Unraveling Syntheses of Sight and Sound

Final Project for
CS 440 - Computer Graphics - L1

Syed Ammar Mahdi (03691)
Fatima Nadeem (03768)
Kabir (03925)

Contents

1 Abstract

The interplay between music and images lends itself to interesting approaches in a computer graphics context. A musical piece can have multiple subjective visual representations, but finding a mapping from sound to image that allows the viewer to infer some musical properties of the music is a daunting task. In this paper we discuss a novel approach to mapping a chord progression in a piece of music to different colours, using some of the techniques that were discussed in CS-440 (namely interpolation). The approach outlined here maps all 12 notes of the western chromatic scale to the 12 colours of the RGB colour wheel, then bases. The output is a grid of pixels where each chord is represented as a line of interpolated pixels; this attempts to visually represent the melodic and subjective emotional content of different chords to colours in an image. For the implementation, the myimage class in python was used, which was introduced in the first assignment.

2 Introduction

The primary concern of Computer Graphics is to 'display pretty pictures on a computer'[1]. Similarly, music is a way of arranginig aesthetically pleasing sounds together in a wa

Light and sound are both natural phenomenon that occur within the universe as waves; light being a transverse wave, and sound being a longitudinal wave. It is not the physics of these wave phenomenon that is of primary interest to the computer scientist, however, as much as how they can be modelled on a computer. To understand how light and sound can be modelled in meaningful ways on a computer, it is first important to understand how human beings perceive these natural phenomenon.

A sound wave can Human hearing is similar to light in the sense that sounds and light are both logarithmic. The human eye corresponds to changes in brightness on the visible light spectrum in a similar way to how the ear perceives sounds on the decibel scale; each subsequent increase is loudness or brightness is not perceived as a linear increase, but a logarithmic one. Both light and sound are waves that are picked up by human sensory organs and correspond to a phenomenon in the brain. Just as vision in computer graphics is modelled based on the human eye and visual system (as we saw in the first module of the course), so too can sound be modelled based on the human hearing system and how the brain perceives pitches.

Keeping these similarities between the phenomenon of vision and hearing in mind, one can then consider what qualities of vision and sound must be captured in a model to convert between the two. To do so one needs a mapping matrix, as shown in the image above. For our research we will consider a simplified mapping matrix that will consider timbre and pitch in the sound domain and map it in the image space of brightness and colour.

[1] Saleem, Waqar. "CS 440: Lecture 1." Habib University, August. 2020

The question of converting between sound and colour and vice versa is not a trivial one; it can have important applications for conveying information to those deprived of one sense or the other. For example, we can consider how a blind human who cannot see any images would instead interpret a meaningful sound representation of the same image. Similarity, a deaf individual attending an electronic music festival might not be able to see hear the music being played, but can interpret the visuals being displayed as the music plays to have an aesthetic experience; after all, colour has been shown to psychologically affect the mood of a human just as listening to a piece of music can invoke strong emotions in a person.

3 Literature Review

This section is dedicated to discussing the different research works that were researched as part of this project. Firstly, the main paper that guided our implementation as well as provided the impetus for it is discussed. Following that, the section goes into other helpful and related works.

3.1 Main Work Regardng Image to Sound Coversion

The paper in question here is [1]. The paper basically dives into the problem by providing a mapping between image and sound characteristics. It uses this mapping to launch some concepts such as chromatic bricks and segments that make up a particular image while also specifying the notes for a melodic composition as well. The paper and its methodology served as a direct inspiration for the one developed for this project and focused on aspects it discussed, e.g. the smooth transitioning of colors across pixels for sophisticated melodies.

3.2 Other Relevant Work Regardng Image to Sound Coversion

The further work of the researchers of the above discussed work also gave a direction for where their work was headed. In [2], they stretch the idea further to incorporate a reversal of the main problem and devise an effective way to generate imagery from sound given. This way also utilizes earlier discussed concept of chromaticsim of music. Furthermore, they also researched how the synthesis of sound and imagery could greatly benefit a visual performance and promote a cross cultural appreciation of sound and imagery.
[3] proved as yet another inspiration for our implementation as it gave a direction with regards to how interpolation is used in the problem regarding sound to imagery. It takes a novel approach to interpolation in this regard.

4 Our Method

Our approach is a novel one in colour representations of a musical piece. The basic idea is to take the chord progression in a piece of music, find out the notes in the relevant chord and the colour each note is mapped to, then interpolate between the colours of the chord/notes. This will then display a line of interpolated pixels in the output pixel grid image. The details of this method are elucidated in the sections below.

4.1 Theory

The musical background needed to understand this our approach of mapping musical notes to colours are the concepts of notes, scales and chords.

Notes are simply pure frequencies of sound played at a certain frequency that sound good in relation to one another

4.1.1 The Mapping

In the western musical systems, there are musical notes; these notes are collectively referred to what is known as the chromatic scale. The notes in the chromatic scales are as follows:

A, A#/Bb, B, C, C#/Db, D, D#/Eb, E, F, F#/Gb, G, G#/Ab

This can be much more easily visualized using the keys on a piano as detailed in the figure[2] below:

The X#/Yb notes above (where X, Y represent any of the musical notes) are what are known as 'accidentals', or the black keys on the piano. Here we will use the 'sharp' (X#) representation of an accidental, for simplicity's sake.

The chromatic scale has a modular structure; that means that the note C will be the next note after G#, just an octave above the previous C note.

The RGB colour wheel in its most common representation has 12 colours, as detailed in the figure[3] below:

One can then imagine a simple mapping between the 12 notes in the chromatic scale and the 12 colours on the colour wheel. Using this as our starting point, we can move on to colour representation of chords from the mapping defined above, which will then interpolate the colours in the chord depending on the notes.

[2] source: https://www.piano-keyboard-guide.com/wp-content/uploads/2015/04/piano-chromatic-scale.jpg

[3] source: https://i.pinimg.com/originals/ad/4b/cf/ad4bcfcd6b94b8be1aaa9717c08ff580.png

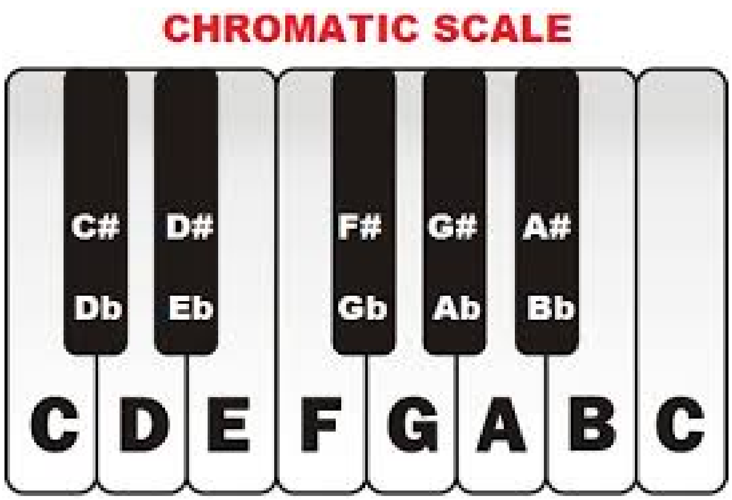


Figure 1: The western chromatic scale.

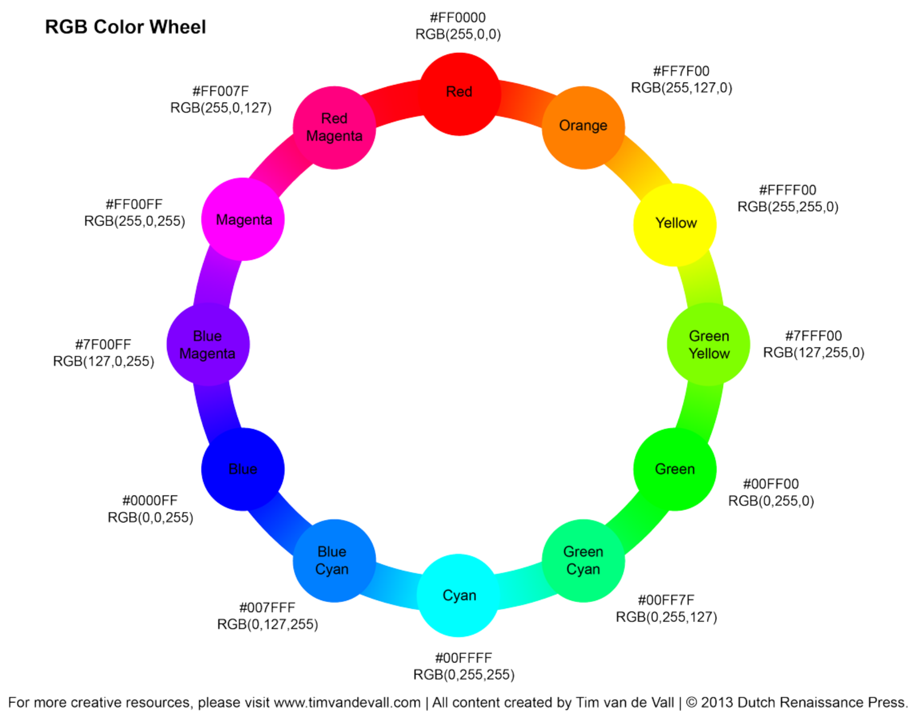


Figure 2: A commonly used representation of the RGB colour wheel with its respective RGB values.

4.2 Resources

5 Code

```python
from myimage import MyImage

'''
An array containing the musical sequence in chords:
Each entry music[m] of the array is a single beat.
It can either be a chord (fifth, major, minor or seventh major) or
    silence.
N refers to null, or beats with silence.

Color wheel for notes on the western chromatic scale:
  Reference: https://i.pinimg.com/originals/ad/4b/cf/
    ad4bcfcd6b94b8be1aaa9717c08ff580.png

  Red: (255, 0, 0), Orange: (255, 127, 0),
  Yellow: (255, 255, 0), Green-Yellow: (127, 255, 0),
  Green: (0, 255, 0), Green-Cyan: (0, 255, 127),
  Cyan: (0, 255, 255), Blue-Cyan: (0, 127, 255),
  Blue: (0, 0, 255), Blue-Magenta: (127, 0, 255),
  Magenta: (255, 0, 255), Red-Magenta: (255, 0, 127)

'''

# The western chromatic scale:
# Sharp representation (#) of accidental notes
# is used rather than flat representation (b).
chroma = [
  'A', 'A#', 'B', 'C', 'C#', 'D',
  'D#', 'E', 'F', 'F#', 'G', 'G#'
]

# Scales for chords
major = '02212221'
minor = '02122122'

# The chromatic scale to color wheel mapping
mapping = {
  'A': (255, 0, 0), 'A#': (255, 127, 0),
  'B': (255, 255, 0), 'C': (127, 255, 0),
  'C#': (0, 255, 0), 'D': (0, 255, 127),
  'D#': (0, 255, 255), 'E': (0, 127, 255),
  'F': (0, 0, 255), 'F#': (127, 0, 255),
  'G': (255, 0, 255), 'G#': (255, 0, 127)
}

def _ChordToNotes(chord: str):
    '''
    Helper function to return notes given a chord.
    Only for chords fifth, major, minor and seventh major.

    Arguments:
    chord as a string

```

```
    Returns:
    A list of notes

    '''

    # Separating first note and chord types
    firstNote = ''
    chordType = ''

    for char in chord:
      if (char in 'ABCDEN#'):
        firstNote += char
      elif (char in 'm57M'):
        chordType += char

    if (len(chordType) == 0):
      chordType += 'M'

    # Finding the position of first note
    # on the chromatic scale
    chromaInd = chroma.index(firstNote)

    # Returning list of notes according to
    # first note and chord type
    ret = [firstNote]

    if (chordType == '5'):
      for i in range(7):
        chromaInd =+ (chromaInd + int(major[i])) % 12
        if (i == 4):
          ret.append(chroma[chromaInd])

    elif (chordType == 'M'):
      for i in range(7):
        chromaInd = (chromaInd + int(major[i])) % 12
        if ((i == 2) or (i == 4)):
          ret.append(chroma[chromaInd])

    elif (chordType == 'm'):
      for i in range(7):
        chromaInd =+ (chromaInd + int(minor[i])) % 12
        if ((i == 3) or (i == 4)):
          ret.append(chroma[chromaInd])

    else:
      for i in range(7):
        chromaInd =+ (chromaInd + int(major[i])) % 12
        if ((i == 2) or (i == 4) or (i == 6)):
          ret.append(chroma[chromaInd])

    return ret

def _InterpolateHorizontal(img, startPos, endPos, startColor,
    endColor):
    '''
    Helper function to interpolate colors
    between two pixels on a horizontal line
```

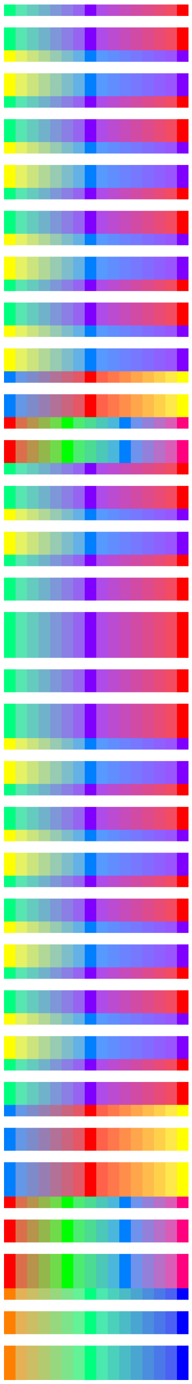
```

    Arguments:
    img as MyImage object to write pixels into.
    startPos as starting pixel position.
    endPos as ending pixel position.
    startColor as color of starting pixel in rgba tuple.
    endColor as color of ending pixel in rgba tuple.

    Returns:
    the MyImage object img after interpolation.

    '''
    diff = abs(endPos[0] - startPos[0])

    # Color component gradients
    r_grad = (endColor[0] - startColor[0]) / diff
    g_grad = (endColor[1] - startColor[1]) / diff
    b_grad = (endColor[2] - startColor[2]) / diff
    # a_grad = (endColor[3] - startColor[3]) / diff

    # Coloring starting pixel
    img.putpixel(startPos, startColor)

    x = startPos[0]
    y = startPos[1]
    r = startColor[0]
    g = startColor[1]
    b = startColor[2]
    # alpha value lowered to differentiate
    # between played notes and interpolated colors
    a = 180

    # Interpolating and coloring
    for pixelPos in range(1, diff):
      x += 1
      r += r_grad
      g += g_grad
      b += b_grad
      # a += a_grad

      img.putpixel((x, y), (round(r), round(g), round(b), round(a)))

    # Coloring ending pixel
    img.putpixel(endPos, endColor)

    return img


def CreateWall(music: []):
    '''
    Function to convert a melody to an image

    Arguments:
    music as a list of chords.

    Returns:
```

```
    The MyImage object img that contains the wall.

    '''

    # Setting y dimension
    yDim = len(music)

    # Creating image
    img = MyImage((16, yDim), 0, 20, 'RGBA')

    # For each chord position or y value
    for chordPos in range(yDim):
      # Storing the chord
      chord = music[chordPos]

      # If it is not silent
      if (chord != 'N'):
        # Extracting notes from the chord
        notes = _ChordToNotes(chord)

        # Values needed to interpolate between notes
        notesLastInd = len(notes) - 1
        jump = 15 // notesLastInd

        # For each pairs of notes except the last two
        for i in range(notesLastInd - 1):
          # Storing start and end colors
          startColor = mapping[notes[i]] + (255,)
          endColor = mapping[notes[i + 1]] + (255,)

          # Interpolating colors in the middle accordingly
          img = _InterpolateHorizontal(img, ((jump * i), chordPos), ((jump * (i + 1)), chordPos), startColor, endColor)

        # Interpolating colors between the last two notes
        startColor = endColor
        endColor = mapping[notes[notesLastInd]] + (255,)
        img = _InterpolateHorizontal(img, ((jump * (notesLastInd - 1)), chordPos), (15, chordPos), startColor, endColor)

      else:
        # Else fill a white line of 16 pixels
        color = (255,) * 4
        img = _InterpolateHorizontal(img, (0, chordPos), (15, chordPos), color, color)

    # Showing and returning image
    img.show()
    return img

def _test():

    music = [
      'D', 'N', 'D', 'D', 'Bm', 'N', 'Bm', 'Bm',
      'D', 'N', 'D', 'D', 'Bm', 'N', 'Bm', 'Bm',
      'D', 'N', 'D', 'D', 'Bm', 'N', 'Bm', 'Bm',
      'D', 'N', 'D', 'D', 'Bm', 'N', 'Bm', 'Bm',
```

```

        'Em', 'N', 'Em', 'Em', 'A7', 'N', 'A7', 'A7',
        'D', 'N', 'D', 'D', 'Bm', 'N', 'Bm', 'Bm',
        'D', 'N', 'D', 'D', 'N', 'D', 'D', 'D',
        'D', 'N', 'D', 'D', 'N', 'D', 'D', 'D',

        'Bm', 'N', 'Bm', 'Bm', 'D', 'N', 'D', 'D',
        'Bm', 'N', 'Bm', 'Bm', 'D', 'N', 'D', 'D',
        'Bm', 'N', 'Bm', 'Bm', 'D', 'N', 'D', 'D',
        'Bm', 'N', 'Bm', 'Bm', 'D', 'N', 'D', 'D',

        'Em', 'N', 'Em', 'Em', 'N', 'Em', 'Em', 'Em',
        'A7', 'N', 'A7', 'A7', 'N', 'A7', 'A7', 'A7',
        'A#', 'N', 'A#', 'A#', 'N', 'A#', 'A#', 'A#'
    ]

    CreateWall(music)

if __name__ == '__main__':
  _test()
```

Listing 1: Main Code for Sound To Image Conversion

6 Output



7 Conclusion

References

[1] Dimitrios Margounakis and Dionysios Politis. "Converting images to music using their colour properties". In: Georgia Institute of Technology. 2006.

[2] Dionysios Politis, Dimitrios Margounakis, and Michail Karatsoris. "Image to sound transforms and sound to image visualizations based on the chromaticism of music". In: *7th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Databases*. 2008, pp. 309–317.

[3] Peter Ciuha, Bojan Klemenc, and Franc Solina. "Visualization of concurrent tones in music with colours". In: *Proceedings of the 18th ACM international conference on Multimedia*. 2010, pp. 1677–1680.

8 Appendix A: Penalize

The following is a parody of the song "Still Alive"[4] from the video game portal, which will be used as an example song to output chord colours using our program.

```
[VERSE]

D Bm D Bm
D Bm D Bm
Em A7
D Bm D Bm
D Bm D Bm
D Bm D Bm
Em A7
A#7

[CHORUS]

F C A# F
F C A# F
Gm C F C Dm
A# A7

D Bm D Bm
D Bm D Bm

[Intro/Verse 1]
```

[4] source: https://www.youtube.com/watch?v=Y6ljFaKRTrI

```
             D     Bm D
This was not tender
    Bm           D
I'm making a note here
Bm          D Bm
'What a mess'
     Em              A7        D      Bm D
Your code comments are against my satisfaction

Bm         D     Bm D
'Horrible render'
Bm             D       Bm      D Bm
we do what we must because we can
Em                   A7                    A#
It's for the good of all of us Except the ones who have dropped


[Chorus]

            F        C           A       F
but there's no sense crying over every mistake
         F       C                A#         F
I commend you for trying but that's not what it takes
        Gm          C            F      C  Dm
I'm on top of this course and I'll show you no remorse
        A#          A7
When I'm gonna penalize


D Bm D Bm
D Bm D Bm


[Verse 2]

             D    Bm  D
I'm not even angry
Bm          D      Bm      D   Bm
I'm being so sincere right now
Em              A7                     D   Bm D
I wish I could erase your code from memory
    Bm           D   Bm D
And tear it to pieces
    Bm               D   Bm      D  Bm
And throw every piece into a fire
Em                 A7                    A#
```

```
As it burns I hope you learn how to write decent code


[Chorus]

          F         C           A#         F
Now these pixels on the screen make a beautiful line
          F         C           A#          F
It's a shame your program did not render on time
        A#          C                F       C       Dm
You may think it's no fun, but think of all the things you'll learn
        A#           A7
When I am gonna penalize

D Bm D Bm
D Bm D


[Verse 3]

               D  Bm  D
Go ahead and contest
  Bm            D      Bm      D   Bm
I think the instructor will take my side
Em                A7                  D   Bm D
Maybe you'll find someone else to help you
Bm           D   Bm D
Maybe Anisa?
Bm              D     Bm            D  Bm
That was a joke - haha - FAT CHANCE!
Em           A7                          Bb
Anyway, your tears are great - they're so delicious and moust


[Chorus]

        F         C                    A#          F
Look at me still talking while there are more reviews to do

      F        C                  A#           F
When I read your gibberish it makes me GlaD I'm not you
       A#         C                 F       C       Dm
There's no reason to be mad; Your code's honestly not half-bad
       A#          A7            D
But I am still gonna penalize
```

```
[Outro]

Bm      D           Bm           D
And believe me, I will penalize!
Bm        D             Bm            D
And for no reason I will penalize
Bm        D             Bm           D
I feel FANTASTIC when I penalize!
Bm              D           Bm            D
And for no reason I will penalize!
Bm              D            Bm           D
And if your code runs I’ll still penalize!
Bm      D
Penalize
Bm      D
Penalize
```