven up the lightings in the area. The outputs from Audio Interpretation section is fed to another code which gives data for intensity, flickering frequency and color.

The algorithms do the following actions –

* Randomize the colours of each spotlight when a beat or sound pattern is changed
* Control the flickering frequency based on beat frequency
* Control the flicker intensity based on beat amplitude for sections of concert with beats in the song
* For beat-less, slow sections, control fade in and fade out of lights

**Randomizing Colours**

To randomize the colours, an algorithm similar to “pixel colouring” algorithm (RGB grey-level frame buffer in CRT display) is used to decide the colour of each spotlight. Algorithm trigger flow chart & Algorithm are explained below. Sound pattern change trigger acts, when sound pattern has drastic change in beat frequency, or pitch of sound.



**Controlling Flickering Frequency**

Beat frequency is obtained from beat tracking code.

The same frequency is assigned to overall ON / OFF state of spotlight, thus controlling flickering of spotlight.

Note that the beat frequencies change every few milliseconds, and thus giving dynamism to the spotlight flicker.

Spotlight colours do not change in this flicker rate control loop

**Controlling Flicker Intensity**

Beat flicker intensity can be understood as amplitude of intensity of lights based on amplitude of beats. For very high beat amplitude, spotlight tends to be very bright, whereas for low amplitude beats, the colour of spotlight is seen dull.

Mathematically, this can be expressed as scaling the individual intensity values of RGB by an amplitude factor A with every time step

**Fade In / Fade Out Control**

For mid-range frequencies, or high frequencies, generally beats do not exist. In such cases, we slowly fade in OR fade out selected spotlights (some spotlights fade in, some fade out, some stay with same intensity – generally the central light)

The rate of fade in is taken to be same as rate of fade out, which is taken as rate of pitch variation in the sound with each time step.

OR

**Results**

* The visuals of controlling stage lights as per its frequency spectrum are appreciable.
* Other inputs such as beats per seconds and length of beats will give a more intuitive visual of lights.
* Doing an FFT for sound packets of 0.1 second extracted from lengthy song is easy on computation load.
* Taking output from frequency spectrum in forms of amplitudes and randomly distributing to different lights gives a more immersive experience. This step ensures that light spectrum do not look like an equalizer pattern which may be monotonic.
* The audio analysis gave the beats per minute (bpm) of the sample audio, the time for each beat and the tempo of the beat. These can be effectively used for timing the stage lights and their movement.
* The audio analysis also showed the various pitch classes and instruments time along with their onset and correlation.

**Conclusion and Future Work**

Building on our understanding of the mathematical basis for product modelling and aim of implementing an interactive stage design for controlling lights on the beats of music, we have designed such a stage in Unreal Engine. Alongside this, audio analysis has been done in Matlab and Python.

Future Work:

* Machine Learning: Since our project aims at removing the light man behind the stage, the human decision part has to be automated too. For example, the color of lights will depend on the mood of the song, or more specifically the genre of the song. Genre identification in music can be effectively implemented using Neural Networks and which lights to be flickered or moved in a specific song can be automated.
* Post-processing effects to make the lights match the music even more accurately. These include strobes, black out, bright white flashes, hue shifts.
* Currently, we are limited to music files already stored in system. We would like to expand the capability to take in user input of audio tracks in real time.
* If possible, implement a live demo of the working of our algorithm by designing an electrical circuit and using LED lights.
* Stratification of colour bands to be used in different kinds of sound patterns.
* Automatic connection of the spotlight control code with audio analysis code.
* Control of the rotation of spotlight & control of solid angle of the spotlight cone.
* Control of lasers & artificial fog.