Introdução à computação musical

Cláudio Gomes[1]

[1]Departamento de Ciências Exatas e Tecnologias - Universidade Federal do Amapá (UNIFAP)
Caixa Postal 261 – 68.906-970 – Macapá – AP – Brazil

claudiorogerio@unifap.br

Resumo. *A computação musical tem o papel de facilitar as diversas formas de geração de criatividade artística, informação e inovação através de métodos totalmente diversificados. Atualmente, a computação musical atua por diversas ações unilaterais sobre barreiras de criatividade e tecnológicas. Este minicurso tem o objetivo de apresentar a grande área de computação musical para o público interessado desde a teoria musical, fundamentos eletroacústica e geração de sinais, tópicos atuais de interesse, exemplos por diversas linguagens de programação e ainda comentários sobre diversas atuações em universidades brasileiras através de seus laboratórios de pesquisa.*

Palavras-chave — instrumento musical digital, recuperação da informação musical, sistemas interativos em tempo real

Abstract. *Computer music has the role of facilitating the various forms of generating artistic creativity, information and innovation through totally diversified methods. Currently, computer music acts in several unilateral actions on creativity and technological barriers. This workshop aims to present the great area of musical computing to the interested public from music theory, electroacoustic fundamentals and signal generation, current topics of interest, examples by various programming languages and even comments on various performances in Brazilian universities through of their research labs.*

Keywords — digital musical instruments, music information retrieval, real-time interactive systems

1 Introdução

A computação musical é uma área interdisciplinar da Ciência da Computação agregada com diversas outras áreas tais como a Música. Muito próximo da arte, a computação musical adiquire o status de gerador de conteúdo científico e artístico permitindo discussões sobre o "fazer ciência" para a computação musical. Dessa forma, há um vasto campo de atuação desde que viabiliza soluções usuais, performáticas quanto teóricas.

Atualmente, os principais tópicos de interesse incluem: inteligência artificial, processamento de sinais digitais, composição musical facilitada, modelagem acústica, bibliotecas musicais, estrutura e representação de dados musicais e instrumentos musicais digitais. Neste minicurso, abordou-se os tópicos de visão geral sobre teoria musical, construção de instrumentos digitais musicais comparado por diferentes modelos de programação e ainda discussões sobre recuperação de informação musical.

2 Visão geral sobre teoria musical

Os intrumentos musicais apresentam sonoridades, formatos, limites, estudos e definições totalmente únicas. Conforme a figura 1, cada instrumento apresenta um modelo temporal e frequência que é único e exclusivo. Dessa forma, a sonoridade pode-se definir como parte integrante do timbre encontrado por cada instrumento. O timbre é um registro único e peculiar para cada instrumento. No entanto, conforme a figura 2, por diversos motivos, cada instrumento tem sua faixa confortável de geração de possíveis notas. Ou seja, além das características timbristas, os instrumentos diferenciam-se pelo alcançe em frequências maiores/menores, permitindo o concesso e auxílio de quais instrumentos pode-se optar na utilização em uma banda musical, por exemplo, para que não sobrepunham a sonoridade entre os instrumentos selecionados.

Figura 1: Instrumentos e timbres

Digamos que um saxofonista queira tocar uma música escrita para contra-baixo, conforme a figura 2, os intrumentos do saxofone e contra-baixo estão em faixas de frequências bastantes distintas, sendo necessário algumas modificações como a mudança por oitavas e tonalidades para que o saxofonista pode-se tocar uma música escrita para contra-baixo. Caso o saxofonista, utiliza-se as mesmas notas e mesma oitava do contra-baixo, por exemplo, a sonoridade poderia ter um resultado não satisfatório. Assim, a transcrição musical adapta-se a música como referência original, para a notação de instrumento desejado.

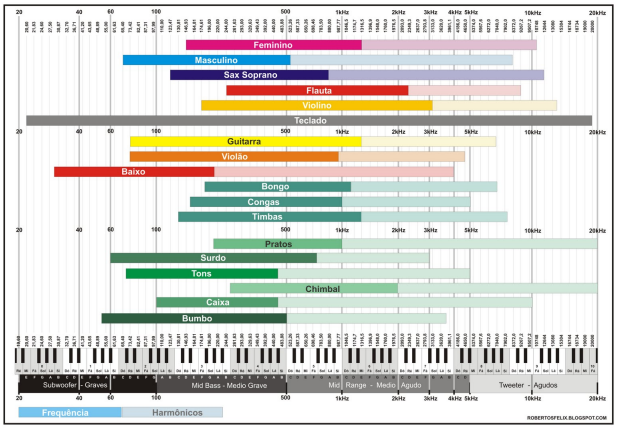

Figura 2: Instrumentos e frequências [1]

3 Fundamentos de instrumentos musicais digitais – IMD

Diversas linguagens de programação não foram construídas para o propósito específico de produção musical. Logo, a computação

musical tornou-se uma necessidade e linguagens de programação foram adaptadas porém, ainda não ideais para diversos cenários, permitindo o espaço para o surgimento de linguagens com diversos propósitos musicais. Dessa forma, C, C++, Python, Java e outras linguagens criaram bibliotecas com propósito musical, assim como o surgimento de linguagens computacionais e musicais como CSound, SuperCollider, Faust, Chuck, Processing, Overtone, Pure Data, Max/SP, etc. [2, 3, 4, 5, 6, 7].

A linguagem de programação com propósito geral C pode ser utilizada para diversas ações tais como para a música. Na figura 3, apresenta-se um exemplo de codificação de geração sonora a partir da biblioteca *OpenAL* [8], na qual defini-se a frequência de execução [1].

```
 4 #include <stdio.h>
 5 #include <stdlib.h>
 6 #include <math.h>
 7 #ifdef __APPLE__
 8 #include <OpenAL/al.h>
 9 #include <OpenAL/alc.h>
10 #elif __linux
11 #include <AL/al.h>
12 #include <AL/alc.h>
13 #include <unistd.h>
14 #endif
15
16 ALCdevice  * openal_output_device;
17 ALCcontext * openal_output_context;
18 ALuint internal_buffer;     // buffer sonoro
19 ALuint streaming_source[1]; // stream gerado
20
21 // Funcao padrao para erros
22 int al_check_error(const char * given_label) {
23     ALenum al_error;
24     al_error = alGetError();
25     if(AL_NO_ERROR != al_error) {
26         printf("ERROR - %s  (%s)\n", alGetString(al_error), given_label);
27         return al_error;
28     }
29     return 0;
30 }
31
32 // inicializacao
33 void MM_init_all() {
34
35     const char * defname = alcGetString(NULL, ALC_DEFAULT_DEVICE_SPECIFIER);
36
37     openal_output_device  = alcOpenDevice( defname );
38     openal_output_context = alcCreateContext( openal_output_device, NULL );
39     alcMakeContextCurrent( openal_output_context );
40     // setup buffer and source
41     alGenBuffers(1, & internal_buffer );
42     al_check_error("failed call to alGenBuffers");
43 }
```

Figura 3: Exemplo de geração sonora em C (p.1).

Percebe-se que em C, são necessários os seguintes passos para a geração de um som sintético: **inicialização** - definição do dispositivo sonoro, criação de um ambiente ou contexto, ativar o ambiente, criar áreas que receberão informações de áudio, os *buffers*, (*MM_initi_all()*: linhas 38-50); **geração de fonte sonora** - definição temporal, taxa de amostragem, variável de *samples* conforme a dimensão definida, geração temporal da senoide (linha 80) com possíveis ajustes, atualização do *buffer* pelos *samples* de senoides, criação de transmissão e recepção do *buffer*, executar a transmissão (linha 96), análise de execução da transmissão do áudio (*MM_one_buffer(float): linhas 65-112*); **término** - suspende a transmissão, retira dados da transmissão, deleta o ambiente de transmissão o contexto e dispositivo (*MM_exit_all(): linhas 46-62*). Há vários testes que estão comentados no código para a compreensão de outras possibilidades de ações com os *buffers*.

A linguagem de programação ChucK [9] tem o propósito musical com funcionalidades de síntese, composição, análise e desempenho em tempo real. Uma das principais vantagens na utilização do ChucK é a programação sincronizada ao tempo de precisão de controle musical além da adição

[1] O arquivo completo pode ser acessado em: *https://github.com/claudiorogerio/sintese_sonora*

```
46 void MM_exit_all() {
47
48     ALenum errorCode = 0;
49     // Stop the sources
50     alSourceStopv(1, & streaming_source[0]);        //      streaming_source
51     int ii;
52     for (ii = 0; ii < 1; ++ii) {
53         alSourcei(streaming_source[ii], AL_BUFFER, 0);  }
54     // Clean-up
55     alDeleteSources(1, &streaming_source[0]);
56     alDeleteBuffers(16, &streaming_source[0]);
57     errorCode = alGetError();
58     alcMakeContextCurrent(NULL);
59     errorCode = alGetError();
60     alcDestroyContext(openal_output_context);
61     alcCloseDevice(openal_output_device);
62 }
63
64 // gerador de senoides
65 void MM_render_one_buffer( float freq ) {
66
67     float incr_freq = 0.06f;
68     int seconds = 4;
69     // unsigned sample_rate = 22050;    // testes com diferentes rates
70     unsigned sample_rate = 44100;
71     double my_pi = 3.14159;
72     size_t buf_size = seconds * sample_rate;
73
74     // allocate PCM audio buffer vetor de samples
75     short * samples = malloc(sizeof(short) * buf_size);
76
77     printf("\nFreq senoide: %f\n", freq);
78     int i=0;
79     for(; i<buf_size; ++i) {
80         samples[i] = 32760 * sin( (2.f * my_pi * freq)/sample_rate * i );
81         freq += incr_freq;
82         if (100.0 > freq || freq > 2000.0)
83             incr_freq *= -1.0f;  // toggle direction of freq increment
84         if (3000.0 > freq || freq > 5000.0)
85             incr_freq *= 1.0f;  // toggle direction of freq increment
86     }
```

Figura 4: Exemplo de geração sonora em C (p.2).

```
87
88     /* atualizando o internal_buffer com a senoide criada para o OpenAL */
89     alBufferData( internal_buffer, AL_FORMAT_MONO16, samples, buf_size, sample_rate);
90     al_check_error("populating alBufferData");
91
92     free(samples);
93
94     alGenSources(1, & streaming_source[0]); // cria um stream
95     alSourcei(streaming_source[0], AL_BUFFER, internal_buffer); // adiciona buffer ao stream
96     alSourcePlay(streaming_source[0]); // play stream
97
98     // --------------------
99
100    ALenum current_playing_state;
101    alGetSourcei(streaming_source[0], AL_SOURCE_STATE, & current_playing_state);
102    al_check_error("alGetSourcei AL_SOURCE_STATE");
103
104    while (AL_PLAYING == current_playing_state) {
105
106        printf("playing ... so sleep\n");
107        sleep(1);   // should use a thread sleep NOT sleep() for a more responsive finish
108        alGetSourcei(streaming_source[0], AL_SOURCE_STATE, & current_playing_state);
109        al_check_error("alGetSourcei AL_SOURCE_STATE");
110    }
111
112    printf("end of playing\n");
113
114    MM_exit_all();
115 }
116
117 int main() {
118
119    MM_init_all();
120    MM_render_one_buffer( 400.0 );
121 }
```

Figura 5: Exemplo de geração sonora em C (p.3).

de variáveis de manipulação temporal bastante úteis durante a programação É uma linguagem multiplataforma integrada com outras linguagens de programação com grande flexibilidade, improvisação, permitindo o uso em diversos cenários artísticos.

A figura 6 apresenta um exemplo dos tipos de variáveis fundamentais úteis na codificação na linguagem ChucK. Percebe-se que a principal mudança está na sintaxe orientada da esquerda para a direita. Além disso, há variáveis de controle temporal (variável *dur, time*) e do ambiente de contexto (variável *now*).

A figura 7 apresenta um exemplo a partir de funções de geradora de sinal impulso. Para esse exemplo, foi necessário passar o sinal impulso do filtro *BiQuad* para proteger e manter o ganho do sinal. Depois, avaliado pelo *Pan2*, que define a localização 2D da fonte sonora para a placa sonora (*Dac - Digital Analogic Conversor*). Após configurações do filtro BiQuad (linha

5-7), define-se a força de geração do impulso sempre constante, porém a frequência sonora e a localização da fonte sonora têm funções individuais aleatórias.

```
// variaveis
<<<"","">>>;
<<<"","">>>;
-1 => int i;
<<< i >>>;
1000 => i;
<<< i >>>;
0xA0 => i; // hexadecimal
<<< i >>>;
5.678 => float f; // float
<<< f >>>;
9.0::second => dur d;
<<< d >>>;
now => time t;
<<< t >>>;

<<< "Waiting 3 seconds..." >>>;
3::second => now;  // stop de 3 segundos
60::ms => dur t_control;
<<< "Info:", now, t_control, now/t_control >>>;
0.1::second => now;  // variavel de controle do tempo
<<< "Info:", now, t_control, now/t_control >>>;

0 => int j;
0 => int i;
do {
    2 + => j;
    1 + => i;
    <<< i, j >>>;
} until ( i >= 5 );
<<< "Final", i, j >>>;
if ( j == 5 ) <<< "success" >>>;
```

Figura 6: Exemplo de variáveis em ChucK.

```
// sinal impulso para filtro para Posicao sonora

Impulse i => BiQuad f => Pan2 p => dac;

.99 => f.prad; // set filtro
1 => f.eqzs; // set equal gain zeros
.5 => f.gain; // set filter gain

while( true ) {
    // forca do impulso
    1.0 => i.next;

    // frequencia aleatoria
    Math.random2f( 250, 5000 ) => f.pfreq;
    // direcao da fonte
    Math.random2f( -1, 1 ) => p.pan;
    // tempo de pausa
    300::ms => now;
}
```

Figura 7: Exemplo de sinais sonoros em ChucK.

O ChucK apresenta diversas funções que facilitam o desenvolvimento e ações tais como gerar sons a partir de sintetizadores pré-configurados de STK (do inglês - *Synthesis ToolKit*). O STK é uma ferramenta de código de fonte aberto com vários timbres: flauta, sax, percussão, etc. Dessa forma, o ChucK tem variáveis na utilização do STK nativo, conforme o exemplo na figura 8.

A partir da linguagem Python, pode-se gerar ondas sonoras audíveis com biblioteca *IPython*. Conforme a figura 9, criou-se uma função que cria senoides e permite a soma de senoides baseada em posições na escala de notas musicais. Na linha 9, a variável *f0* tem a informação de frequência fundamental com base na equação de posição de notas musicais que será aplicada na função de senoide, linha 13. Na linha 26, executa-se a função descrita anteriormente, para em seguida gerar a visualização em 2D e ao final, o resultado para a fonte sonora, nas linhas 38 e 39.

Em resumo, cada linguagem de programação tem vantagens e desvantagens que deve-se avaliar qual o melhor paradigma de programação para utilizá-las em determinado cenário. É possível encontrar diversos exemplos, tutoriais nas referências

```
Flute flute => PoleZero f => JCRev r => dac;
.55 => r.gain;
.05 => r.mix;
.99 => f.blockZero;
// notas midis
[ 61, 63, 63, 60, 65, 66, 68 ] @=> int notes[];
while( true ) {
    flute.clear( 1.0 ); // novas configuracoes aleatorias do instr
    Math.random2f( 0, 1 ) => flute.jetDelay;
    Math.random2f( 0, 1 ) => flute.jetReflection;
    Math.random2f( 0, 1 ) => flute.endReflection;
    Math.random2f( 0, 1 ) => flute.noiseGain;
    Math.random2f( 0, 1 ) => flute.vibratoFreq;
    Math.random2f( 0, 1 ) => flute.vibratoGain;
    Math.random2f( 0, 1 ) => flute.pressure;
    <<< "---", "" >>>;
    <<< "jetDelay:", flute.jetDelay() >>>;
    <<< "jetReflection:", flute.jetReflection() >>>;
    <<< "endReflection:", flute.endReflection() >>>;
    <<< "noiseGain:", flute.noiseGain() >>>;
    <<< "vibratoFreq:", flute.vibratoFreq() >>>;
    <<< "vibratoGain:", flute.vibratoGain() >>>;
    <<< "breath pressure:", flute.pressure() >>>;
    for( int i; i < notes.size(); i++ ) {
        // duracao de cada nota aleatoriamente
        Math.random2f( .75, 2 ) => float factor;
        <<< "Tocar nota", notes[i] >>>;
        play( notes[i], Math.random2f( .6, .9 ) );  // chama funcao play()
        300::ms * factor => now;    }
}
// funcao play
fun void play( float note, float velocity ) {
    // midi para frequencia
    Std.mtof( note ) => flute.freq;
    velocity => flute.noteOn;
}
```

Figura 8: Exemplo de STK em ChucK.

```
import numpy as np

def timbre( steps, base, add, fs ):
    pos= [0, 11, 23, 35, 47, 52]
    ts = 1/fs # período de amostragem
    duracao = 1   # em segundos
    n1 = np.arange( fs*duracao ) # domínio do tempo discreto

    f0 = base*np.power( 2, ((0+pos[steps])/12) ) # f0 de cada nota
    amostra = np.sin( 2*np.pi*f0*n1*ts )
    if add:
        f0 = base*np.power( 2, ((0+pos[0])/12) ) # f0 de cada nota
        amostra = np.sin( 2*np.pi*f0*n1*ts )
        if steps >= 1:
            for f in np.arange(steps)+1:
                f1 = base*np.power( 2, ((0+pos[f])/12) )
                amostra += (1/f)*np.sin( 2*np.pi*(f1)*n1*ts )
                print( "Senoides somadas", f, pos[f], f0, f1 )
    return amostra

import matplotlib.pyplot as plt
sr = 44100
harmonicos = 5
f0 = 220.0
somar_harmonicos = True
y = timbre( harmonicos, f0, somar_harmonicos, sr )
plt.figure( 1,figsize=(12,4) )
plt.subplot(121)
plt.plot(y)
plt.xlabel( 'Frame' )
plt.ylabel( 'Amplitude' )

plt.subplot(122)
plt.plot( y[0:800] )
plt.xlabel( 'Frame' )
plt.ylabel( 'Amplitude' )

from IPython.display import Audio
Audio( data = y, rate= sr )
```

Figura 9: Exemplo de sinais sonoros em Python.

de cada linguagens e suas respectivas bibliotecas.

3.1 Recuperação de Informação Musical – RIM

O tópico interdisciplinar de Recuperação de informação Musical – RIM (do inglês: *Music information Retrieval* – MIR) vem em continua evolução de ferramentas e modelos permitindo diversas aplicações e soluções científicas e mercadológicas. Atualmente, essas soluções são combinações com outros tópicos de suporte como: musicologia, psico-acústica, psicologia, processamento de sinais, aprendizado de máquina. Classificação de ritmos, sistemas de recomentações musicais, identificação de fontes musicais ou instrumentos, sincronização temporal, automática transcrição de músicas e geração musical são algumas das possíveis caterogias em RIM.

Em RIM, basicamente extrai-se características do

áudio para utilização em um modelo definido. Dessa forma, em grande maioria, é necessário uma base de dados para entender as propriedades de cada áudio. A partir de uma fonte pode-se, basicamente, extrair características e identificar propriedades musicais e não-musicais, além de várias operações dependentes de cada solução. Há diversos tipos de extratores de características de uma fonte que tem funcionalidades similares tais como: MFCC – *Mel-Frequency Cepstral Coefficients*, RMS – *root-mean-square*, CENS – *chroma energy normalized*, CQT – *Constant-Q chromagram* etc. e demais extratores podem ser encontrados com mais detalhes em Lerch et. al. 2019 [10].

Há diversas bibliotecas ou ferramentas ágeis para auxiliar na extração de características de áudio. A a biblioteca LibRosa [11] apresenta facilidade e ferramentas atualizadas. Conforme a figura 10, percebe-se que a diferença na utilização de cada extrator, está na própria chamada da função de extração de característica. Dessa forma, vale ao pesquisador aprofundar quais as características de fato são importantes para seu projeto.

```
import librosa

y, sr = librosa.load( 'file.wav', sr=None, mono=True )

print( y, sr )

four_tempog = librosa.feature.fourier_tempogram( y= y, sr=sr )
tempog = librosa.feature.tempogram( y = y, sr=sr )
chrom_stft = librosa.feature.chroma_stft( y = y, sr = sr, n_fft=2048, hop_length=512 )
chrom_cens = librosa.feature.chroma_cens(y = y, sr = sr, hop_length=512 )
chrom_cqt = librosa.feature.chroma_cqt(y = y, sr = sr, hop_length=512 )
mel = librosa.feature.melspectrogram( y = y, sr=sr, n_mels=128 )
mfcc = librosa.feature.mfcc( y= y, sr=sr, n_mfcc=20 )
```

Figura 10: Extraindo características de áudios em Python.

No entanto, quando tem-se a necessidade de realização de soluções para os tópicos de identificação de padrões seja para acordes, ritmos, emoções, timbres etc. tem-se em, uma abordagem, a necessidade de realização de extração de características de áudio de vários arquivos, geralmente confeccionados em uma base de dados. Logo em seguida, parte ao processo inteligível (regressões, lógicas, redes neurais, aprendizado de máquina etc.) que depende do cenário e das particularidades da base de dados. A conferência ISMIR, por exemplo, apresenta uma lista de várias bases de dados para diferentes atuações [2].

4 Conclusões

Neste minicurso, tentou-se cativar leigos e curiosos de diversas áreas para o tópico de computação musical. Para isso, utilizou-se a estratégia de inicializar fundamentos de teoria musical, características sonoras, ferramentas livres de edição e mixagem, programaçao para instrumentos musicais digitais de diferentes maneiras e por fim, apresentar características de recuperação de informação musical, um dos tópicos em grande movimentação no cenário internacional e nacional. Sugere-se consultas nas fontes no SBCM, Vortex Music Journal, ENCM, Revista Eletrônica de Iniciação Científica em Computação (REIC), International Society for Music Information Retrieval (ISMIR), International Country Music Journal (ICMC), Journal of New Research (JNMR), New Interfaces for Musical Expression (NIME), Ubiquitous Music Symposium (UBIMUS), Music Information Retrieval Evaluation eXchange (MIREX).

Referências

[1] Forum Som ao vivo. Disponível em: http://www.somaovivo.org. *Acesso em 10.09.2021.*

[2] Rodrigo Ramos de Araujo, José Mauro da Silva Sandy, Elder José Reioli Cirilo, and Flávio Luiz Schiavoni. Análise e classificação de linguagens de programação musical. *Revista Vórtex*, 6(2), 2018.

[3] Clenio B Goncalves Junior and Murillo Rodrigo Petrucelli Homem. Um ambiente de programação multiparadigma voltado à composição musical automática. *Latin American Journal of Development*, 3(1):447–462, 2021.

[4] Perry Cook. 2001: Principles for designing computer music controllers. In *A NIME Reader*, pages 1–13. Springer, 2017.

[5] Felicia Nafeeza Persaud. *In Search of Computer Music Analysis: Music Information Retrieval, Optimization, and Machine Learning from 2000-2016*. PhD thesis, Université d'Ottawa/University of Ottawa, 2018.

[6] Clemens Wöllner. *Body, sound and space in music and beyond: multimodal explorations*. Taylor & Francis, 2017.

[7] Atau Tanaka and Miguel Ortiz. Gestural musical performance with physiological sensors, focusing on the electromyogram. 2017.

[8] Garin Hiebert, K Charley, P Harrison, JM Jot, D Peacock, JM Trivi, and C Vogelsang. Openal programmer's guide. *URL: http://connect. creativelabs. com/openal/Documentation/Forms/AllItems. aspx*, 2007.

[9] Ge Wang, Perry R Cook, and Spencer Salazar. Chuck: A strongly timed computer music language. *Computer Music Journal*, 39(4):10–29, 2015.

[10] Alexander Lerch, Claire Arthur, Ashis Pati, and Siddharth Gururani. Music performance analysis: A survey. *arXiv preprint arXiv:1907.00178*, 2019.

[11] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, pages 18–25. Citeseer, 2015.

[2] ISMIR – Datasets: *https://ismir.net/resources/datasets/*