УДК 004.94
DOI: 10.18384/2949-5067-2023-4-64-80

ПРЕОБРАЗОВАНИЕ ПИКСЕЛЬНОЙ СТРУКТУРЫ В ЗВУКОВЫЕ ОТОБРАЖЕНИЯ. Часть 1.

Ключников С. А., Калашников Е. В.

Государственный университет просвещения
141014, Московская область, г. Мытищи, ул. Веры Волошиной, д. 24,
Российская Федерация

Аннотация
Цель. Выявить связь между визуальным и звуковым восприятием.
Процедура и методы. При помощи объектно-ориентированного программирования (ООП) языка Python ищется способ преобразования визуального (пиксельного) отображения в звуковое отображение. Применяется ряд современных и функциональных библиотек. Используется современные способы «упаковки» всех программных компонентов в один файл для удобной развёртки программы на электронно-вычислительном устройстве (ЭВМ) с любой современной операционной системой (ОС).
Результаты. Создан программный продукт на основе современного ООП языка программирования и функциональных библиотек, позволяющий представить пиксельную структуру визуального изображения в звуковое отображение.
Теоретическая и/или практическая значимость исследования заключается в раскрытии современного способа «упаковки» всех программных компонентов в один файл для удобной развёртки программы на электронно-вычислительном устройстве (ЭВМ) с любой современной операционной системой. Это позволяет преобразовать цветовое визуальное изображение со множеством оттенков в звуковое отображение.
Ключевые слова: преобразование, программирование, Python, библиотеки, музыка, октавы, фортепиано, цветовая модель, пиксели

CONVERTING A PIXEL STRUCTURE INTO SOUND IMAGINATIONS. Part 1

S. Klyuchnikov, E. Kalashnikov

Federal State University of Education
ulitsa Very Voloshinoi 24, Mytishchi 141014, Moscow Region, Russian Federation

Abstract
Aim. To identify the connection between visual and sound perception.

Methodology. Using Object-Oriented Programming (OOP) of the Python language, we are looking for a way to convert a visual (pixel) display into an audio display. A number of modern and functional libraries are used. Modern methods of "packing" all software components into a single file are used for convenient program deployment on an electronic computing device (computer) with any modern operating system (OS).
Results. A software product based on the modern object-oriented programming language and functional libraries was created, which allows to present the pixel structure of a visual image in a sound display format.
Research implications. The significance lies in the disclosure of a modern way of "packing" all software components into a single file for convenient program deployment on an electronic computing device (computer) with any modern operating system (OS).
Keywords: Conversion, Programming, Python, libraries, music, octaves, piano, color model, pixels

Введение

Цвет и звук всегда привлекали внимание человека. Умение сочетать цвета приводит к созданию изображений. Сочетание различных звуков позволяет создавать музыкальные мелодии. И то и другое вызывает определённые эмоции человека. Более того, созерцание изображений часто вызывает звуковые (музыкальные) ассоциации. Также музыкальные (звуковые) произведения вызывают визуальные ассоциации. Такая связь не очевидна. С другой стороны, одно из направлений теоретической физики предусматривает исследования физических моделей когнитивных процессов, в которых задействованы восприятие, память, размышления, ассоциации. Для того чтобы моделировать такие сложные и плохо определённые (в математическом и физическом смысле) ситуации необходимо рассмотреть ситуации, в которых была бы возможность выразить звуковые отображения через цветовые и наоборот. Данная работа как раз и направлена на исследование такой возможной связи «цветовое изображение – звуковое изображение».

Первые попытки сопровождать изображения музыкой (тапёры) относятся ко времени создания первых немых фильмов. А первое осмысленное сочетание музыкального произведения с цветовым и световым сопровождением создал Скрябин А. Н. («Прометей (Поэма огня)» впервые со световой партией был исполнен в 1915 г. в Нью-Йорке[1]). Позже, в 20-30-х гг. прошлого века над созданием цветомузыкальных произведений работал Лев Термен. В настоящее время практически нет препятствий для создания программ, генерирующих музыкальные произведения, сочетающие музыкальное сопровождение с изображением и его движением[2]. Однако в связи визуального

[1] См.: Александр Николаевич Скрябин. «Прометей» [Электронный ресурс] // Музыкальные сезоны: [сайт]. URL: https://musicseasons.org/aleksandr-nikolaevich-skryabin-prometej/ (дата обращения: 04.06.2023).

[2] См.: Генерация музыки из изображений с помощью Python (перевод Д. Брайта) [Электронный ресурс] // Хабр : [сайт]. URL: https://habr.com/ru/companies/ruvds/articles/708890/ (дата обращения: 04.06.2023); Murcia V. Making Music From Images with Python [Электронный ресурс] // Medium : [сайт]. URL: https://medium.com/m/global-identity-2?redirectUrl=https%3A%2F%2Fbetterprogramming.pub%2Fmaking-music-from-images-with-python-81db627fd549 (дата обращения: 04.06.2023).

изображения с возможным звуковым отображением нас интересует формальная сторона взаимосвязи звукового и цветового ряда.

Действительно, существует семь основных нот и семь основных цветов. И вопрос состоит в том, чтобы выяснить, как цвет изображения отображается в звук и, наоборот, как звук может отображаться в цветовое изображение. Минимальным элементом изображения является пиксель. В связи с этим возникает задача: по любому изображению, основываясь на его пиксельной сетке, создать звуковое отображение.

1. Формирование отображения цветового изображения в звуковое поле

Для того чтобы построить схему преобразования цветовой гаммы изображения в форму, удобную для анализа изображения, необходимо выделить особенности формирования цветов с учётом восприятия цветов зрением человека [1; 2][3]. Человек, в свою очередь, должен нормально воспринимать цвета компьютерного изображения. По большей части люди трихроматы, т. е. способны воспринимать три основных цвета, заложенных в матрице всех мониторов, а именно: синий, красный и зелёный. Сетчатка человека благодаря рецепторам воспринимает длины световых волн, которые маркируются как S, M, L: S – воспринимает синий, L – красный и M – зелёный; или RGB – маркировка первых трёх букв цветов, воспринимаемым человеческой сетчаткой (см. рис. 1). Также существует RGBA, где A – это степень прозрачности цвета.

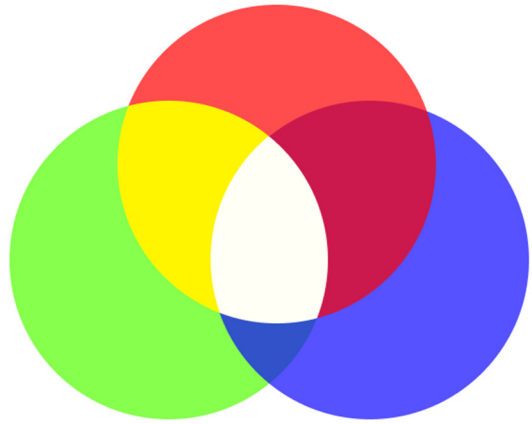

Рис. 1 / Fig. 1. Модель RGB-схемы / RGB Scheme Model

Источник: составлено авторами.

[3] Также см.: Теория цвета как основа для дизайна и иллюстрации (перевод Д. Брайта) [Электронный ресурс] // Хабр: [сайт]. URL: https://habr.com/ru/companies/ruvds/articles/553582/ (дата обращения: 04.06.2023). Далее: Теория цвета как основа для дизайна и иллюстрации // Хабр ; Color Theory: The Primer for Designers and Illustrators [Электронный ресурс] // DESIGNXPLORER: [сайт]. URL: https://designxplorer.co/color-theory-for-designers-and-illustrators/ (дата обращения: 04.06.2023). Далее: Color Theory: The Primer for Designers and Illustrators // DESIGNXPLORER.

Сложение данных трёх SML-цветов даёт различные оттенки. Также существует и аналогичная данной модели субтрактивная модель, которая маркируется как CMYK и формируется от обратного модели RGB. CMYK-модель создаётся вычитанием цвета, это её фундаментальная особенность в отличие от RGB-модели [1].

Существуют и обратные модели RGB, это BGR-модели. Данная модель является устаревшей и не удобной для восприятия текстовой составляющей, особенно на операционных системах семейства Windows. При воспроизведении изображения происходит компоновка субпикселей обратной модели из-за чего восприятие небольших деталей (текста) становится трудно читаемым. BGR-формат, благодаря своим недостаткам, очень редко встречается в современной электронной технике.

1.1. Цветовое изображение на пиксельной сетке

Цветовая модель HSV (Hue Saturation Value) также известна как «цветной круг» в любом редакторе изображений, начиная от Paint, заканчивая профессиональными специализированными программами для обработки графических изображений. Данная цветовая модель была представлена в 1978 г. Элви Реем Смитом. Модель Hue Saturation Value является нелинейным преобразованием модели RGB[4].

Модель *HSV* можно представить в виде трёхмерного объекта в цилиндрической системе координат (см. рис. 2). Полярный угол радиус-вектор *S* в данной системе координат носит значение *H*. Чтобы выбрать нужный оттенок цвета, необходимо перемещаться вдоль окружности цилиндра, яркость выбирается при помощи высоты, а для того чтобы выбрать насыщенность цвета, следует перемещаться вдоль радиуса [1][5]. Данная цветовая модель имеет как ряд преимуществ, так и недостатков, одним из них служит невозможность различить цвета, когда переменная V – яркость, приближается к оттенкам 0 (чёрный), поэтому, чтобы сгладить такие недостатки, используется каноническая модель представления HSV. В прикладном программном обеспечении модель Hue Saturation Value используется в двумерном виде, представленном в ряде современных прикладных программ, предназначенных для обработки цветовых изображений[6].

[4] См.: HSV (цветовая модель) [Электронный ресурс] // Академик : [сайт]. URL: https://dic.academic.ru/dic.nsf/ruwiki/605367 (дата обращения: 04.06.2023). Далее: HSV (цветовая модель) // Академик.

[5] Также см.: Теория цвета как основа для дизайна и иллюстрации // Хабр; Color Theory: The Primer for Designers and Illustrators // DESIGNXPLORERHSV; HSV (цветовая модель) // Академик.

[6] См.: HSV (цветовая модель) // Академик.

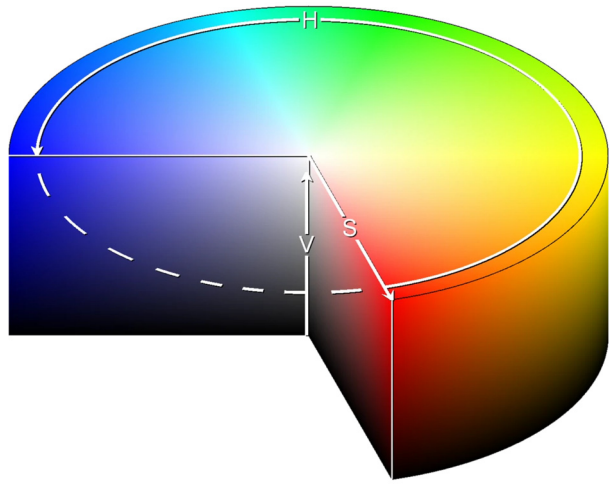

Рис. 2 / Fig. 2. Трёхмерный объект HSV в цилиндрической системе координат / Three-dimensional HSV object in a cylindrical coordinate system

Источник: HSL and HSV [Электронный ресурс] // Psychology Wiki. Fandom: [сайт]. URL: https://psychology.fandom.com/wiki/HSL_and_HSV (дата обращения: 04.06.2023).

При определении выбора цветового пространства для построения программного компонента выбрали модель цветового пространства HSV (рис. 2). Главная цель программы – определение графических примитивов на изображении. Для этого необходимо понимать, ***каким образом фильтровать цвет от всего массива изображения***. Возьмём основное известное цветовое пространство RGB (красный, зелёный, синий), рис. 1, которое в свою очередь состоит из набора трёх различных матриц, одноимённых названию трёх заглавных английских букв RGB. Для каждого пикселя в трёх матрицах существует своё собственное значение, варьирующееся от 0 до 255, данное значение показывает насыщенность или интенсивность свечения цвета. Кроме того, для обработки одного цвета придётся обрабатывать 3 различные матрицы, каждая из которых состоит из набора значений от 0 до 255. Из чего можно сделать вывод, что цветовое пространство RGB нецелесообразно использовать для обработки преобразований изображения.

Такое цветовое пространство, как HSV, позволяет удобно и интуитивно понятно работать и обрабатывать цветовые матрицы, используя только один массив данных H (см. рис. 3), который в свою очередь отвечает за все тона изображения.

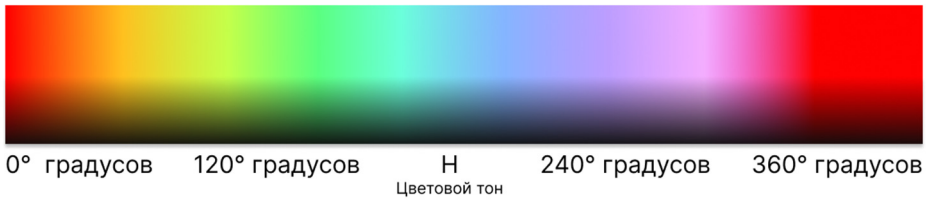

Рис. 3 / Fig. 3. Цветовое пространство HSV / HSV color space

Источник: составлено авторами.

Из рис. 3 можно сделать вывод, что основные цвета HSV носят координаты радиуса круга, а именно: оранжевый от 0° до 44°, жёлтый от 44° до 76°, зелёный от 76° до 150°, синий от 150° до 260°, фиолетовый от 260° до 320°, красный от 320° до 360°.

1.2. Формализация звукового поля

В процессе преобразования изображения в музыку важно выбрать подходящий музыкальный инструмент. По-видимому, наиболее подходящим для реализации поставленной задачи является фортепиано. Фортепиано имеет клавиши, ноты и частоты. Эталонная нота Ля первой октавы имеет базовую частоту в 440 Гц, благодаря которой происходит вся настройка звуковых инструментов, так как ориентироваться приходится именно на данную ноту. Для того чтобы отобразить полноту гармонии звуков с октавами, разделёнными на двенадцать полутонов, где в каждой следующей октаве частота основного тона выше эталонной в два раза, необходимо использовать всю наполненность частот и тонов такого музыкального инструмента, как фортепиано.

Фортепьяно имеет набор из 7 +1/4 октавы, или другими словами 88 клавиш от A0 до C8 (рис. 4). 7 +1/4 октавы означает определённый диапазон клавиш, в который входит 7 полных октав и дополнительные 1/4 октавы. Октава состоит из уникальных 12 нот, и в 7 октавах как раз есть 84 клавиши, а именно 7*12 = 84; если к 84 основным клавишам добавить 1/4 октавы, 12*1/4 =3, общее число клавиш получается 87, но стандартный музыкальный инструмент имеет 88, это связанно с A0 – субконтроктавой, которая расширяет диапазон на одну ноту вниз, нота A0 является самой низкой нотой. Таким образом, число клавиш ровняется 88 или 7 +1/4 октавы.

Все клавиши, включая чёрные и белые, представлены на рис. 4, их количество составляет 88. В программе это выражено при помощи массива «keys», в котором сохраняются необходимые значения стандартных 88 клавиш с нотациями от A0 до C8.

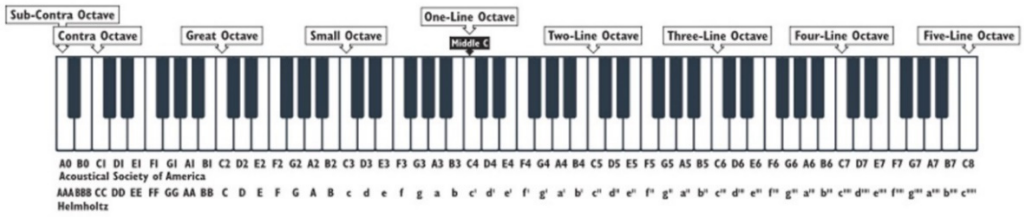

Рис. 4 / Fig. 4. Октавы на фортепиано / Octaves on the piano

Источник: [3].

1.3. Основной принцип работы функции программы на языке объектно-ориентированного программирования Python

Данная функция является фундаментальной в преобразовании пиксельной структуры кадра (изображения) в звуковое сопровождение. Работу функции программы удобно разбить на последовательность действий.

1. Задачей для написания функции является её определение при помощи def.

2. Следующим шагом необходимо создать список, в который будут входить наименование нот в октаве, таким образом:

Octave = [‘C’, ‘c’, ‘D’, ’d’, ‘E’, ‘F’, ‘f’, ‘G’, 'g', ‘A’, ‘a’, ‘B’]

3. Рассматривая вышеописанное, необходимо определить базовую частоту эталонной ноты Ля (440 Гц), для этого присвоим одноимённой переменной значение 440.

4. Определим массив при помощи библиотеки NymPy, поскольку для преобразования гармоничной музыкальной композиции необходимо перебрать и записать все возможные варианты комбинации букв из массива октав 0 до 8.

5. После создания комбинаций необходимо определить индекс первого и последнего элемента в массиве, а именно первой клавиши A0 и последней – C8.

6. Необходимо ограничить массив набором только из необходимых элементов, а именно от A0 до C8.

7. На данном этапе следует создать словарь, в котором будет происходить объединение самого словаря с ключами и значениями, которые сопоставляются с частотами нот, используя формулы 2**((n+1-49)/120) *440 [2].

1.4. Построение «словаря» соответствия нот частотам

Рассмотрим данный элемент программы подробней:

note_freqs = dict(zip(keys, [2**((n+1-49)/12)*440 for n in range(len(keys))])

Для последующего понимания этого элемента программы важно привести комментарии всех его составляющих:

(1) *range(len(keys))* формирует последовательность чисел от 0 до длины массива *keys*. Последовательность будет использована для итерационного перебора элементов массива *keys*.

(2) *((n+1-49)/12)* вычисляет значение внутри показательной функции формулы, используемой для определения частоты ноты.

(3) *(n+1-49)* представляет собой позицию текущей ноты в массиве клавиш со смещением на 1 для соответствия индексации массива.

Деление этого значения на 12 представляет собой количество полутонов, отстоящих от опорной (идеальной) ноты A4 (49 используется потому, что представляет собой позицию A4 в массиве клавиш).

(4) *2**((n+1-49)/12)*440* вычисляет частоту ноты по этому соотношению. При этом 2 возводится в степень ((n+1-49)/12) и умножается на значение 440. Это соотношение отражает связь между частотой ноты и её положением относительно опорной (идеальной) ноты A4.

(5) *[2**((n+1-49)/12)*440 for n in range(len(keys))]* – это списочное вычисление, которое выполняет итерацию по диапазону чисел, сгенерированных на шаге (1). Для каждого числа *n* вычисляется соответствующая частота по формуле из шага (3), в результате чего получается список частот.

zip(keys, [2**((n+1-49)/12)*440 for n in range(len(keys))]) объединяет массив keys со списком частот в пары (ключ, частота).

В результате создаётся последовательность кортежей, в которой каждый кортеж представляет собой клавишу фортепиано и соответствующую ей частоту.

dict(...) создаёт словарь из последовательности кортежей, сформированной на шаге 5.

Ключами словаря являются клавиши фортепиано из массива keys, а значениями – соответствующие им частоты.

Полученный словарь присваивается переменной note_freqs, в которой хранится соответствие между клавишами фортепиано и их частотами.

Таким образом, в данной строке кода с помощью комбинации вычислений, понимания списков и создания словаря формируется словарь (note_freqs), в котором клавиши фортепиано сопоставляются с соответствующими им частотами в зависимости от их положения в массиве клавиш.

Завершающими строками функции служат:

Добавление «пустого ключа» в словарь note_freqs с нулевой частотой. Данная процедура даёт возможность остановить воспроизведение звуковой композиции.

И заключительным этапом является возврат функции:

return note_freqs .

1.4.1. Пример нахождения частоты клавиши отсчёта A0

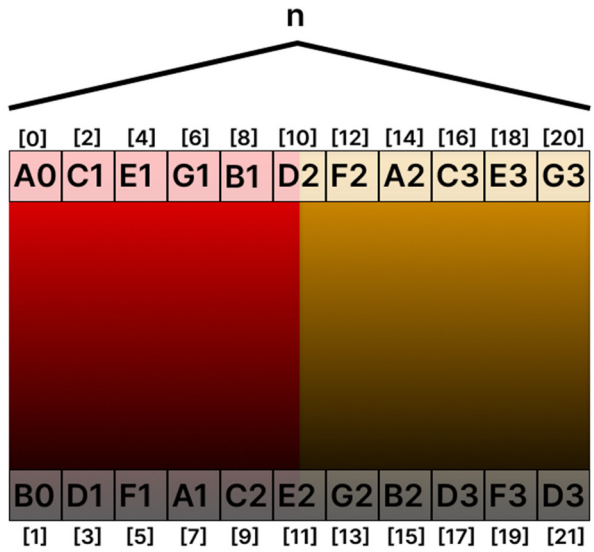

Рис. 5 / Fig. 5. Модель получения индекса частоты, исходящая из логики работы программы / Model for obtaining the frequency index based on the logic of the program

Источник: составлено авторами.

При помощи представленной выше формулы 2**((n+1-49)/12) *440 разберём математическое получение по индексу элемента частоты (Гц) звуковой клавиши фортепиано, рис. 5:

$2^{\frac{n+1-49}{12}} * 440$, возьмём за n первый элемент индекса [0] и вычислим частоту (Гц), а далее и звуковую клавишу фортепиано, $2^{\frac{0+1-49}{12}} * 440 = 2^{\frac{-48}{12}} * 440 = 2^{-4} * 440 = \frac{1}{2^4} * 440 = \frac{1}{16} * \frac{440}{1} = \frac{440}{16} = 27{,}5$. После проделанного решения можно получить 27,5 Гц, что эквивалентно клавише A0, и звуку ЛЯ. Первый звук для первого пикселя будет занесён в словарь, а затем проигран в звуковом сопровождении.

1.5. Частоты для нот фортепиано[7]

Номер клавиши; Нота; Английская нотация; Координатные частоты; Частота (Гц).

88	до7	C8	+43	4186,0096	
87	си6	B7	+42	3951,0656	
86	ля♯6 (си♭6)	A♯7/B♭7	+41	3729,3184	
85	ля6	A7	+40	3520,0000	
84	соль♯6 (ля♭6)	G♯7/A♭7	+39	3322,4320	
83	соль6	G7	+38	3135,9680	
82	фа♯6 (соль♭6)	F♯7/G♭7	+37	2959,9552	
81	фа6	F7	+36	2793,8304	
80	ми6	E7	+35	2637,0240	
79	ре♯6 (ми♭6)	D♯7/E♭7	+34	2489,0176	
78	ре6	D7	+33	2349,3184	
77	до♯6 (ре♭6)	C♯7/D♭7	+32	2217,4656	
76	до6	C7	+31	2093,0048	
75	си5	B6	+30	1975,5328	
74	ля♯5 (си♭5)	A♯6/B♭6	+29	1864,6592	
73	ля5	A6	+28	1760,0000	
72	соль♯5 (ля♭5)	G♯6/A♭6	+27	1661,2160	
71	соль5	G6	+26	1567,9840	
70	фа♯5 (соль♭5)	F♯6/G♭6	+25	1479,9776	
69	фа5	F6	+24	1396,9152	
68	ми5	E6	+23	1318,5120	
67	ре♯5 (ми♭5)	D♯6/E♭6	+22	1244,5088	
66	ре5	D6	+21	1174,6592	
65	до♯5 (ре♭5)	C♯6/D♭6	+20	1108,7328	
64	до5	C6	+19	1046,5024	
63	си4	B5	+18	987,7664	
62	ля♯4 (си♭4)	A♯5/B♭5	+17	932,3296	
61	ля4	A5	+16	880,0000	
60	соль♯4 (ля♭4)	G♯5/A♭5	+15	830,6080	
59	соль4	G5	+14	783,9920	
58	фа♯4 (соль♭4)	F♯5/G♭5	+13	739,9888	
57	фа4	F5	+12	698,4576	
56	ми4	E5	+11	659,2560	
55	ре♯4 (ми♭4)	D♯5/E♭5	+10	622,2544	
54	ре4	D5	+9	587,3296	
53	до♯4 (ре♭4)	C♯5/D♭5	+8	554,3664	
52	до4	C5	+7	523,2512	

[7] Частоты настройки фортепиано [Электронный ресурс] // Циклопедия: [сайт]. URL: https://cyclowiki.org/wiki/Частоты_настройки_фортепиано (дата обращения: 04.06.2023).

51	си3	B4	+6	493,8832	
50	ля♯3 (си♭3)	A♯4/B♭4	+5	466,1648	
49	ля3	A4	+4	440,0000 (является эталонной частотой ноты ля первой октавы)	
48	соль♯3 (ля♭3)	G♯4/A♭4	+3	415,3040	
47	соль3	G4	+2	391,9960	
46	фа♯3 (соль♭3)	F♯4/G♭4	+1	369,9944	
45	фа3	F4	+0	349,2288	
44	ми3	E4	-0	329,6280	
43	ре♯3 (ми♭3)	D♯4/E♭4	-1	311,1272	
42	ре3	D4	-2	293,6648	
41	до♯3 (ре♭3)	C♯4/D♭4	-3	277,1832	
40	до3	C4	-4	261,6256	
39	си2	B3	-5	246,9416	
38	ля♯2 (си♭2)	A♯3/B♭3	-6	233,0824	
37	ля2	A3	-7	220,0000	
36	соль♯2 (ля♭2)	G♯3/A♭3	-8	207,6520	
35	соль2	G3	-9	195,9980	
34	фа♯2 (соль♭2)	F♯3/G♭3	-10	184,9972	
33	фа2	F3	-11	174,6144	
32	ми2	E3	-12	164,8140	
31	ре♯2 (ми♭2)	D♯3/E♭3	-13	155,5636	
30	ре2	D3	-14	146,8324	
29	до♯2 (ре♭2)	C♯3/D♭3	-15	138,5916	
28	до2	C3	-16	130,8128	
27	си1	B2	-17	123,4708	
26	ля♯1 (си♭1)	A♯2/B♭2	-18	116,5412	
25	ля1	A2	-19	110,0000	
24	соль♯1 (ля♭1)	G♯2/A♭2	-20	103,8260	
23	соль1	G2	-21	97,9990	
22	фа♯1 (соль♭ 1)	F♯2/G♭2	-22	92,4986	
21	фа1	F2	-23	87,3072	
20	ми1	E2	-24	82,4070	
19	ре♯1 (ми♭1)	D♯2/E♭2	-25	77,7818	
18	ре1	D2	-26	73,4162	
17	до♯1 (ре♭1)	C♯2/D♭2	-27	69,2958	
16	до1	C2	-28	65,4064	
15	си0	B1	-29	61,7354	
14	ля♯0 (си♭0)	A♯1/B♭1	-30	58,2706	
13	ля0	A1	-31	55,0000	
12	соль♯0 (ля♭0)	G♯1/A♭1	-32	51,9130	
11	соль0	G1	-33	48,9995	
10	фа♯0 (соль♭0)	F♯1/G♭1	-34	46,2493	

9	фа0	F1	-35	43,6536		
8	ми0	E1	-36	41,2035		
7	ре♯0 (ми♭0)		D♯1/E♭1		-37	38,8909
6	ре0	D1	-38	36,7081		
5	до♯0 (ре♭0)		C♯1/D♭1		-39	34,6479
4	до0	C1	-40	32,7032		
3	си−1	B0	-41	30,8677		
2	ля♯−1 (си♭−1)		A♯0/B♭0		-42	29,1353
1	ля−1	A0	-43	27,5000		

2. Описание функциональной составляющей программного компонента

Для создания и обработки изображений рассматривались различные цветовые модели, описанные выше, также учитывались их плюсы и минусы при разработке программных компонентов. Выбрали цветовую модель HSV; поскольку цветовое пространство априори разделено на цветовые зоны, то это позволяет упростить сопоставление элементов с частотой и делает данный процесс более интуитивным.

Отметим, что для создания данной программы необходим ряд библиотек, без которых оптимальная и лояльная к пользователю структура будет невозможна.

Список библиотек:

librosa=0.9.2 (версия актуальной программы на момент установки)[8];

numpy=1.21.6 (версия актуальной программы на момент установки)[9];

opencv-python-headless (версия актуальной программы на момент установки)[10;]

pandas=1.3.5 (версия актуальной программы на момент установки)[11];

pedalboard=0.5.9 (версия актуальной программы на момент установки)[12];

Pillow=9.2.0 (версия актуальной программы на момент установки)[13];

scipy=1.7.3 (версия актуальной программы на момент установки)[14];

streamlit=1.13.0 (версия актуальной программы на момент установки)[15].

В модели HSV тон представляет цвет, а именно: красный, зелёный, синий, фиолетовый, жёлтый, оранжевый.

[8]См.: Librosa – librosa 0.10.1 documentation [Электронный ресурс]. URL: https://librosa.org/doc/latest/index.html (дата обращения: 04.06.2023).

[9] См.: NumPy [Электронный ресурс]. URL: https://numpy.org/ (дата обращения: 04.06.2023).

[10] См.: Opencv-python-headless. 4.9.0.80 [Электронный ресурс]. URL: https://pypi.org/project/opencv-python-headless/ (дата обращения: 04.06.2023).

[11] См.: Pandas=1.3.5 documentation [Электронный ресурс]. URL: https://pandas.pydata.org/pandas-docs/version/1.3/ (дата обращения: 04.06.2023).

[12] См.: Pedalboard 0.8.7 [Электронный ресурс]. URL: https://pypi.org/project/pedalboard/ (дата обращения: 04.06.2023).

[13] См.: Pillow (PIL Fork) 10.2.0 documentation [Электронный ресурс]. URL: https://pillow.readthedocs.io/en/stable/ (дата обращения: 04.06.2023).

[14] См: SciPy=1.7.3 Release Notes [Электронный ресурс]. URL: https://docs.scipy.org/doc/scipy/release/1.7.3-notes.html (дата обращения: 04.06.2023).

[15] См.: Streamlit=1.13.0 [Электронный ресурс]. URL: https://pypi.org/project/streamlit/ (дата обращения: 04.06.2023).

Насыщенность в свою очередь отвечает за цветовой диапазон, который ответственен за яркость объекта.

Яркость отвечает за визуальную составляющую и представление объекта, влияющие на уровень свечения. Уровень смешения светового изображения с символом (-,-%, 0%) соответсвует чёрному.

Рассмотрим значение тона 6 цветов:

Красный от 320 до 360.
Фиолетовый от 260 до 320.
Синий от 150 до 260.
Зелёный от 76 до 150.
Жёлтый от 44 до 76.
Оранжевый от 0 до 44.

На основе цветового пространства рассмотрим код для его генерации:

```
# для считывания значения тона пикселя необходима функция
hsv = cv2.cvtColor(ori_img, cv2.COLOR_BGR2HSV)
# Алгоритм построение изображения
fig, axs = plt.subplots(1, 3, figsize = (15,15))
name = ['BGR','RGB','HSV']
img  = [img, hsv, ori_img]
i = 0
for el in img:
   axs[i].title.set_text(names[i])
   axs[i].imshow(elem)
   axs[i].grid(False)
   i += 1
plt.show()
```

После перекодирования изображения в HSV нам необходимо получить понятный угол радиуса вектора (тон h) для каждого пикселя изображения. В данном случае для этого действия нам необходимо использовать вложенный цикл for, в котором мы будем анализировать и проходить изображение по ширине и высоте.

Пример кода с вложенным циклом for:

```
i=0; j=0
```

Инициализируем массив, в котором закладываем алгоритм нахождения тона h каждого пикселя изображения.

```
hues = []
for i in range(height):
   for j in range(width):
      hue = hsv[i][j][0] # значение тона пикселя по координатам (i, j)
      hues.append(hue)
```

После получения значения тона h помещаем его в датафрейм Pandas. В этом датафрейме по левую сторону указывался номер пикселя, а по правую –

значение его тона. На данном этапе датафрейм состоит из ряда элементов, расположенных в одном столбце.

Следующий этап кода состоит из преобразования тона в частоту, сопоставляется предопределённый набор элементов (частот значения переменной h). Данному этапу программного кода соответствует следующая функция:

```
scale_freqs = [220.00, 246.94 ,261.63, 293.66, 329.63, 349.23, 415.30]
def hue2freq(h,scale_freqs):
  thresholds = [26 , 52 , 78 , 104,  128 , 154 , 180]
  note = scale_freqs[0]
  if (h <= thresholds[0]):
     note = scale_freqs[0]
  elif (h > thresholds[0]) & (h <= thresholds[1]):
    note = scale_freqs[1]
  elif (h > thresholds[1]) & (h <= thresholds[2]):
    note = scale_freqs[2]
  elif (h > thresholds[2]) & (h <= thresholds[3]):
    note = scale_freqs[3]
  elif (h > thresholds[3]) & (h <= thresholds[4]):
    note = scale_freqs[4]
  elif (h > thresholds[4]) & (h <= thresholds[5]):
    note = scale_freqs[5]
  elif (h > thresholds[5]) & (h <= thresholds[6]):
    note = scale_freqs[6]
  else:
    note = scale_freqs[0]
  return note
```

Функция принимает значение переменной h и элементы массива частот. В данной части кода для определения частоты элемента используется массив, именованный как scale_freqs. Частоты этого массива полностью соответствуют тональности Ля минор.

Далее определяется значение порога, которое обозначено переменной thresholds для h, при помощи которой в дальнейшем будут преобразовывать h в частоту массива scale, посредством функции лямбды.

```
pixels_df['notes'] = pixels_df.apply(lambda row : hue2freq(row['hues'],scale_freqs)
```

Следующим логическим этапом идёт преобразование массива библиотеки NumPy в звуковое преображение. Столбец с массивом частоты преобразуется в массив NumPy, с помощью которого как раз можно сгенерировать аудиосигналы. Для этого используются SciPy – функции wavfile.write, поскольку массив имеет различный тип данных, для одномерного массива используется np.float32.

```
frequencies = pixels_df['notes'].to_numpy()
song = np.array([])
sr = 22050 # частота дискретизации
T = 0.1  # продолжительность
t = np.linspace(0, T, int(T*sr), endpoint=False) # определение переменной времени
#создание звукового преобразования NumPy :]
#nPixels = int(len(frequencies))# точки (пиксели) изображения
nPixels = 60
for i in range(nPixels):
   val = frequencies[i]
   note  = 0.5*np.sin(2*np.pi*val*t) # представление каждого элемента (ноты) в виде синусоиды
   song  = np.concatenate([song, note]) # добавляет ноту в массив (song) для объединения и создания звукового преобразования.
ipd.Audio(song, rate=sr) # загрузка цельного массива библиотеки NumPy в форме аудио.
```

Уже на данном этапе можно воспользоваться программой и получить композицию по изображению. Благодаря задействованию определённого объёма пикселей и изображения можно получить продолжительность и насыщенность трека. Но вариативности всё ещё не хватает. Далее следует добавить различные дополнительные возможности манипуляции с настройками.

2.1. Работа с пикселями

Внесём ещё один элемент в цикл, который позволит произвести сдвиг октав. То есть в определённый момент времени будет браться определённая октава для ноты из массива элементов. Для работоспособности данной идеи используется следующий программный код в цикле:

```
octave=random.choice(octaves)
```

Благодаря внесению данного элемента вносится больше разнообразия в аудио. Теперь можно работать с пикселями, поскольку изображение строится из сотни пикселей, мы можем произвольным образом генерировать схему, по которой программа будет выбирать случайным образом сетку точек и, обрабатывая их, воспроизводить мелодию.

```
val=octave * random.choice(frequencies)# в переменную val добавляем элемент случайной выборки пикселей.
```

Теперь можно внести куда больше разнообразия в настройку мелодии, полученную с изображения. В программе, приведённой выше, использовалась одна гамма Ля минор, но при всех плюсах данной гармоники всё равно происходит множество подходящих и однотипных композиций, чтобы получилось изменить однотипность, необходимо внести следующие изменения в программный компонент кода.

Заключение

В этой части работы были выполнены преобразования цветовых изображений в пиксельную сетку. Каждый пиксель содержит информацию о цвете, яркости и насыщенности. Также были выполнены преобразования нот, тональности и их частот в звуковое отображение. Найдены преобразования сетки пиксельной структуры в звуковое (частотное) отображение. Однако при настройке программы по гармонике Ля минор существует множество подходящих и однотипных композиций. Чтобы изменить однотипность, необходимо внести изменения в программный компонент кода.

Статья поступила в редакцию 10.08.2023 г.

ЛИТЕРАТУРА

1. Amani M., Falk H., Jensen O. D., Vartdal G., Aune A., Lindseth F. Color Calibration on Human Skin Images // International Conference on Computer Vision Systems. ICVS 2019 / eds. Tzovaras D., Giakoumis D., Vincze M., Argyros A. Cham: Springer, 2019. P. 211–223 (Series: Lecture Notes in Computer Science. Vol. 11754). DOI: 10.1007/978-3-030-34995-0_20.
2. Чижов С. А. Управление цветом в широкоформатной печати: RGB VS CMYK // Science Time. 2018. № 9 (57). С. 43–47.
3. Open music theory. Version 2.0 / M. Gotham, K. Gullings, Ch. Hamm et al. Oklahoma State University, 2021 [Электронный ресурс]. URL: https://viva.pressbooks.pub/openmusictheory/ (дата обращения: 04.06.2023).

REFERENCES

1. Amani M., Falk H., Jensen O. D., Vartdal G., Aune A., Lindseth F. Color Calibration on Human Skin Images. In: Tzovaras D., Giakoumis D., Vincze M., Argyros A., eds. International Conference on Computer Vision Systems. ICVS 2019. Cham, Springer, 2019, pp. 211–223 (Series: Lecture Notes in Computer Science. Vol. 11754). DOI: 10.1007/978-3-030-34995-0_20.
2. Chizhov S. A. [Color management in large-format printing: RGB VS CMYK]. In: *Science Time*, 2018, no. 9 (57), pp. 43–47.
3. Gotham M., Gullings K., Hamm Ch. et al. Open music theory. Version 2.0. Oklahoma State University, 2021. Available at: https://viva.pressbooks.pub/openmusictheory/ (accessed: 04.06.2023).

ИНФОРМАЦИЯ ОБ АВТОРАХ

Ключников Семён Александрович – студент магистратуры физико-математического факультета Государственного университета просвещения;
e-mail: semen.klyuchnikov@mail.ru;

Калашников Евгений Владимирович – доктор физико-математических наук, профессор кафедры вычислительной математики и информационных технологий Государственного университета просвещения;
e-mail: ekevkalashnikov1@gmail.com.

INFORMATION ABOUT THE AUTHORS

Semen A. Klyuchnikov – Master's Degree Student, Faculty of Physics and Mathematics, Federal State University of Education;
e-mail: semen.klyuchnikov@mail.ru;

Evgenii V. Kalashnikov –Dr. Sci. (Phys.-Math.), Prof., Department of Computational Mathematics and Information Technology, Federal State University of Education;
e-mail: ekevkalashnikov1@gmail.com.

ПРАВИЛЬНАЯ ССЫЛКА НА СТАТЬЮ

Ключников С. А., Калашников Е. В. Преобразование пиксельной структуры в звуковые отображения. Часть 1 // Вестник Государственного университета просвещения. Серия: Физика-Математика. 2023. № 4. С. 64–80.
DOI: 10.18384/2949-5067-2023-4-64-80

FOR CITATION

Klyuchnikov S. A., Kalashnikov E. V. Converting a pixel structure into sound imaginations. Part 1. In: *Bulletin of Federal State University of Education. Series: Physics and Mathematics*, 2023, no. 4, pp. 64–80.
DOI: 10.18384/2949-5067-2023-4-64-80