Universidade Federal Da Paraíba – UFPB

Centro de Informática

Programa De Pós-graduação Em Computação, Comunicação E Artes (PPGCCA)

DANIEL SORRENTINO MARQUES

GLITCH ART E PURE DATA: UTILIZANDO PURE DATA PARA TRANSFORMAR O PROCESSO DE CRIAÇÃO DE GLITCH EM FERRAMENTA

João Pessoa

Novembro /2018

Daniel Sorrentino Marques

GLITCH ART E PURE DATA: UTILIZANDO PURE DATA PARA TRANSFORMAR O PROCESSO DE CRIAÇÃO DE GLITCH EM FERRAMENTA

Dissertação apresentada ao Programa de Pós-Graduação em Computação, Comunicação e Artes (PPGCCA) da Universidade Federal da Paraíba, como requisito parcial para a obtenção do título de Mestre em Computação, Comunicação e Artes, na linha de pesquisa arte computacional.

Orientador: Prof. Dr. Carlos Eduardo Coelho Freire Batista

Orientador: Prof. Dr. José Amâncio Tonezzi Rodrigues Pereira

João Pessoa
Novembro /2018

Catalogação na publicação
Seção de Catalogação e Classificação

M357g Marques, Daniel Sorrentino. Glitch art e Pure Data: utilizando Pure Data para transformar o processo de criação de glitch em ferramenta / Daniel Sorrentino Marques. - João Pessoa, 2019.
125 f. : il.

Orientação: Carlos Eduardo Coelho Freire Batista Batista, José Amâncio Tonezzi Rodrigues Pereira Pereira.
Dissertação (Mestrado) - UFPB/CI.

1. Pure Data. 2. Glitch art. 3. Arte computacional. I. Batista, Carlos Eduardo Coelho Freire Batista. II. Pereira, José Amâncio Tonezzi Rodrigues Pereira. III. Título.

UFPB/BC

Daniel Sorrentino Marques

GLITCH ART E PURE DATA: UTILIZANDO PURE DATA PARA TRANSFORMAR O PROCESSO DE CRIAÇÃO DE GLITCH EM FERRAMENTA

Dissertação apresentada ao Programa de Pós-Graduação em Computação, Comunicação e Artes da Universidade Federal da Paraíba, como requisito parcial para a obtenção do título de Mestre em Computação, Comunicação e Artes, na linha de pesquisa arte computacional.

A banca considera o presente Trabalho Final: aprovado

Data: 20 / 02 /2019.

Prof. Dr. Carlos Eduardo Coelho Freire Batista
(Orientador – PPGCCA/UFPB)

Prof. Dr. José Amâncio Tonezzi Rodrigues Pereira
(Orientador – PPGCCA/UFPB)

Prof. Dr. Francisco de Paula Barretto (Universidade Federal da Bahia (UFBA)

Profa. Dr Tiago Maritan Ugulino de Araújo (PPGCCA/UFPB)

DEDICATÓRIA

Dedico este trabalho to all the dreamers like me, somehow, with a little help,we can make in.

AGRADECIMENTOS

Agradeço em especial aos meus orientadores Tonezzi e Bidu que sem a paciência e a dedicação de ambos nada disso seria possível, além dos pesquisadores Francisco Barreto, Thiago Maritan e Guilherme Schulze com suas valorosas correções, sugestões e discussões. Agradeço ao apoio constante de minha mãe Gianna, meu pai Alberto e minhas irmãs Beatriz e Olga. Gostaria de agradecer também o apoio de minha companheira Bárbara, meu cachorro Beluga, aos meus amigos Norma, Italo, Thiago, João Paulo, Francisco, Isis, Raquel, Igor entre tantos outros presentes que sempre apoiaram e acreditaram nessa nossa empreitada acadêmica e meus colegas de mestrado Luna, Aluízio, Renata, Ely.

RESUMO

A interação da arte com a digitalização da mídia gera um espectro abrangente de arte digital. Dentro deste espectro, dissertamos sobre glitch art, uma forma de arte digital que trabalha com a introdução intencional de erros em suas obras de arte. Especificamente, a mídia digital foi estudada para fazer uma produção criativa, usando, subvertendo ou listando soluções para criação artística através da computação. Desenvolvemos um programa de computador utilizando a linguagem de programação visual Pure Data, que transforma o processo de criação de glitch art em algo prático para o artista, mostrando resultados que possam ser explorados visualmente e com menos atrasos. Esse novo método oferece menor risco de destruição dos arquivos e reduz a carga de conhecimento computacional necessária para causar uma falha com intenção artística. Adicionalmente, investigamos as possibilidades que este programa facilitador de glitch cria para novas interações dentro dos dois campos de conhecimento focados neste projeto: arte e computação.

Palavras Chaves: Pure Data, *Glitch Art,* arte computacional

ABSTRACT

The interaction of art with the digitization of the media generates a comprehensive spectrum of digital art. Within this spectrum, will lectured on glitch art, a digital art form that works with the intentional introduction of errors in his works of art. Specifically, digital media was studied to make a creative production, using, subverting or listing solutions for artistic creation through computing. We developed a computer program using the visual programming language Pure Data, which transforms the process of creating glitch art into something practical for the artist, showing results that can be explored visually and with less delay. This new method offers less risk of destruction of the files and reduces the computational knowledge load necessary to cause a failure with artistic intention. Additionally, we investigate the possibilities that this glitch facilitator creates for new interactions within the two fields of knowledge focused on this project: art and computing.

Keywords: Pure Data, Glitch Art, Computacional Art

LISTA DE ILUSTRAÇÕES

II.

II.1

II.2

IV.3

IV.4

IV.5

IV.7

V

LISTA DE TABELAS

SUMÁRIO

I. INTRODUÇÃO

A produção artística contemporânea ainda reverbera as diferentes ondas de impacto produzidas pela digitalização das mídias. Depois de traduzir o que existia no mundo analógico, o processo de criação no meio digital hoje define métodos e mecanismos particulares, que são determinantes para estabelecer o universo de possibilidades do que se cria, inclusive esteticamente. Dentro desse espectro, destacamos então a *glitch art*, forma de arte digital que trabalha com a introdução proposital de erros em suas obras.

Trabalhamos com a linha de pesquisa: arte computacional, na qual mídias digitais foram estudadas, a fim de fazer uma produção criativa, utilizando, subvertendo ou elencando soluções para a criação artística através da computação.

Nesta dissertação, a *glitch art* foi explorada utilizando a corruptibilidade de dados de um arquivo JPEG[1], por meio de um programa desenvolvido para facilitar a criação de *glitch* diminuindo a carga de conhecimento computacional necessária, buscando obter o aspecto estético de *glitch art.* A linguagem de programação escolhida para o desenvolvimento dessa ferramenta foi o *Pure Data,* uma linguagem de programação visual, que possui um ambiente de desenvolvimento popular, de ampla utilização, maturidade e flexibilidade para se acomodar a cenários de usos distintos.

No capítulo II, dissertamos sobre a linguagem de programação visual *Pure Data* com foco em sua usabilidade para o desenvolvimento das ferramentas para as experimentações desta pesquisa. Vimos seu ambiente de funcionamento e suas principais características. Fizemos também uma análise da estrutura JPEG, que é o formato de arquivo que elencamos trabalhar para a versão final da ferramenta desenvolvida.

Fizemos uma análise da falha, do *glitch,* do movimento artístico que explora a estética do erro, a *glitch art*, traçando movimentos artísticos da estética *glitch* do seu início analógico para o foco da pesquisa, a sua presença no âmbito digital. Reproduzimos os principais procedimentos encontrados em nossa busca bibliográfica para gerar *glitch* a fim de verificar seus resultados e de desenvolver o nosso próprio algoritmo. Verificamos o uso de *glitch* criado de forma textual explorando o código ASCII via notepad, Notepad++ e finalmente Frhed.

[1] *joint photographic experts group image*. In: https://jpeg.org/jpeg/

No capítulo III apresentamos nossa metodologia utilizada para realizar os experimentos, bem como a pesquisa bibliográfica

No capítulo IV, verificamos a sinestesia da *glitch art* experimentando com a edição de imagens em editores de áudio. Desenvolvemos experimentações com *glitch art* em seus variados formatos de mídias digitais, uma música *glitch,* um vídeo *glitch* e imagens e experimentamos com os principais métodos de *glitch* listados na literatura levantada na bibliografia.

Por fim, no capítulo V desenvolvemos *patches* de *Pure Data* com características de *glitch-like*, que nos auxiliaram no desenvolvimento de nossos protótipos geradores de *true-glitch,* assim desenvolvendo a nossa ferramenta final.

No capítulo VI apresentamos resultados dos experimentos e a análise acerca dos dados obtidos. No VII capítulo escrevemos sobre as conclusões e trabalhos futuros.

I.1 Justificativa

A importância da compreensão dos processos de digitalização das mídias que geram novas expressões artísticas, em específico a *glitch art*, é o motriz dessa pesquisa. Entender a falha é necessário, pois possibilita a geração de novos modos de pensamentos e ações. Os erros geram dúvidas, tornando-se assim, meios de sugestão de novos caminhos na criação artística. Estudos sobre falhas, *glitch* ou a utilização de *glitch* na arte: *glitch art* são estudos de uma estética que se destrói por sua própria escolha. É possível ver a falha apenas como uma coisa tecnológica, mas devemos percebê-la nesse estudo como uma construção cultural. É preciso haver mais pesquisas na arte dos artefatos *glitch* e sua estética transmidiática e sinestésica, que, no geral, inclui o uso artístico de artefatos digitais.

Glitch art é um movimento de novas expressões, uma linguagem sempre crescente, um movimento que busca desconstruir as normas, presunções e expectativas inerentes a uma linguagem, formato ou mídia. *Glitches* vêm da percepção humana, um elemento cultural ou depreciamento

funcional, porém o que foi uma falha no passado pode não ser mais uma falha no presente. Essa contingência ambígua vai da falha a construção, operação e conteúdo a vivência do artista de *glitch art* e essas interpretações mudam de acordo com o meio e, por consequência, sua interpretação pelo leitor, visualizador ou usuário. Assim, deixa de ser apenas o erro de uma máquina, que aparece aleatoriamente em uma obra, para ser uma forma de arte em constante mudança que depende das interações entre formas digitalizadas de expressões artísticas, essa dinâmica cultural, estética e de formatos e de mídia como textos, sons, imagens, vídeos. São inovações estéticas motivadoras do estudo deste tipo de conteúdo.

Para dar suporte às explorações com *glitch art*, foi desenvolvido um conjunto de ferramentas, utilizando a linguagem *Pure Data*, que visam facilitar o processo de criação, tornando a corrupção de arquivos JPEG algo mais acessível, por meio do desenvolvimento de um programa de computador que transforma a introdução de erros arbitrários em um arquivo em algo rápido e prático.

Pure data é uma ferramenta de trabalho de código aberto, gratuita e com grande amplitude de funções que o ambiente de desenvolvimento engloba lhe dá a complexidade necessária para a criação de diversas ferramentas artísticas, Por ser um ambiente de desenvolvimento e uma linguagem de programação visual voltada para criações artísticas por artistas, o *Pure Data* se mostra ideal para o desenvolvimento da ferramenta de *glitch art* que entregamos ao fim desta dissertação. Ao facilitar o uso através de sua interface de programação visual podemos ter uma compreensão diferenciada do processo de construção da ferramenta, e da compreensão de ferramentas já existentes feitas nessa linguagem.

O estudo de *Pure Data*[2] como um facilitador da *glitch art* é relevante, pois esta última "estimula novos métodos de criação, e foge de *templates* pré estabelecidos; foge de ações estabelecidas por *scripts*, e se junta ao *avant-garde* do desconhecido." (MENKMAN 2011, p.11)

[2] Pure Data https://puredata.info/ , uma linguagem de programação visual.

I.2 Objetivo Geral

Criar uma ferramenta para simplificar o processo de criação de *glitch art,* utilizando a linguagem *Pure Data,* para o desenvolvimento de um *patch* que manipula a informação de um arquivo JPEG, criando erros ou *glitch* no arquivo original e aplicando esses erros ao arquivo a fim de transformá-lo em *glitch art*.

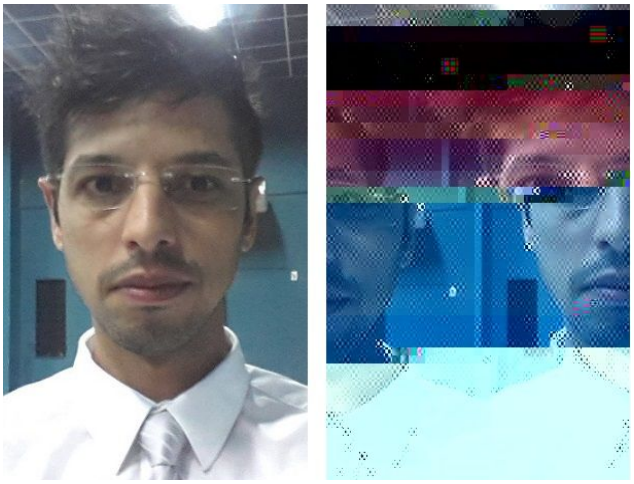
I.3 Objetivos específicos

1) Analisar os métodos de aplicação de *glitch art* entre os principais meios de mídias digitais para se ter um panorama atual dessa estética de arte digital junto ao desenvolvimento da ferramenta.
2) Geração de diferentes versões da ferramenta, sendo estas desenvolvidas para serem a base dos experimentos.
3) Geração de diferentes testes experimentais que são feitos para embasar as análises.
4) Verificar a interação artística para se criar uma obra de *glitch art* por meio desta ferramenta e suas possíveis demais interações.
5) Analisar as perspectivas que essa nova ferramenta facilitadora de *glitch art* oferecerá através de novas interações para os campos do conhecimento artístico e computacional como na música, no processamento de imagens videográficas e em imagens estáticas

II. FUNDAMENTAÇÃO TEÓRICA

Glitch é definido como um erro dentro de um sistema elétrico ou eletrônico, que impede algo de funcionar como deveria. O *glitch* pode ocorrer em qualquer tipo de arquivo, seja ele de áudio, vídeo ou imagem. Durante o estudo, transitamos entre essas formas de mídias digitalizadas a fim de explorar funções para a ferramenta desenvolvida. Começaremos com o exemplo da figura abaixo, figura 1, na qual vemos lado a lado a imagem original e a que teve seus dados corrompidos gerando um erro, ou seja, um *glitch*:

Figura 1 - Exemplo de *glitch* em imagem



Fonte: Arquivo pessoal (2016)

A *glitch art*, por sua própria natureza interdisciplinar, visa a criação de obras artísticas explorando a corruptibilidade da mídia digitalizada. Nesta dissertação, o tipo de *glitch art* explorada como ponto de referência para desenvolver a ferramenta final é a corruptibilidade de dados de um arquivo JPEG, para se obter um aspecto estético favorável ao crivo do artista.

Segundo McCormack (2010, p.15)[3] "*Glitch art* é a arte que modifica tecnologia e causa, ou o *hardware* ou o *software*, a crepitar, falhar, acionar de forma falha, ou se comportar de forma errônea. Definir de forma mais específica pode ser contraproducente". Para criar *glitch art* é necessário conhecimento do âmbito computacional e artístico, segundo Menkman (2011, p.11) "O estudo de

[3]Tradução nossa.

glitch art questiona e propõe o raciocínio social e técnico sobre o nosso entendimento de funções através de erros; questiona e estimula o debate sobre a constante busca da eliminação total do ruído na comunicação".

Glitch art tem um histórico extenso, levando em consideração as suas origens analógicas. Seguem alguns exemplos elencados por McCormack:

> Nos seus primórdios podemos comparar Len Lye, A Colour Box (1937), um filme onde a película foi arranhada e pintada a mão; com o advento de novas tecnologias, como a televisão, surge a obra de Nam June Paik, intitulada Magnet TV (1965); posteriormente, Cory Arcangels TH42PV60EH lança sua Plasma Screen Burn (2007) e Joan Heemskerk e Dirk Paesmans sua Webcrash2800 \%SCR2 (2009), que utiliza uma TV LCD, assim é possível perceber que a glitch art acompanha as etapas das mudanças tecnológicas da televisão. Contudo, a obra mais famosa envolvendo computadores é a de Joan Heemskerk e Dirk Paesmans, Untitled Game (1996-2001), que consiste em 14 variações do jogo Quake 1, sendo todas releituras quase ilegíveis de formas abstratas do conteúdo original. (MCCORMACK, 2010 p.15)[4]

Os exemplos acima servem para compor um mostruário das vertentes que a *glitch art* desenvolveu, como também um panorama da sua história. Dados os avanços tecnológicos, o desenvolvimento da *glitch art* remete à proposta da dissertação. A criação de uma ferramenta facilitadora, que produza *glitch art* em uma imagem, não é inédita. Um software chamado MONGLOT em 2011, desenvolvido por Menkman e Johan Larsby, junta duas imagens que resultam em outra, com aspecto de *glitch art*, e em sua elaboração, foi conceitualizado para ser usado por pessoas sem habilidades e conhecimento sobre corrupção de dados. Com essa base, foi construído o *patch*[5] de *Pure Data* que simplifica o processo de criação de *glitch*, para facilitar a criação de *glitch art*.

Para o desenvolvimento do trabalho que dá suporte a esta dissertação, preservamos de forma purista a geração de um *glitch*. Segundo McCormack, "Puristas farão uma distinção entre arte que usa maus funcionamentos reais, e artes que imitam o mau funcionamento" (MCCORMACK, 2010, p.15), ou seja, o programa altera os dados do arquivo do formato JPEG sendo processado por ele, e não aplica simplesmente um filtro a uma imagem. Seguimos um dos exemplos de Menkman, no qual ela

[4]Tradução nossa.

[5]*Patch* ou no plural *patches*, é o nome dado a programas criados utilizando *Pure Data*.

demonstra uma imagem que trabalhou como uma das possíveis formas de glitch art. “Uma imagem jpeg 2000[6], que foi *databent* através da introdução de erros randômicos de informação nos dados.” (MENKMAN, 2011, p.22). Tomamos esse exemplo como base do que o patch desenvolvido deveria ter como funcionalidade. Utilizamos a seguinte definição dos termos encontrados em discussões a cerca de *glitch art* “*databending, datamoshing* e *circuitbending* têm vindo a existir para nomear a variedade de formas de glitch praticadas, mas todas de fato se referem a práticas similares de quebrar os fluxos dentro de diferentes tecnologias ou plataformas. (MENKMAN, 2011 p.59,)

Com o crescimento dessa estética no mercado, a existência de aplicativos para criação de *glitch art* tem se difundido, porém, como não temos acesso ao código fonte desses aplicativos comerciais, os métodos que eles seguem para gerar *glitch* podem, ou não, estar de acordo com as informações que seguimos baseando nossa bibliografia, ou seja, eles podem ou não ser *true glitch*.

Como exemplos de aplicativos no mercado temos: Glitch! (glitch4ndroid)[7], Glitch Art - Video Effects Edit Joseph Gordan[8], HYPERSPEKTIV[9], Glitché[10], Glitch Wizard - Distort Photos to Trippy GIFs Allan[11] Lavell, Glitchr[12]. Essas ferramentas geram imagens com características de *glitch,* porém, para a pesquisa, interessa o método utilizado funcionalmente para gerar essa estética. O motivo é que buscamos gerar o erro verdadeiro, como sugerido na bibliografia, pois ele estimula novas facetas de criação artística. É possível que os programas utilizem manipulação de ASCII, código binário ou sejam apenas filtros aplicados a uma imagem. Como não temos acesso aos código dos programas e não sabemos seu funcionamento interno, a relevância das funcionalidades oferecidas é questionável. Se forem apenas filtros, esses aplicativos nada mais são que editores de imagens copiando o estilo visual da *glitch art*, porém, sem acesso ao seu código não podemos afirmar tais perspectivas.

[6]JPEG 2000, é uma versão mais recente do formato JPEG.

[7]**Disponível em https://play.google.com/store/apps/details?id=com.lucagrillo.ImageGlitcher&hl=pt_BRhttps://play.google.com/store/apps/details?id=com.lucagrillo.Ima**geGlitcher&hl=pt_BR. **Acessado em: 26 fev. 2018.**

[8]**Disponível em** https://itunes.apple.com/br/app/glitch-art-video-effects-edit/id1024492593?mt=8 **Acessado em: 26 fev. 2018**

[9]**Disponível em** https://itunes.apple.com/us/app/hyperspektiv/id1058051662?mt=8 .**Acessado em: 26 fev. 2018**

[10]**Disponível em** http://glitche.com/ Acessado em: 26 fev. 2018

[11]**Disponível em** https://itunes.apple.com/us/app/glitch-wizard-distort-photos/id904640439?mt=8 **Acessado em: 26 fev. 2018**

[12]**Disponível em** https://play.google.com/store/apps/details?id=com.apporio.glitchr **Acessado em: 26 fev. 2018**

Isso nos impulsiona a desenvolver nossa própria ferramenta de código aberto, na qual está claro o processo de true glitch, e gratuita, pois o processo de criação é tão relevante quanto a obra resultante do processo. "A estética final de uma peça de arte falha não é a única coisa que importa. Na arte de *glitch*, o processo de criação da obra de arte significa tanto quanto o produto final, e determina o que pode ser considerado uma verdadeira arte *glitch* versus arte que parece *glitch*". (ROY, 2014 p.1)[13]

Como o espectro de opções a serem trabalhadas na *glitch art* é deveras amplo, esta pesquisa não abrange todos os softwares que geram *glitch art,* e sim um conjunto representativo de sistemas similares que nos permita estabelecer um panorama adequado para análise da nossa solução, reduzindo o nosso escopo para algo compatível com uma dissertação. Dentro das opções encontradas para se trabalhar *glitch* com *Pure Data* temos o *Pure Data File Killer*[14], do Hellocatfood[15], que faz parte da investigação de nossa pesquisa. Abaixo uma tabela comparativa dos programas citados.

Tabela 1 - Tabela comparativa

Programa	**Sistema Operacional**	**Funcionalidade**
Glitch! (glitch4ndroid)	Android	Múltiplos efeitos de *glitch* com ajuste para intensidade dos efeitos
Glitch Art - Video Effects Edit Joseph Gordan	IOS	Editor de vídeo e imagem que aplica efeitos de *glitch* como filtros e tem possibilidade de adicionar texto às imagens.
HYPERSPEKTIV	IOS	Editor de vídeo e imagem que aplica efeitos de *glitch* como

[13] Tradução nossa.

[14]**Disponível em** https://www.hellocatfood.com/pure-data-file-killer/ **Acessado em: 26 fev. 2018**

[15]Antonio Roberts (hellocatfood) é um artista e curador da New Media baseado em Birmingham, no Reino Unido. Ele usa processos baseados em tecnologia para explorar questões relacionadas a software de código aberto, cultura livre e práticas colaborativas.Seu trabalho visual e performático foi apresentado em galerias e festivais, incluindo databit.me em Arles, França (2012), Glitch Moment / ums na Furtherfield Gallery, Londres (2013), Loud Tate: Code na Tate Britain (2014), glitChicago at Instituto Ucraniano de Arte Moderna em Chicago, EUA (2014), Permissão Obtida no Birmingham Open Media e Universidade de Birmingham (2015-2016), Propriedade Comum em Jerwood Visual Arts, Londres (2016), Green Man Festival, País de Gales (2017) e Barbican, Londres (2018).Fez curadoria de exposições e projetos, incluindo GLI.TC/H Birmingham (2011), as edições de Birmingham de Bring Your Own Beamer (2012, 2013), µChip 3 (2015), Stealth (2015) e Sem violação de direitos autorais pretendida (2017)Ele é Curador do Vivid Projects, Near Now Fellow, e faz parte do Grupo Consultivo de New Art West Midlands. Tradução nossa.

		filtros
Glitché	IOS	Editor de imagem que que aplica efeitos de *glitch* como filtros
Glitch Wizard - Distort Photos to Trippy GIFs Allan	IOS	Editor de vídeos, imagens e GIFS que aplica efeitos de *glitch* como filtros com níveis ajustáveis
Glitchr, Lavell	Android	Aplica efeitos como filtros nas imagens.
Pure Data File Killer hellocatfood	Linux, Windows, Mac OS, Source Code disponível	Altera o binário de um arquivo, com opção de ajuste do ponto inicial da alteração.

Fonte: arquivo pessoal.

Pela própria natureza fluida dos dados de mídias digitalizadas utilizamos algumas formas de interação diferenciadas como músicas e projeções a fim de entender a grandeza e definir o foco final do programa desenvolvido. Esse enorme segmento da cibercultura que é a *glitch art* carrega elementos que André Lemos define abaixo:

> A cibercultura parece jogar com elementos da sociedade do espetáculo, colocando informações e produzindo ruídos, re-apropriando e simulando o mundo. A arte eletrônica é indiferente a objetos originais, ela busca a circulação de informações, o híbrido, a comunicação e interação em tempo real, a tradução do mundo em bits, manipuláveis e postos em circulação na velocidade da luz. (LEMOS, 1997 p.28)

Tanto a cibercultura como a *glitch art* são aspectos do universo maior que é a arte digital. Um dos aspectos da arte digital que abordamos nesta dissertação vem de Johnson, que em seu livro, Cultura de Interface, se refere a arte digital como a mistura elementos de nossa cultura analógica e a tecnologia de seu período, corroborando com Couchot que define em seu livro A Tecnologia na Arte, que uma imagem, texto ou som digitalizado é, no meio digital, informação, podendo fluidicamente ou sinestesicamente serem convertidos de um para outro formato de mídia. Essa fluidez é um dos componentes também apontados por Manovich em seu livro *Language of the New Media* como um

dos princípios das novas mídias, ou seja, característica da arte digital. Para Julio Plaza se trata da abertura de terceiro grau discutida em seu livro, Arte e Interatividade: autor-obra-recepção, interações da arte e da comunicação que permitem a hibridização multimídia, no qual discute também o fim do autor individual de uma obra que é o caso dos resultados apresentados nos experimentos artísticos.

Chamamos a atenção também para o trabalho de Stephen Wilson, que, em seu livro *Information Arts,* faz uma análise de artistas contemporâneos que trabalham com ciência e tecnologia, ressaltando as dificuldades da catalogação desses trabalhos e obras exibidas em instalações, ou com equipamentos que remetem a uma era e se tornam escassos com o tempo, enaltecendo a importância da documentação acadêmica desses processos de exibição e criação de arte computacional.

A *glitch art* é um escopo da arte eletrônica que faz parte dessa enorme massa informacional que é a cibercultura. Talvez não tão segmentada e distanciada pela questão do tempo de criação do termo *glitch art,* pois na concepção de Lévy o termo cibercultura traz elementos de criação recorrentes nos guias de Menkman sobre a criação de *glitch art:*

> Aqueles que se contentam em perambular irão talvez conceber sistemas ou esculpir dados mais à frente. Nada na evolução técnica garante essa reciprocidade; não é mais do que uma possibilidade favorável aberta por novos dispositivos de comunicação. Cabe aos atores sociais, aos ativistas culturais aproveitá-la, a fim de não reproduzir no ciberespaço a mortal dissimetria do sistema das mídias de massa" (LÉVY, 1999 p.150)

Sendo assim Lévy traz o ativismo cultural que acontece no meio do ciberespaço como uma força motriz para mudança e inovação, um tema recorrente em *glitch art* e cibercultura.

II.1 *Pure Data*

Pure data é uma ferramenta de trabalho bastante ampla e a amplitude de funções que o ambiente de desenvolvimento engloba lhe dá a complexidade necessária para a criação de diversas ferramentas artísticas. Porém, essa mesma vantagem gera uma curva de aprendizado elevada. Apesar de ser uma linguagem de programação visual, o número de opções para se trabalhar acaba dificultando o desenvolvimento de ferramentas. Para a criação de ferramentas mais complexas, é necessário dedicar tempo para a compreensão das diversas interações possibilitadas pelo *Pure Data*, tornando

inviável o desenvolvimento de algo complexo logo no primeiro contato com a linguagem, porém, pode-se dizer que este problema é comum no aprendizado de qualquer linguagem.

A distinção do *Pure Data* vem na forma de sua apresentação: a informação sendo apresentada de forma visual e não textual aproxima pessoas que têm maior facilidade em manipular informação desta forma. Em uma descrição mais aprofundada:

> O Pure Data é um ambiente de programação visual de código aberto que é executado em qualquer coisa, desde computadores pessoais até dispositivos embutidos (ou seja, Raspberry Pi) e smartphones (via libpd, DroidParty (Android) e PdParty (iOS). É um dos principais ramos da família de linguagens de programação patcher conhecidas como Max (Max / FTS, ISPW Max, Max / MSP, etc.), originalmente desenvolvido por Miller Puckette no IRCAM. O Pd permite que músicos, artistas visuais, artistas, pesquisadores e desenvolvedores criem software graficamente sem escrever linhas de código. Pd pode ser usado para processar e gerar som, vídeo, gráficos 2D / 3D e sensores de interface, dispositivos de entrada e MIDI. A Pd pode facilmente trabalhar em redes locais e remotas para integrar tecnologia wearable, sistemas de motor, equipamentos de iluminação e outros equipamentos. É adequado para aprender métodos básicos de processamento multimídia e programação visual, bem como para a realização de sistemas complexos para projetos de grande escala.
> As funções algorítmicas são representadas em Pd por caixas visuais denominadas objetos colocados dentro de uma janela de correção chamada tela. O fluxo de dados entre objetos é alcançado através de conexões visuais chamadas de cordas de patch. Cada objeto executa uma tarefa específica, que pode variar em complexidade de operações matemáticas de muito baixo nível para funções complicadas de áudio ou vídeo, como reverberação, transformações FFT ou decodificação de vídeo. Os objetos incluem objetos de base Pd vanilla, objetos externos ou externos (objetos Pd compilados a partir de C ou C ++) e abstrações (patches Pd carregados como objetos).
>
> (PURE DATA, 2018)[16]

O processo de aprendizado da utilização do *Pure Data* foi facilitado pela ampla documentação, o apoio do fórum e de vídeos tutoriais encontrados no YouTube[17], *patches* previamente desenvolvidos e disponibilizados, além de uma gama de recursos adicionais conhecidos como extensões[18]. Um problema encontrado é a variedade em relação a organização visual dentro do *patch*, pois ela muda de acordo com cada criador. Isso dificulta a leitura, pois nem todo *patch* está disposto de uma maneira que o fluxo de informação siga uma ordem parecida com a de uma leitura textual. Neste sentido, seria

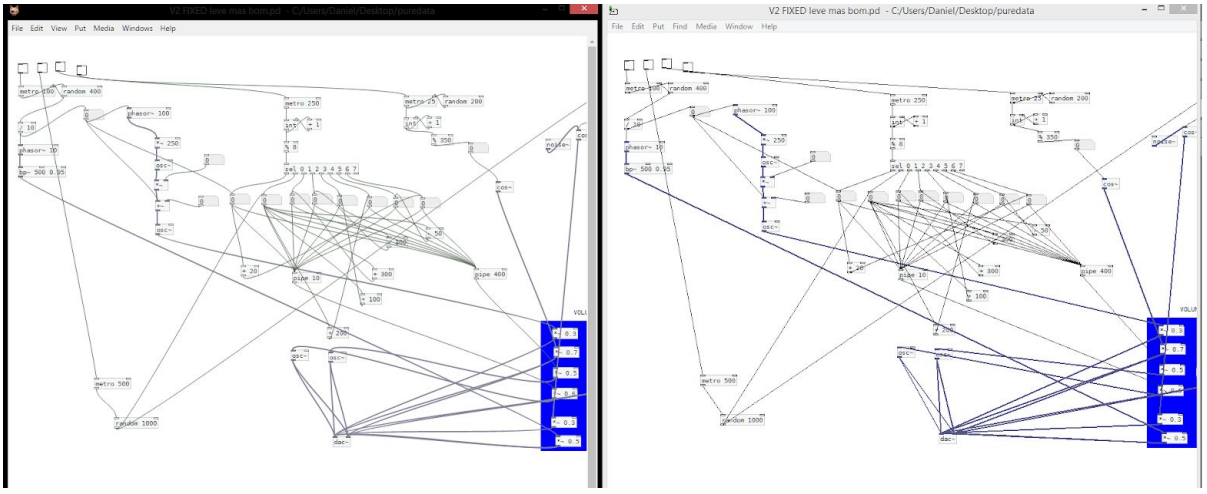
[16]Tradução nossa.
[17]Plataforma de vídeos disponibilizada pela Google LLC.
[18]Extensão ,tradução nossa, um extensão acrescida ao *Pure Data* que lhe atribui novas funções.

importante criar uma função que possa destacar o fluxo de informação, etapa por etapa, para facilitar a leitura de patches maiores, ou para guiar novatos no *Pure Data.*

Apesar da gama de funções já existentes no *Pure Data* em sua forma original, conhecido como *Pure Data Vanilla*[19] optamos por trabalhar com uma versão popular, o *Pure Data Extended*[20] que, apesar de ter seu desenvolvimento parado, é a versão do *Pure Data* com maior número de funcionalidades. O *Pure Data Extended* versão 0.43.4 é uma versão que possui o maior número dessas modificações acrescidas ao *Pure Data Vanilla*. A outra versão que optamos por trabalhar é o *Purr data*[21] Pd-l2ork version 2.1.2 (20170322-rev.6f652fe).

Apesar de ter menos funcionalidades que o *Pure Data Extended,* o *Purr Data* oferece uma boa variedade de funções e atualizações, pois continua em desenvolvimento, além de contar com uma interface gráfica melhorada, se comparada com o *Pure Data Extended*. Utilizando essas duas versões partimos para o aprendizado de uso e o desenvolvimento de nossas primeiras ferramentas.

Figura 2 - *Pure Data* interfaces diferentes.



Fonte: Arquivo pessoal (2018)

[19]In: < https://puredata.info/downloads >. Miller S. Puckette's "vanilla" distribuição do *Pure Data*. *Vanilla* sendo baunilha comumente utilizado na computação para se referir a versão de algo original ou sem modificações.

[20]In: < https://puredata.info/downloads/pd-extended >. Este sendo um software abandonado sem atualizações ou suporte oficial, apenas mantido pela comunidade. Versão atual: Pd-extended 0.43.4 lançada em 25/01/2013 testada com *Pure Data vanilla* 0.43 com uma interface reescritas, e suporte a UTF-8, acesso a diversas traduções da interface, e plugins para o edição e customização, além de muitas novas funções. Tradução nossa.

[21]In: < https://puredata.info/downloads/purr-data >. Jonathan Wilkes' nova cross-plataforma baseado em Java-script com a versão de Ico Bukvic's Pd-l2ork. Tradução nossa.

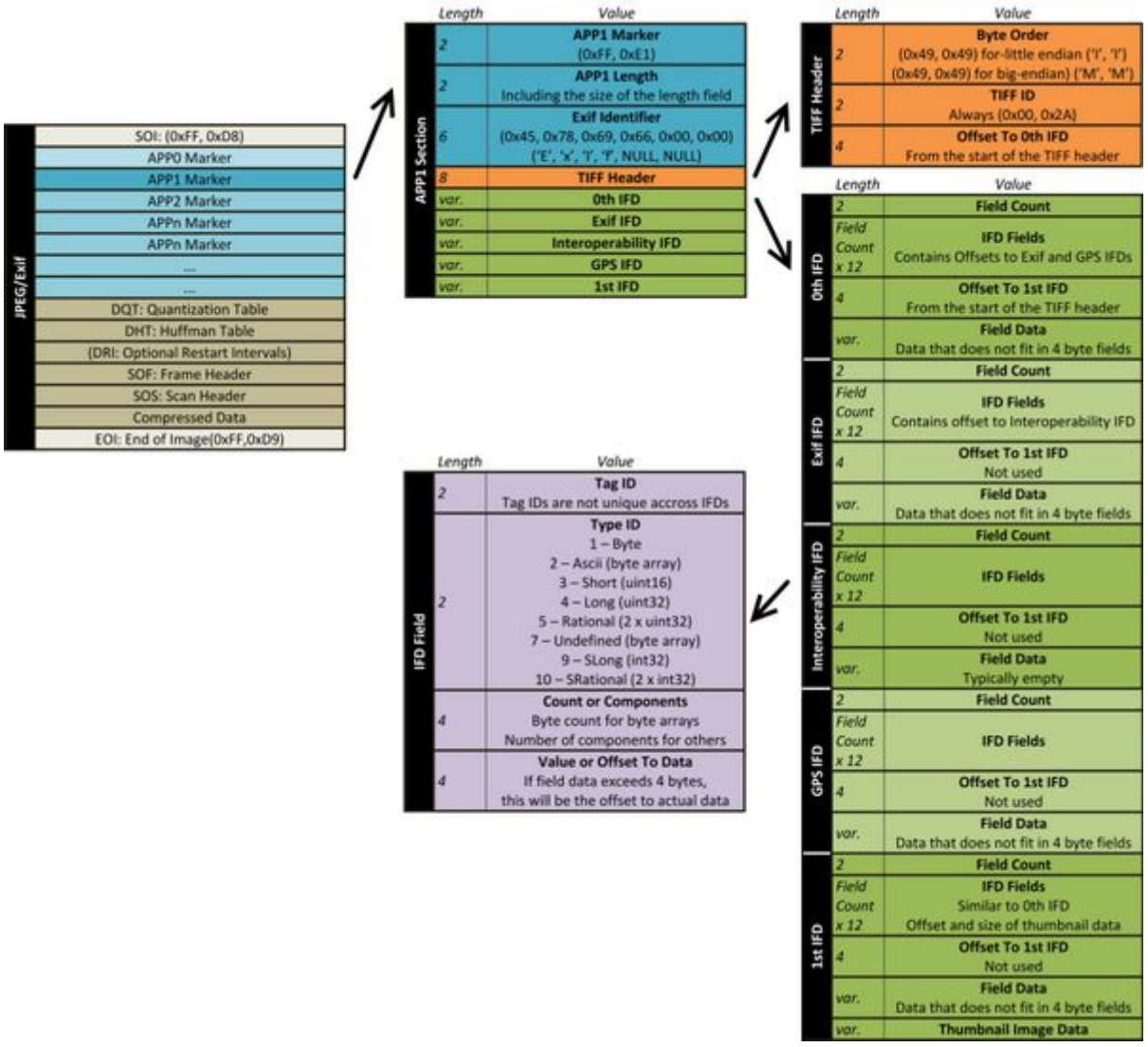
Uma característica relevante em relação ao uso de *Pure Data* que é a sua ativa e receptiva comunidade de usuários, que se comunica avidamente com desenvolvedores tanto veteranos na linguagem quanto novatos buscando informações. Essa injeção inicial acerca da linguagem e a generosidade de informações através de *patches* antigos e conversas no fórum oficial[22] é comum em ambientes que utilizam código aberto, e uma experiência que enriqueceu a pesquisa. Porém, não sem dificuldades iniciais. A linguagem básica usada pelos programadores veteranos, por exemplo, quando tentavam ajudar em algo requisitado por nós, era aquém dos nossos conhecimentos, sendo necessário um estudo prévio do linguajar técnico para, efetivamente, fazer uso favorável das sugestões.

II.2 JPGE estrutura

Continuamos com a busca por um editor ASCII que não produzisse distorção nos arquivos apenas por abri-los, nos dando mais opções de intervir nos mesmos. A tradução sem alteração do arquivo também era importante para a preservação do *header,* pois, se algo se perdesse na recodificação do *header,* o arquivo não funcionaria. Para compreender a metodologia aplicada é necessária uma breve explicação do formato do arquivo. Apesar da complexidade dos elementos contidos dentro do formato JPEG, a imagem abaixo demonstra claramente que o *header* está contido no marcador APP1, porém quando se traduz a informação do arquivo para ASCII através do notepad++ não temos uma relação direta que identifique o que cada código do ASCII representa na imagem. Por isso, optamos por trabalhar com a identificação visual de diferentes padrões dentro do código ASCII, podendo assim identificar o *header* do restante das informações ali contidas.

Figura 3 - Estrutura de arquivo JPEG

[22] In: < https://forum.pdpatchrepo.info/ >



Fonte: https://www.codeproject.com/Articles/43665/ExifLibrary-for-NET (2009)

A macro estrutura do formato JPEG consiste do SOI, que possui apenas dois bytes expressos em hexadecimal (0xFF, 0xD8)[23] e tem a função de identificar o arquivo como JPEG. Seguimos com seções de marcadores de aplicativos ou APPn:

> Para identificar as seções do APPn, iniciamos a partir do SOI e lemos os próximos bytes. Embora o conteúdo das seções APPn varie, os dois primeiros bytes são sempre o marcador APPn. Para a seção APP0, o marcador é (0xFF, 0xE0), para a seção APP1 (0xFF, 0xE1) e assim por diante. Os bytes do marcador são seguidos por dois bytes

[23]Apresentado aqui em hexadecimal.

> para o tamanho da seção (excluindo o marcador APPn, incluindo os bytes de tamanho). O campo de comprimento é seguido por dados de aplicativos de tamanho variável. (OZCITAK, ExifLibrary for .NET, 2009)[24]

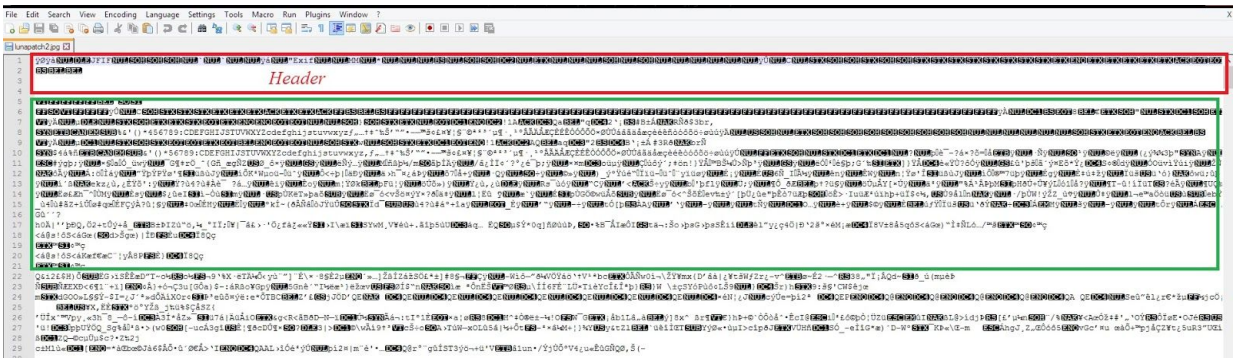
Focaremos nossa atenção na seção APP1, pois é onde os metadados[25] são armazenados. Nesta seção temos as informações de marcador e tamanho, e o tão importante marcador Exif de 6 bytes[26], que identifica o arquivo como uma imagem JPEG / Exif. Depois disso, há o cabeçalho TIFF que contém informações sobre a ordem dos bytes. Dentro da seção APP1 temos o TIFF *header* que pode ser explicado como:

> O cabeçalho TIFF contém dois valores importantes. Os dois primeiros bytes do cabeçalho TIFF informam se as seguintes seções IFD estão na ordem de bytes little endian ou big-endian. Como os arquivos de imagem são normalmente transferidos entre dispositivos, e esses dispositivos podem ter ordens de bytes diferentes, é crucial interpretar corretamente a ordem de bytes fornecida no cabeçalho TIFF. O segundo valor importante é a localização do 0º IFD. Esse local é fornecido como um deslocamento desde o início do cabeçalho TIFF. (OZCITAK, ExifLibrary for .NET, 2009)[27]

Estes são os pontos principais para a proteção do *header.* Esse entendimento básico da estrutura do arquivo nos permite localizar o *header* por número de linhas de código. Abaixo, temos um exemplo do *header* separado do corpo do arquivo através de identificação visual dentro do código e não da compreensão do código em si. Identificamos que o *header* está nas primeiras linhas do arquivo e apresenta um padrão diferente do restante do código ASCII, por isso não alteramos essas linhas. A partir dessa identificação visual, conseguimos preservar o funcionamento dos arquivos editados com mais sucesso.

Figura 4 - Exemplo de *glitch* com notepad++

[24]Tradução nossa.
[25]Metadados tem a função de facilitar o entendimento e utilização dos dados de um arquivo.
[26]Em Hexadecimal (0x45, 0x78, 0x69, 0x66, 0x00, 0x00) ('Exif \ 0 \ 0')
[27]Tradução nossa.



Fonte: Arquivo pessoal (2018)

Trabalhamos com a hipótese de que os dois programas que utilizamos para traduzir a informação para ASCII não mantinham o *header* intacto quando a imagem era importada, porém, não pudemos comprovar a teoria. Achamos a seguinte informação sobre o tipo de codificação usado para se abrir os arquivos de imagem em formato de texto:

> Dependendo da codificação usada para abrir o arquivo, você poderá ver um comportamento diferente. O meu bloco de notas do Windows 7 permite abrir um arquivo em ANSI[28], UTF-8[29], Unicode ou Unicode big endian[30]. Eu testei esse problema com uma pequena imagem JPEG de 2x2 pixels criada com o gimp e abrindo e salvando o arquivo de imagem com a codificação ANSI. Abrindo o original e a imagem salva com um editor hexadecimal, vejo que todas as seqüências 00 (dois dígitos hexadecimais, caractere de controle NUL) foram convertidas em 20 (caractere de espaço). Substituindo de volta no editor hexadecimal todos os 20 por 00 restaura o formato da imagem. Eu pesquisei um pouco e não encontrei nenhuma referência que explique por que isso acontece. Apenas uma referência a uma postagem que avisa sobre isso (link de cache do google, a página não está disponível). Se você salvar / abrir o arquivo como UTF-8, parece que ele ainda converte caracteres NUL em

[28]A codificação ANSI permite apenas caracteres suportados na página de códigos atual, o que limita o uso internacional. (tradução nossa) Disponivel em https://msdn.microsoft.com/pt-br/library/w11571b4.aspx acessado em 23 mar 2018.

[29]A codificação UTF8 pode ser usada para caracteres internacionais. No entanto, não use codificação UTF8 com o Visual SourceSafe 6.0 ou anterior porque são feitas alterações nos arquivos durante o check-in, check-out, verificação de diferença e mesclagem que podem manipular ou corromper incorretamente o conteúdo dos arquivos. (tradução nossa) Disponivel em https://msdn.microsoft.com/pt-br/library/w11571b4.aspx acessado em 23 mar 2018.

[30]A codificação Unicode pode ser usada para caracteres internacionais. No entanto, se você estiver usando o Visual SourceSafe 6.0 ou anterior, não será possível usar a funcionalidade de check-out compartilhado, verificação de diferença ou mesclagem, pois os arquivos com codificação Unicode são tratados como arquivos binários. Tradução nossa. Disponivel em https://msdn.microsoft.com/pt-br/library/w11571b4.aspx acessado em 23 mar 2018.

> espaços, mas também aumenta o tamanho do arquivo resultante devido a conversões de caracteres de byte único para seqüências de bytes múltiplos UTF-8. Se você salvar / abrir o arquivo como Unicode, parece que ele ainda converte caracteres NUL em espaços, mas também adiciona um byte ao início do arquivo, a BOM. (MANGPER[31], https://goo.gl/HBgXN7, 2014)[32]

Com essas informaçõe seguimos a pesquisa utilizando as informações da estrutura do arquivo.

II.3 Poética e estética

A *glitch art*, sendo trabalhada como um aspecto da arte computacional, não é, em sua essência, novidade em seu aspecto estético. Um gênero semelhante em sua poética, mas em mídias diferentes, já existia. Assim, sua origem não computacional pode ser relacionada com o grotesco na arte, que tem seus primeiros registros de estudo vindos do século XVII. O grotesco pode ser pensado como uma evolução, um processo de transformação constante, lidando com dualidades como o nascimento e a morte. "Seu segundo traço indispensável, que decorre do primeiro é a ambivalência: os dois pólos da mudança – o antigo e o novo, o que morre e o que nasce, o princípio e o fim da metamorfose – são expressados ou esboçados em uma ou outra forma." (BAKHTIN, 1987, p. 21-22). Todas essas possibilidades integram a imagem grotesca.

Essa dualidade do grotesco e da *glitch art* pode ser interpretada como a funcionalidade e a não funcionalidade. Assim como no grotesco, também na *glitch art* se defende a evolução como propósito poético, sendo que a relação desses âmbitos estéticos se estreita ainda mais ao se interpretar as principais características do grotesco, ou seja: constante dualidade e alternância de sentidos como, por exemplo, o fantástico e o monstruoso ou a harmonia e o caos. Tais semelhanças fazem ver no grotesco um possível gênese para a *glitch art,* ou ainda, a *glitch art* como forma de hibridização – algo típico do grotesco – no uso de mídias digitais, com a geração de ruídos digitais através de interferências nos arquivos, como processo de composição. "Valorizado pelos românticos (para os quais a arte deve representar tanto o belo como o feio e o deformado), o grotesco se transforma, posteriormente, em categoria estética e literária para fazer referência a um tipo de descrição ou de tratamento deformador da realidade, ..." (ENCICLOPÉDIA Itaú Cultural de Arte e Cultura Brasileiras, 2019 p.1)

[31]Pseudônimo do autor do texto.
[32]Tradução nossa.

O erro e o acaso na construção desse experimento nos fizeram lidar com duas formas de influência diferenciadas. Primeiramente temos o erro como uma interferência interna, erros atribuídos a codificações que, muitas vezes, eram feitas ao acaso. O acaso é a nossa segunda influência, tratando-se de uma interferência externa em nossa composição, que se dá pela falta da compreensão absoluta do que cada byte[33] representa dentro da macro escala de um arquivo, gerando informações aleatórias geralmente tidas como erros. Trabalharemos o acaso (ou chance) baseado na definição de George Brecht:

> A chance nas artes fornece um meio para escapar dos preconceitos enraizados em nossa personalidade por nossa cultura e história passada pessoal, isto é, é um meio de alcançar maior generalidade. O resultado é um método de abordagem com ampla aplicação. Os métodos de chance e aleatoriedade podem ser aplicados na seleção e disposição pelo compositor, (...) (BRECHT, 1966, p. 23)[34]

Ambas influências na composição se retroalimentam e distorcem a nossa obra final. De maneira semelhante à relação que há entre a *glitch art* e o grotesco, Briz (2011) sugere uma relação de *glitch art* com o dadaísmo: "A revolução digital/informação é (poderia levar a) uma mudança de paradigma própria - a arte Glitch responde a isso ao questionar a estabilidade (ou esforços para uma estabilidade idealizada) de tal mudança. Desta forma, a arte da falha é como o dadaísmo." (BRIZ, 2011, p.55)[35]

Esse retorno da *glitch art* para o ruído em suas diversas formas, que no geral se busca eliminar da comunicação, e suas relações nas obras de *glitch* trazem um labor para inseri-las, "... ao passo que a falha incorpora o retorno inadequado de trabalhos analógicos laboriosos e confusos no contexto da moda digital, pixel polida, drag-and-drop perfeccionista." (MANON E TEMKIN, 2011 p.11)[36]. Esses resultados de obras digitais usando métodos computacionais para alterações, gerando esteticamente resultados que se assemelham com o grotesco, possuem toda uma gama de artistas que podem ser tomados como referência, como os precursores deste movimento:

[33]Uma unidade de medida da quantidade de informação que um computador consegue guardar. CAMBRIDGE Dictionary: Definition of "Byte" - English Dictionary. Definition of "Byte" - English Dictionary. 2016. Disponível em: <http://dictionary.cambridge.org/us/dictionary/english/byte>. Acesso em:16 out. 2017. Tradução nossa.

[34]Tradução nossa.

[35]Tradução nossa.

[36]Tradução nossa.

> Precursores importantes para a falha incluem Nam Jan Paik, John Cage, Annie Albers (que pintou padrões de busca fractal antes de serem descobertos), artistas associados com Novas Tendências, como Hiroshi Kawano (cujas pinturas de Markov em 1964 são intensamente problemáticas) e outras formas de arte generativa (quando era mais entrincheirado em pixilation blocky), bem como Max Headroom: o rosto amigável no modo escuro de cyberpunk dos anos 80, Metal Machine Music de Lou Reed (para não mencionar a longa história da música de ruído), EinstürzendeNeubauten, Lis Rhodes (cuja Light Music , construído inteiramente a partir de elementos analógicos, é quase indistinguível de algum trabalho de vídeo de falha) e net.art
> pioneiros que abraçaram o caos, como JODI e NetochkaNezvanova. Uma lista mais extensa de artistas de proto-glitch pode incluir figuras como François Rouan, Andy Warhol, Ianis Xenakis, Ken Jacobs, Gerhard Richter e Jamie Reid. (MANON e TEMKIN, 2011 p.3)[37]

Em sua árvore genealógica artistas que buscam o ruído, a distorção da imagem, o uso do erro e randomicidade no processo de criação, são influências aparentes no trabalho de *glitch art*, que ressurgem na vanguarda do campo virtual.

II.4 Música *glitch*

Buscando constituir ferramentas que permitissem compor a partir da inter-relação da música com a computação, encontramos o *chiptune*, estilo musical eletrônico que necessariamente utiliza áudio gravado a 8 bits[38], valendo-se geralmente de *hardware* da década de 1980 e que, através da manipulação direta dos chips de áudio, compõe uma nova estética musical. Esse estilo se repopularizou em 2007, com o álbum da banda Kraftwerk, intitulado "Kraftwerk: 8-Bit Operators: The Music of Kraftwerk".

Utilizamos também o *bytebeat,* uma forma de emulação por *software* do *chiptune,* a fim de se obter o som característico do estilo através de sintaxes matemáticas. Trataremos aqui do processo de criação do *glitch* usado na música e da influência dessas técnicas no processo de criação, como as limitações dessas ferramentas, suas peculiaridades, como o acaso e o erro influenciaram em sua composição.

O processo começou com o estudo do Chiptune (ou chipmusic), um estilo de música de 8-bit: gênero de música eletrônica que manipula videogame, máquinas de arcade[39] e antigos computadores

[37]Tradução nossa.

[38]O Bit é uma simplificação do termo dígito binário, sendo este a menor unidade utilizada na computação.

[39]Máquina de fliperama "Eletrôn Máquina montada numa espécie de caixa, geralmente em vídeo e dispositivos de controle, dotada de um cartucho ou microprocessador que possibilita um tipo só de videojogo ou telejogo em suas variações ou graus de dificuldade." Disponivel em: < https://www.dicio.com.br/fliperama/ >. Acessado em 14 dez 2017

para se criar música a partir desses circuitos. A prática de manipulação de maquinário visando a composição é anterior a este tipo de música eletrônica:

> Alguns compositores vêm a tempos considerando o gravador de fita um instrumento musical capaz de mais que uma transcrição de alta fidelidade que os fabricantes limitam como sua função. Agora há híbridos, filhos de instrumentos eletrônicos e acústicos, com áudio que imita os clones digitais dos gravadores de fita. (OSWALD,1985, p.23)[40]

Esse estilo de manipulação de maquinário como brinquedos que produzem som, ou a alteração de teclados para mudar o som que eles originalmente faziam e utilizar esses novos sons em suas composições, é também conhecido como *circuit bending*[41].

Como não possuíamos o *hardware* necessário para fazer o trabalho com *chiptune,* como por exemplo um *game boy* ou um Commodore 64, tivemos que pensar em outras ferramentas. Munidos do computador, buscamos outras soluções. Alguns sintetizadores modernos oferecem opções simplificadas para a utilização, tanto do *chiptune* em forma de emulação, quanto do *glitch art*. Eles, entretanto, não são ferramentas universais e nos preocupava o afastamento de práticas que se originaram da *glitch art* e do *chiptune.* Assim utilizamos um facilitador para chegar a elas. Baseado na palestra ministrada por Erlehmann, no evento SIGINT13 para o Chaos Communication Congress em 23 de julho de 2013:

> É possível a síntese e composição de áudio sem instrumentos, notas ou faixas: pode-se criar música interpretando a saída de programas curtos escritos na linguagem de programação C como uma descrição de uma forma de onda usando modulação de código de pulso (por canalização para / dev / dsp). Os efeitos podem ser criados explorando artefatos arquitetônicos como aritmética inteira modular. (ERLEHMANN, 2013)[42]

Ele oferecia uma solução para o problema com um compilador C, que tem a capacidade de gerar sons 8 bits. Vejamos abaixo:

Tabela 1. Código fonte em C utilizado para geração de sons 8 bits.

[40]Tradução nossa.
[41]Dobrar circuito, tradução nossa.
[42]Disponível em https://www.youtube.com/watch?v=vCEUyx-SxPw Acesso em 22 abril 2016
Tradução nossa.

```
main(t) {
for(t=0;;t++)
putchar(t*(((t>>12)|(t>>8))&(63&(t>>4))));}
```

fonte: (ERLEHMANN, 2013)

III. METODOLOGIA

A metodologia empregada foi uma combinação de pesquisa de campo, pesquisa bibliográfica e experimentação. Os dados da pesquisa de campo foram coletados com o acesso ao fórum do *Pure Data*, também por meio da análise de *patches* feitos anteriormente e de ferramentas já existentes, juntamente com novas informações relevantes a *glitch art* que encontramos através de nossas experimentações.

A linguagem de programação elencada para o desenvolvimento dessa ferramenta foi o *Pure Data,* por causa de sua comunidade ativa, pelo fato de não ter custo, ser de código aberto e também de estar disponível em diversas plataformas. Uma de suas característica é que o software criado em *Pure Data,* conhecido como *patch*, só pode ser executado dentro do próprio *Pure Data*, que além de um ambiente de desenvolvimento, é também uma máquina virtual que serve para executar os programas desenvolvidos no mesmo, tornando a visualização dos resultados do processo de desenvolvimento mais rápida.

Com essa ferramenta computacional, o *Pure Data*, elaboramos o primeiro *patch* protótipo, visando alcançar os objetivos traçados, que são: primeiramente eliminar a necessidade de utilização de editores de linhas de códigos ou de softwares que são subvertidos de sua função original, métodos que são comumente utilizados na manipulação dos dados do arquivo a ser utilizado na criação de *glitch art.* A eliminação desse passo é positiva pois ele exige do usuário uma carga de conhecimento técnico elevada. Posteriormente, evitar destruir partes essenciais dos dados contidos no arquivo, para que ele possa, mesmo tendo características estéticas de *glitch art*, funcionar em vários *softwares* ou ambientes de trabalho. Essas metas foram testadas através de diversas experimentações e também com o desenvolvimento do *patch,* até chegarmos ao resultado no qual o processo de criação de *glitch art* foi simplificado.

De um ponto de vista metodológico, a interdisciplinaridade e o resultado artístico da parte experimental nos colocou um desafio interessante. A princípio, nos baseamos na metodologia da *Design Science Research* (DSR) (BAX, 2014, p.299) na qual a base operacional da pesquisa e o método primário de investigação experimental se baseiam no desenvolvimento de *software*. Com o

auxílio da *design science research,* pudemos criar a base do nosso conhecimento teórico durante o processo experimental e justificar como tais processos são de carácter científico.

> A DSR envolve construir, investigar, validar e avaliar artefatos tais como: construtos, arcabouços, modelos, métodos e instâncias de sistema de informações, a fim de resolver novos problemas práticos. Além disso, o estudo de métodos, comportamentos e melhores práticas relacionadas com a análise do problema e com o processo de desenvolvimento de artefato são abrangidos. (BAX, 2014, p.3888)

Seguindo essa base, cada experimento antecede outro para o processo do desenvolvimento do *software* final. Nos primeiros experimentos não fizemos uso da programação em *Pure Data* a fim de melhor compreender como se fazer artefatos de *glitch art,* seguindo os exemplos de criação de *glitch art* citados em nossa pesquisa bibliográfica. Este conhecimento nos ajudou no desenvolvimento das etapas de automatização do processo que investigamos. Assim objetivamos desenvolver um *patch* que não apenas seja funcional, mas esteja claramente alinhado com nossa expectativa do que ele deve gerar.

Seguimos o ciclo comum da *design science research,* que consiste tradicionalmente de: (1) Investigação do problema; (2) Projeto de soluções; (3) Validação da solução; (4) Implementação da solução e (5) Avaliação da implementação (com pequenas variações).

Na etapa (3), a validação de solução é onde há a distinção em nossa parte experimental, pois os resultados são avaliados pelo crivo do artista pesquisador. O caráter interdisciplinar do resultado do experimento nos traz interesse em gerar o nosso próprio método de avaliação, porém, o ciclo padrão nos serve bem para guiar as questões teóricas. “Este ciclo regulador serve tanto para orientar a parte prática da pesquisa, i.e. resolução de problemas práticos, quanto a parte teórica, quanto trata-se sobretudo de gerar conhecimento respondendo a questões de conhecimento (i.e., questões teóricas)” (BAX p.3891 ,2014)

Seguimos explorando *design science research* com nossa metodologia sendo desenvolvida ao longo de nossa pesquisa interdisciplinar. Devido às características de trabalhar em dois campos como pesquisador, arte e computação, possuímos desafios únicos. Como, por exemplo, ser um programador que deve avaliar o *software* que está sendo desenvolvido sob o olhar de uma curadoria artística, pois o

resultado deve inferir sobre questões poéticas e estéticas e vice versa. Esses questionamentos artísticos retroalimentam o desenvolvimento das funções que como programador devemos modificar ou adicionar ao *software*.

A dualidade do objeto de pesquisa nos obrigou a adaptar métodos de ambas as áreas e trazê-los para os moldes dos experimentos executados ao longo de nossa pesquisa, na qual o conhecimento artístico do audiovisual se mescla com o necessário conhecimento técnico para interferir nas obras, além de ser necessária nossa análise metodológica a fim de validar o conhecimento adquirido com tais experimentos. O aplicativo desenvolvido torna-se parte do processo de criação de uma obra de arte como uma ferramenta e não como a peça final em si.

Para esta dissertação, buscamos um conjunto representativo de sistemas similares que nos permitisse estabelecer um panorama adequado para análise da nossa solução. O funcionamento do aplicativo em desenvolvimento se enquadra perfeitamente no escopo do *design science research*, e a avaliação do seu funcionamento segue a busca da estética *glitch.* Temos uma visão clara do que buscamos, mesmo com a gama de opções da *glitch art,* optamos por trabalhar com uma determinada estética. A busca por essa estética, aliada ao resultados de nosso *patches,* é que traz sucesso ou falha aos nossos experimentos, que fazem parte do processo de criar arte. Acrescido a isso, temos o aprendizado da técnica e a documentação da mesma junto ao processo de criação.

III.1 Sobre a busca bibliográfica

A busca bibliográfica inicial começou abarcando informação de como gerar o que era conhecido como um *glitch* em obras de *glitch art* e em mídias digitalizadas. Logo, buscamos sítios especializados e referências na internet acerca do assunto, a fim de achar informações de como gerar o primeiro *glitch*. Essa pesquisa básica era necessária para definir o que é *glitch art* e perceber suas manifestações atuais e também de onde veio essa estética artística.

Esse estilo de arte computacional era baseado primeiramente na manipulação dos arquivos de forma não tradicional, ou seja, não utilizando programas editores de imagem ou áudio da forma como foram criados a serem usados. Dada essa característica base da corruptibilidade do estilo, se fazia necessário entender a parte mais complexa que seria como fazer essas intervenções em arquivos.

O primeiro passo foi verificar os artigos e livros disponíveis no Google Acadêmico, além de matérias jornalísticas sobre o tema e relatos de artistas vivenciaram a criação de *glitch art.* Buscamos fontes que possuíssem uma descrição do processo e que essa descrição fosse em uma linguagem simples o suficiente para que as etapas do processo fossem compreendidas.

Vale ressaltar que a própria experiência pessoal do pesquisador relacionada a computação era de configuração e montagem de computadores, além de, profissionalmente, a produção de conteúdo multimídia. O conhecimento sobre programação começou junto com a criação desta dissertação, logo, o processo de edição de uma imagem ou arquivo por linha de código parecia inusitado.

A compreensão desses processos, por mais simples que pareça a um profissional da computação, ilustra as dificuldades de um artista para utilizá-los como ferramenta. Passada essa etapa de compreensão do básico, começou a busca por textos mais acadêmicos como dos pesquisadores de *glitch art* e arte digital, Rosa Menkman, Iman Moradi, Tom McCormick, Nick Briz, Scott Contreras, Koterbay and Lukasz Mirocha, Giselle Beiguekman, Suzete Venturelli e José Fernandes. Dos pesquisadores de *Pure Data*, Peter Brinkmann, Danny Holmes, Mark Danks, e o criador e pesquisador do *Pure Data,* Miller Puckette, dos pesquisadores de arte Mikhail Bakhtin e George Brecht, entre outros.

Durante os procedimentos experimentais, o processo de criação, tanto da ferramenta quanto do desenvolvimento do conhecimento acerca de *glitch art,* foi dissertado em etapas diferentes de acordo com o diário da pesquisa.

IV. PROCESSO DE PESQUISA

O processo experimental foi dividido em algumas etapas: o aprendizado de como gerar *glitch*, como utilizar *Pure Data* criando *patches*, o desenvolvimento de *patches* para construção de *glitch art* em *Pure Data* e também a exploração das opções de gerar *glitch* com outras ferramentas e mídias digitais. A idealização de automatizar esse processo de *glitch,* mantendo a estética pretendida, faz parte do desafio. Abaixo os tópicos discorrem sobre essas etapas. Verificamos experimentalmente os principais métodos de criação de *glitch art* levantados na bibliografia para obter as informações necessárias para o desenvolvimento do algoritmo que usamos. Consideramos nossos *patches* nossos artefatos que devem ser analisados como projeto de soluções; e através deles iremos fazer a validação da solução, a implementação da solução e buscar a avaliação da implementação.

IV.1 Primeiro contato experimental

A primeira experimentação com *glitch art* foi uma tentativa de fazer um *glitch* em um arquivo BITMAP[43] e um arquivo JPEG utilizando o método mais simples possível: abrindo uma imagem com o editor de texto notepad[44] do Windows, alterando alguns caracteres do código ASCII[45] exibido e salvando o código como imagem novamente. Após esse processo, verificamos se ao tentar abrir o arquivo ele ainda funcionava de forma normal, e apreciamos os resultados dessa pequena mudança. Através de tentativa e erro, percebemos que os arquivos de formato BITMAP eram os únicos que tinham sobrevivido a esse processo inicial, mesmo que a edição tenha sido feita bem abaixo do *header* [46] do arquivo.

Logo no início desta pesquisa, a necessidade da preservação do *header* de um arquivo, para que o mesmo sobrevivesse a esse processo inicial de *glitch,* se apresentou importante. Mas, mesmo com a preservação do *header* e sofrendo alteração apenas no corpo do arquivo, alguns arquivos apresentavam maior ou menor resiliência ao processo de alteração do ASCII. Às vezes, a alteração do ASCII não

[43]Mapa de bit padrão. In:https://www.fileformat.info/format/bmp/egff.htm
[44]Bloco de notas do sistema operacional windows 8.1.
[45]Código padrão americano para intercâmbio de informação, uma tabela de caracteres.
[46]Cabeçalho de um arquivo, o cabeçalho de um arquivo contém informações essenciais para o funcionamento do mesmo.

apresentava mudança perceptível ou destruía a imagem original por completo, sem deixar material visual para a composição da pesquisa.

Percebemos, ao trabalhar com o arquivo BITMAP nesse processo simplificado, que apenas o ato de abrir um arquivo em um editor de texto e salvá-lo como imagem já o alterava. O processo de transcodificação do código original do arquivo em ASCII, feito pelo notepad, modifica o arquivo levemente, ou o danifica de forma que ele não mais seja aberto por editores de imagens, logo, a transcodificação, ou tradução entre as linguagens, não era perfeita. Assim como em traduções da língua falada, existem perdas.

Outro exemplo dessas pequenas variações criadas pela formatação do editor de texto: durante um experimento substituímos no código ASCII todas as ocorrências dos símbolos E e 3 em uma imagem pelos caracteres “ely”e salvamos o arquivo. Mesmo reconstituindo o código ASCII ao seu formato original, a imagem não voltou à sua forma original, mantendo um leve tom vermelho ao seu redor depois de ser reconstruída.

IV.2 Notepad++

No interesse de aperfeiçoar esse processo de edição de imagens através de linha de código, para que a manipulação do código ASCII tivesse uma transcodificação melhor, buscamos uma ferramenta um pouco mais elaborada. Acabamos por utilizar o notepad ++[47], que é uma alternativa mais comum para se programar, por suas opções voltadas para ambiente de desenvolvimento. Porém, com o notepad++ foram obtidos resultados similares aos obtidos com o notepad. Essa distorção, apesar de se aproximar do *glitch art,* estava sendo feita pela própria máquina em uma falha de codificação, sendo esse o nosso primeiro passo experimental dentro do *glitch art,* porém, não nos dava nenhuma opção de controle ou sofisticação do *glitch*.

O notepad++, devido a sua funcionalidade de aceitar plugins que mudam as suas funções, poderia ser uma ferramenta viável para nossa pesquisa, porém, mesmo explorando tal possibilidade. Optamos por trabalhar com o Frhed[48]. O programa, além de preservar os arquivos sem fazer as

[47]Notepad++ é um editor de texto parecido com o bloco de notas, porém tem as funcionalidades de um editor de código fonte adicionado a ele. Sendo assim possui suporte a várias linguagens de programação.

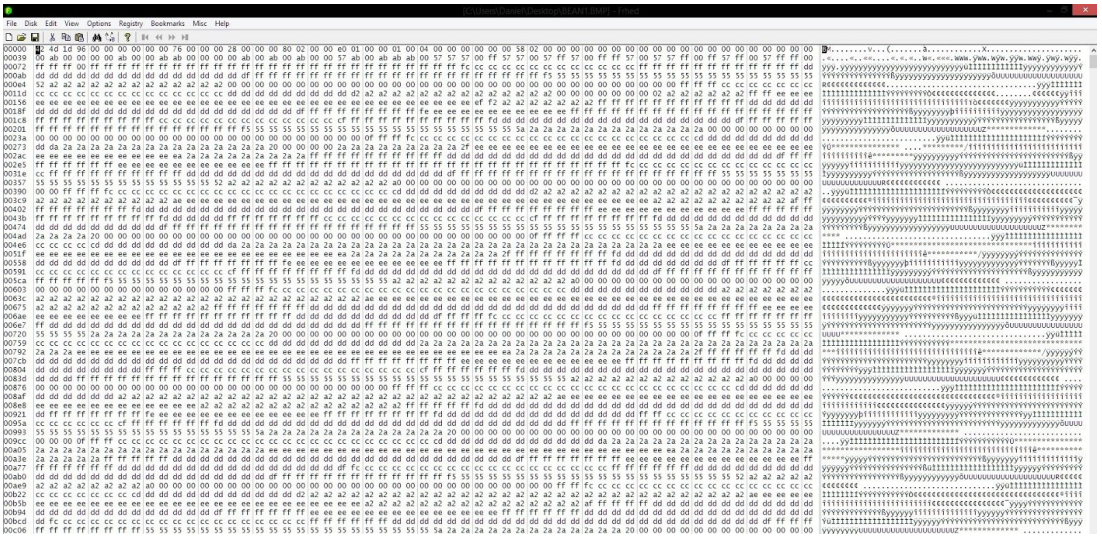
[48]Frhed é um editor de arquivo binário que também trabalha com código ASCII.

alterações vistas durante o uso do notepad e notepad++, abriu a possibilidade de, além de trabalhar ASCII, trabalhar com o hexadecimal[49] do arquivo de imagem.

IV.3 Frhed

O Frhed conseguia manter a transcodificação do arquivo intacta, diferente dos programas anteriores, pois mantinha o *header* preservado. Então essa foi a primeira vez que conseguimos acesso ao código de uma imagem em ASCII e em hexadecimal de forma integral. Mesmo não sendo o intuito original trabalhar com hexadecimal, a opção extra era bem vinda e a interface do programa também facilitava o seu uso. O Frhed é mais uma opção que torna possível trabalhar o processo conhecido como *databending* das imagens sem os erros de tradução. Nesse caso, estamos fazendo o *databending,* que é a manipulação das linhas de código, dentro do arquivo de imagem. Outra opção será vista mais a frente em outros experimentos.

Figura 5 - *Tela de interface do Frhed alterando um arquivo JPEG*



Fonte: Arquivo pessoal (2017)

Para fazermos os experimentos com o programa e tentar criar a *glitch art* seguimos o seguinte procedimento: passamos a editar o código ASCII da imagem em pequenas porções, salvando após

[49]Sistema de numeração onde em vez do 10 que utilizamos como base para o sistema métrico, utiliza-se o 16.

uma pequena alteração e verificando o resultado, voltando a versões antigas caso a imagem fosse destruída pelo excesso de modificações em seu código. Após várias versões da mesma imagem e alterações, compusemos nossa peça *glitch* com o Frhed.

Como dito anteriormente, grandes alterações ao código do arquivo sempre podem destruí-lo ou gerar algo tão distorcido que a peça gerada pelo uso de *glitch* pode não possuir semelhança alguma com a sua origem. Às vezes esse é um resultado que se busca, mas também podemos trabalhar com um meio termo, uma mistura da imagem original com os resultados do *glitch.* Uma das vertentes comuns ao estilo *glitch art* é o artista modificar com *glitch* uma obra original sua ou de outro autor. Ainda que esses primeiros passos alcançassem resultados positivos, buscamos um entendimento primariamente funcional do *glitch art* nessa etapa, buscando sempre ter arquivos alterados e ainda funcionais, experimentando a fim de compreender o processo de geração de *glitch*. Com essa experimentação inicial, temos uma base para definir os caminhos estéticos possíveis e a compreensão do que era controlável nessas alterações e o que não era. Essa etapa ajudou a sedimentar questões funcionais mais básicas.

Durante esse período, pudemos criar uma percepção de que a linha de código que sofre alteração no ASCII e no hexadecimal tinha alguma relação com o local no arquivo que sofre alteração: se uma alteração era feita no final do texto, o *glitch* aparecia perto do fim da imagem.

Figura 6 - *Glitch* com o local controlado

Fonte: Arquivo pessoal (2017)

Nem sempre essa era uma relação direta. Abaixo temos uma alteração que não resultou em um *glitch* no local pretendido. O *glitch* ainda assim alterou a imagem de forma interessante. Essas experimentações nos dão noções que ajudam no processo de criação.

Figura 7 - Falha ao tentar controlar o local do *glitch*

Fonte: Arquivo pessoal (2017)

A falha no *glitch* pode ser atribuída a teoria de que a alteração no código tenha danificado a imagem de forma visualmente imperceptível, ou que os trechos do código hexadecimal ou ASCII que foram alterados não tinham função direta no resultado visual da imagem. Há ainda a possibilidade de esse resultado ser uma característica única do código desta imagem, podendo ser sugerido que em outra imagem obteríamos resultados diferentes. Em nosso entendimento, mesmo que não apareça visualmente, cada inferência de *glitch* numa imagem está intrinsecamente ligada ao código alterado, mesmo que de forma diminuta. Porém, mais experimentos seriam necessários para confirmar cada sugestão citada.

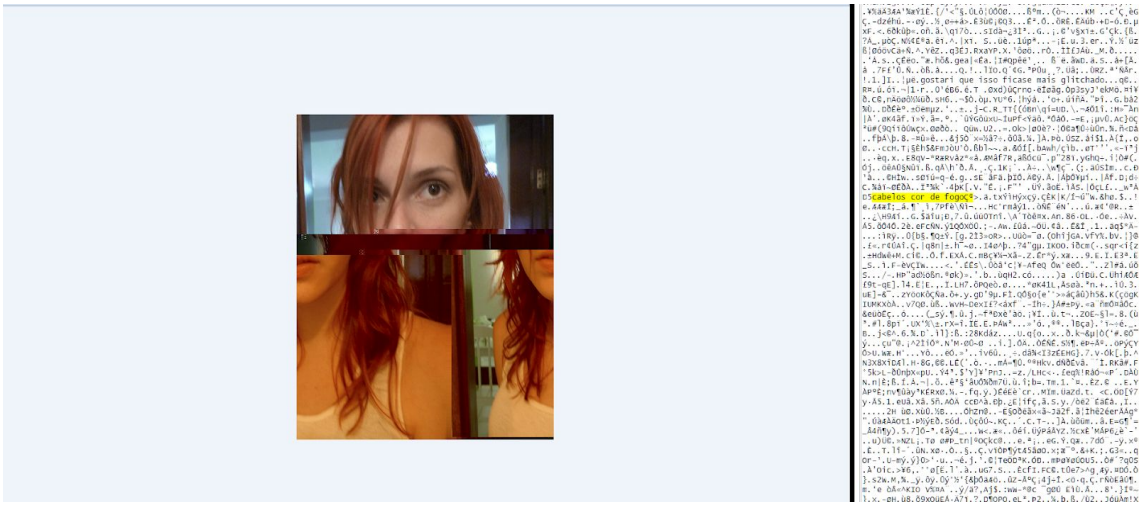
IV.4 *Glitch* poético

Esse método de alteração do ASCII se sedimentou através do uso do Frhed. Como as questões de funcionalidade do experimento estavam satisfatórias, exploramos novas possibilidades artísticas. Passamos a atribuir significado às alterações do ASCII, não simplesmente alterando os caracteres de forma randômica, como havia sido feito anteriormente. Começamos assinando o *glitch* criado com o nome dos pesquisadores no ASCII. Esse processo foi evoluindo para introduzir palavras que

induzissem o resultado esperado do *glitch,* expressando raiva, tristeza ou frustração dentro do ASCII do arquivo, às vezes pequenas anedotas ou pequenos parágrafos, e por fim, deletando pedaços do ASCII. Os resultados estéticos obtidos eram dos mais variados. Essa noção de que as alterações poderiam ter valor poético foi compreendida nessa etapa e trouxe várias experimentações.

Podemos ressaltar que a própria intervenção na imagem, mesmo sem dar significado às alterações no código, nos faz artista e pesquisador, como sugere Venturelli: “Na primeira categoria o computador é utilizado como ferramenta e na segunda categoria o artista é também programador ou trabalha com programadores para desenvolver obras computacionais, interativas ou não.” (VENTURELLI, 2010, p.1). Trabalhamos com a hipótese da segunda categoria, em que o programador é o artista. A resignificação do código ASCII para a língua portuguesa ou inglesa dentro do arquivo é apenas mais uma camada artística da *glitch art.* Fora a imagem vista, se o arquivo for aberto por um outro artista de *glitch* ou programador essas mensagens estariam embutidas junto a obra visual.

Figura 8 - *glitch* poético



Fonte: Arquivo pessoal (2017)

Da perspectiva funcional, não estávamos mais atrelados a trabalhar com arquivos BITMAP, podendo alterar também arquivos JPEG. Isso foi útil, pois o arquivo JPEG é uma formatação mais comum e eliminava a necessidade de renderizar os arquivos em um novo formato para poder trabalhar com eles, pois as imagens que estavam sendo utilizadas estavam geralmente em JPEG, precisando ser

convertidas para BITMAP para conduzir os experimentos. Como, para o resultado final de uma imagem, se gerava uma série de versões salvas, eliminar uma parte desse processo laborioso é recompensador.

IV.5 Audacity

Outra forma de experimentação, baseada na segunda forma de *databending* sugerida por Menkman, é a utilização de um software subvertendo a sua funcionalidade original para se fazer o *glitch*. Para tal, utilizamos o programa *opensource* Audacity[50]. O Audacity é um DAW[51] popular com interface simples e funcionalidades básicas para edição de arquivos de áudio. Porém, o que o torna popular para artistas de *glitch* é a sua capacidade de importar arquivos em formato *raw*[52]. Essa funcionalidade do programa torna possível a importação de arquivos de imagem para dentro do ambiente de trabalho de áudio.

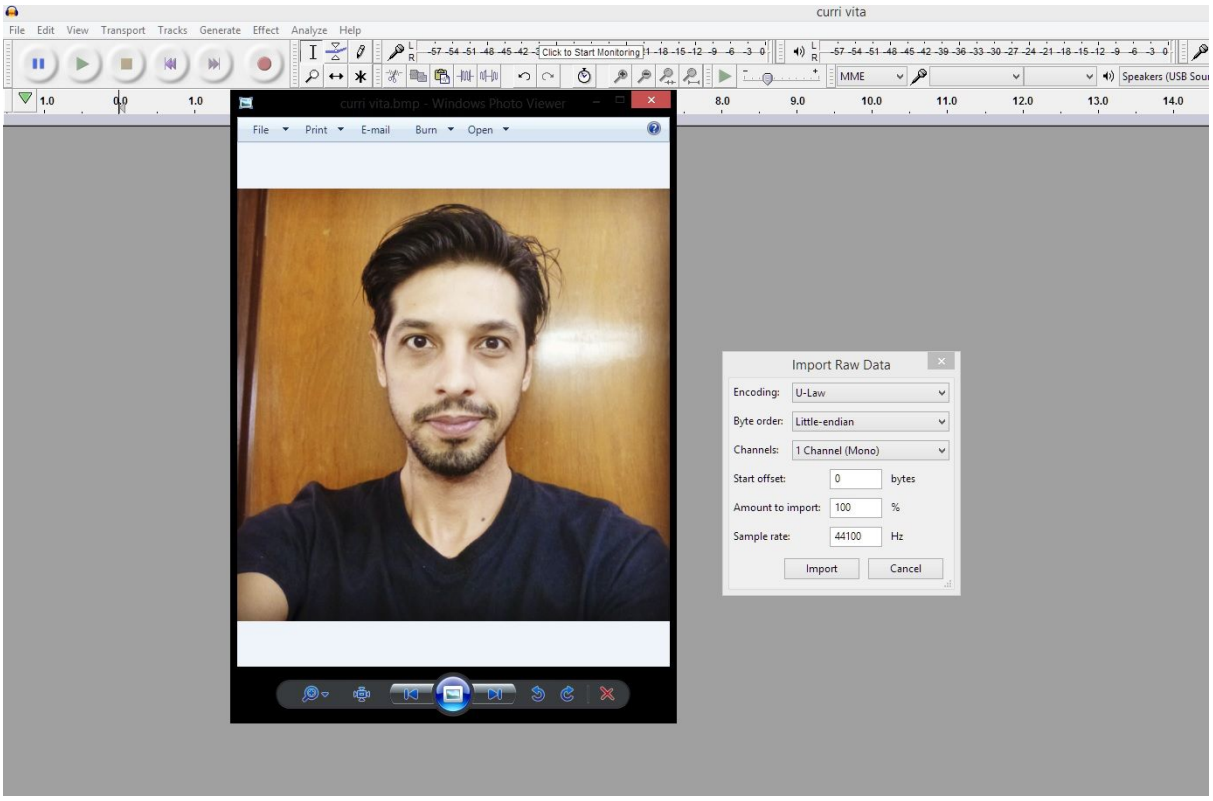
Para manter a funcionalidade do arquivo de imagem, as seguintes configurações são recomendadas: primeiramente trabalhar com arquivos de imagem em formato BITMAP. Ao importar o arquivo usar a opção de encoding U-law ou A-law, lembrar de anotar essa opção pois o mesmo tipo de exportação deverá ser feita para se poder ter uma chance de funcionamento do arquivo. Na opção de *byte order* se recomenda testar as opções *little-endian* e *big-endian*, em nossa experiência estes sendo os de maior chance de sucesso. Recomenda-se não utilizar a opção de *offset,* pois ela vai eliminar pedaços do *header* durante a importação. Na configuração *amount to import* - em português a quantidade a ser importada - recomenda-se deixar em 100%. As demais opções ficam a gosto do artista experimentar.

Figura 9 - Audacity importar

[50]In: < http://www.audacityteam.org/ >.

[51]Digital audio workstation, em português estação de trabalho de áudio digital.

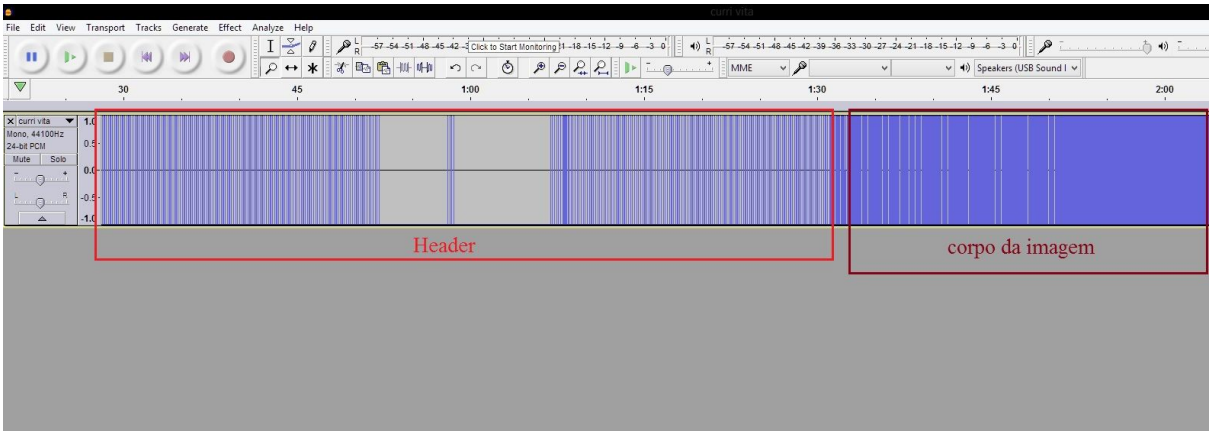
[52]Raw ou, em português, arquivo crú é um formato de arquivo que não utiliza compressão e possui maior quantidade de informação embutida no arquivo de mídia.



Fonte: Arquivo pessoal (2017)

Após importação do arquivo, seguimos as regras que determinamos anteriormente com a codificação ASCII, identificamos visualmente o *header* do arquivo e não fazemos alterações nele. A utilização da lupa dentro do Audacity facilita a identificação desses padrões visuais.

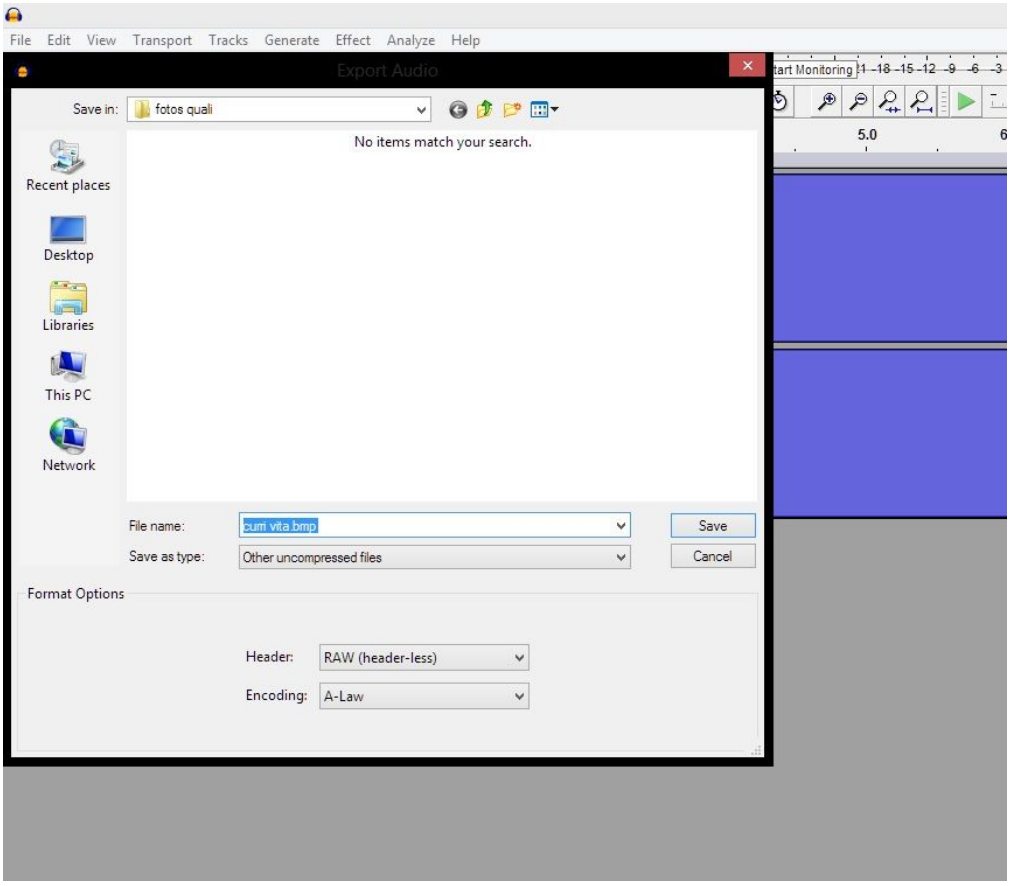
Figura 10 - *header* e corpo



Fonte: Arquivo pessoal (2017)

Após o processo de importação ter sido concluído, qualquer efeito utilizado em arquivos de áudio pode ser utilizado no corpo da imagem para distorcê-la, com cada efeito de áudio atribuindo um efeito visual diferente à imagem. A composição final da imagem só poderá ser vista após a exportação do arquivo e não existe garantia de que a funcionalidade do arquivo tenha sido preservada, mesmo protegendo o *header*. Recomenda-se manter o mesmo tipo de exportação, ou A ou U law, que se utilizou para importar e pode renomear o arquivo para o formato de imagem, no caso BITMAP.

Figura 11 - exporta Audacity



Fonte: Arquivo pessoal (2017)

Apesar de ter tido resultados favoráveis como mostra a figura 13, o processo é bastante laborioso e a chance de quebra do arquivo é alta. Outras opções de programas que suportam importar o arquivo em formato raw podem ser utilizadas de forma similar, outros DAW, como, por exemplo, o Adobe Audition, mas, baseado em nossa experimentação, o controle sobre a preservação do arquivo é baixo e a utilização de outro DAW no processo não deve modificar os resultados.

A utilização de DAW ou outro *software* para se fazer o *databending* passa uma intenção de controle do *glitch*, pois em cada efeito de áudio que se aplica à imagem, se espera um resultado parecido com o efeito de áudio original, ou uma interpretação de que existe uma relação do resultado que o efeito de áudio aplica a imagem, em realidade não sendo esta necessariamente uma relação direta.

IV.6 Música, bits, matemática e *glitch*

Tendo as informações estabelecidas na fundamentação teórica acerca do processo para a experimentação de *glitch art* e música, passamos a procurar opções compatíveis com a forma que gostaríamos de trabalhar. Optamos por utilizar o Glitch do Naive Sound que é um audio engine que gera *bytebeat*.

> A música bytebeat não tem nenhuma notação musical convencional - não há notas ou acordes ou mesmo instrumentos. Tudo o que você tem é uma expressão matemática que é avaliada para cada período de tempo para gerar o próximo valor de amplitude da onda sonora. (GLITCH NAIVE SOUND, 2014) [53]

O software Glitch é, na verdade, uma emulação do Chiptune que a Naive Sound criou. Trata-se de um sintetizador que obedece a uma sintaxe matemática para gerar áudio. O resultado musical desse processo foi nomeado bytebeat. O Glitch da Naive Sound é um emulador que funciona em um navegador rede[54], tem alguns exemplos de composição disponíveis que facilitam o processo de composição através de expressões matemáticas, e também tem a facilidade de exportar (criar um arquivo de áudio) em formato WAVE[55].

Durante o processo de desenvolvimento, percebemos que mesmo com conhecimento musical ainda havia uma carga elevada de conhecimento a se adquirir sobre essa relação entre a música e a computação. Muitas vezes, o resultado parecia excessivamente aleatório para ser mesclado à música

[53]Disponível em http://naivesound.com/. Acessado em 16 out 2016 ,tradução nossa.

[54]Do inglês Browser ou web browser, navegador de web, navegador de internet.

[55]WAVE é um formato de arquivo de áudio que foi desenvolvido pela Microsoft. Um arquivo Wave é identificado por uma extensão de nome de arquivo do WAVE (.wav).

que vínhamos compondo em cima de frases 8 bits. Assim, fomos nos adaptando e trabalhamos com a aleatoriedade e o acaso, considerando isto uma interferência externa que influenciava na composição. Um artigo em específico nos serviu como base, uma pedra de Rosetta[56] entre as equações e a notação musical tradicional. Trata-se do artigo intitulado "Some deep analysis of one-line music programs"[57], publicado no blog Countecomplex do Viznut's. A partir dele, criaríamos a base para a música com o software Glitch: frequências graves que serviram de base, um ritmo, e alguns floreios mais melódicos, e acrescentando um instrumento mais tradicional, no caso, a guitarra e posteriormente a voz.

Acabamos utilizando essas expressões matemáticas para gerar áudio. A primeira cede as notas de frequências graves e gera um ritmo. Abaixo apresentaremos a forma de utilização das expressões matemáticas e em seguida as que foram utilizadas em nossa composição:

> Tudo no Glitch é um número. O tempo é um número. Notas são representadas como números. Cada sinal é representado como o valor de sua amplitude atual. É por isso que a música no Glitch é feita usando os seguintes operadores aritméticos:
> Aritmética: + - * /% (módulo) ** (potência)
> Bitwise: & | ^ (xor ou bit a bit não) << >>
> Compare: ==! = <<=>> = (Retorno 1 ou 0)
> Agrupamento: (), (separa expressões ou argumentos de função)
> Condicional: && || (operadores de curto-circuito)
> Atribuição: = (lado esquerdo deve ser uma variável) (GLITCH NAIVE SOUND, 2018)[58]

Os operadores aritméticos têm o mesmo significado que possuem na linguagem de programação C. A prioridade dos operadores também é a mesma utilizada na linguagem C. Como o Glitch da Naive Sound, os valores numéricos determinam frequências, tempos, tipos de ondas e timbres. A maior parte das características da composição também podem ser misturadas utilizando o +. O + pode ser usado também para misturar outros sinais aritméticos juntos, e o - pode ser usado para subtrair um sinal de outro. Com esse conhecimento desenvolvemos nossa primeira frase musical:

[56]Programa de código aberto. Tradução nossa.

[57]Uma análise profunda de programas de música de uma linha. Tradução nossa.

[58]Tradução nossa.

(t>>6|t%67)-(t>>4)

Esta outra serve como um floreio melódico:

(t>>9)|(t%1)|(((t%1901)|(t>>1))&((t>>49)|(t*t%19)))

Nosso objetivo principal era mesclar os instrumentos tradicionais e o *bytebeat*. Tentamos criar uma linha de guitarra em 8 bits, que substituiria a linha executada no instrumento, a fim de termos menos presença da guitarra na música. Esta foi a expressão usada:

t*(t>>204)+t*(t>>1088)+t*(t>>0)

Foi um longo processo de tentativa e erro para entender o que cada sentença ou fórmula matemática ocasionava na música. A nossa composição se deu a partir dessa base eletrônica, com posterior utilização da guitarra. A música mudou bastante ao longo da experimentação com as expressões matemáticas. A influência do som do computador e a falta de controle total sobre o resultado nos fez perceber que estávamos compondo em conjunto com a máquina, e não sozinhos. Em que pese a necessidade constante de uma decisão final de nossa parte, mostrou-se bastante criativo este processo de experimental de áudio.

As gravações e mixagem foram feitas com o Audacity[59], adicionamos as linhas de voz que foram gravadas às outras linhas já existentes. A guitarra foi regravada com a intenção de acentuar o áudio *bytebeat*. A técnica de edição do áudio também foi modificada, visando mimetizar o *bytebeat* com cortes brutos e efeitos que produzem uma aparência de erro. Sendo, porém, gerada

[59]http://www.audacityteam.org/

intencionalmente essa dualidade entre erro e acaso, texturas ríspidas e suaves trazem à música um aspecto de caos controlado, entrelaçando-se com a letra taciturna.

Outra característica da edição e mixagem foi de não haver preocupação em corrigir a notória mudança de sinal quando se introduz ou retira um clip de áudio abruptamente. Isso foi feito intencionalmente para se tentar reproduzir um som mais mecânico.

Em nossa criação sonora com base no *glitch art*, cumpre esclarecer que o que foi utilizado na introdução da música é um trecho do áudio exportado[60], que passou pelo processo de glitch através de um editor hexadecimal código aberto o Frhed[61]. Por vezes, introduziu-se um texto randômico ou parte integrante da letra da música na codificação ASCII do arquivo de áudio, sendo mudados valores na codificação hexadecimal do arquivo, a fim de corrompê-lo para se obter aspectos sonoros interessantes para a música. Assim, a letra da música, o erro gerado por essas interferências e o acaso são elementos usados na manipulação da música. A interferência gerada por incluir as letras no código ASCII, onde ela é ressignificada, cria uma nova camada da construção poética para a música e não somente uma composição sonora, mas uma obra de glitch art.

As mudanças não podiam ser muito abruptas, pois se modificarmos o header de um arquivo desse modo o arquivo simplesmente não funcionaria, já que o header contém informações essenciais para o sistema operacional reconhecer o que é o arquivo e como deve proceder para utilizá-lo. Como consequência, tivemos uma preocupação em deixar o header, parte onde se tem informações essenciais ao funcionamento do arquivo, intacto. Através de erros, tentativas e várias repetições de modificações que haviam dado certo, fomos construindo uma sequência de arquivos cada vez mais modificados, que ainda funcionavam e emitiam um áudio interessante para a música. Por fim, a versão de número 24 foi a utilizada.

Durante a mixagem, o desafio era entrelaçar o som de 8khz com o de 8bit e o som de 192khz com o de 24bits. Isso geralmente não é uma prática comum, já que os editores de áudio, por padrão, querem converter o áudio para um formato único. Adotamos a técnica de acentuar essas diferenças,

[60] Áudio gerado a partir do programa Audacity.

[61] http://frhed.sourceforge.net/en/

deixando assim as características de cada tipo de áudio serem mais notórias, e criando uma textura sonora diferente do habitual.

O áudio finalizado foi convertido do WAVE para MPEG-1/2 Audio Layer 3 (MP3)[62] e o arquivo foi editado novamente no Frhed para se obter novos glitches, sendo algumas partes tratadas com um efeito chamado bit crusher. No caso foi usado o bit crusher criado pela dblue, ferramenta que que reduz a resolução (quantidade de informação que representar a amostra) do arquivo de áudio, possibilitando manipular o áudio com maior controle em áreas selecionadas.

Na última etapa da produção, a música foi estendida de cinco para onze minutos, depois de passar por diversos efeitos, especialmente o bit crusher, que a deixou com o som mais digitalizado, depois de reduzir-lhe alguns bits. Essa extensão foi pensada durante os ensaios do espetáculo multimídia de dança, em que a música criada tornou-se trilha sonora. Destacamos o uso do delay GSI WatKat, aplicado durante o processo de mixagem.

Ainda que o processo de masterização não se distinga do tratamento convencional, percebemos que a falta de graves nos arquivos 8 bits era mais perceptível nessa etapa. Entretanto, tomamos isso novamente como uma característica do áudio a ser explorada e não como falha.

A descrição do processo, as técnicas e programas utilizados têm o intuito de facilitar a replicabilidade do experimento ou servir como sugestões de uso dessas ferramentas open source em pesquisas com práticas similares. Levamos em consideração o baixo custo da utilização desses programas, pois funcionam em computadores com configuração de baixo custo (com hardware barato) e porque são gratuitos.

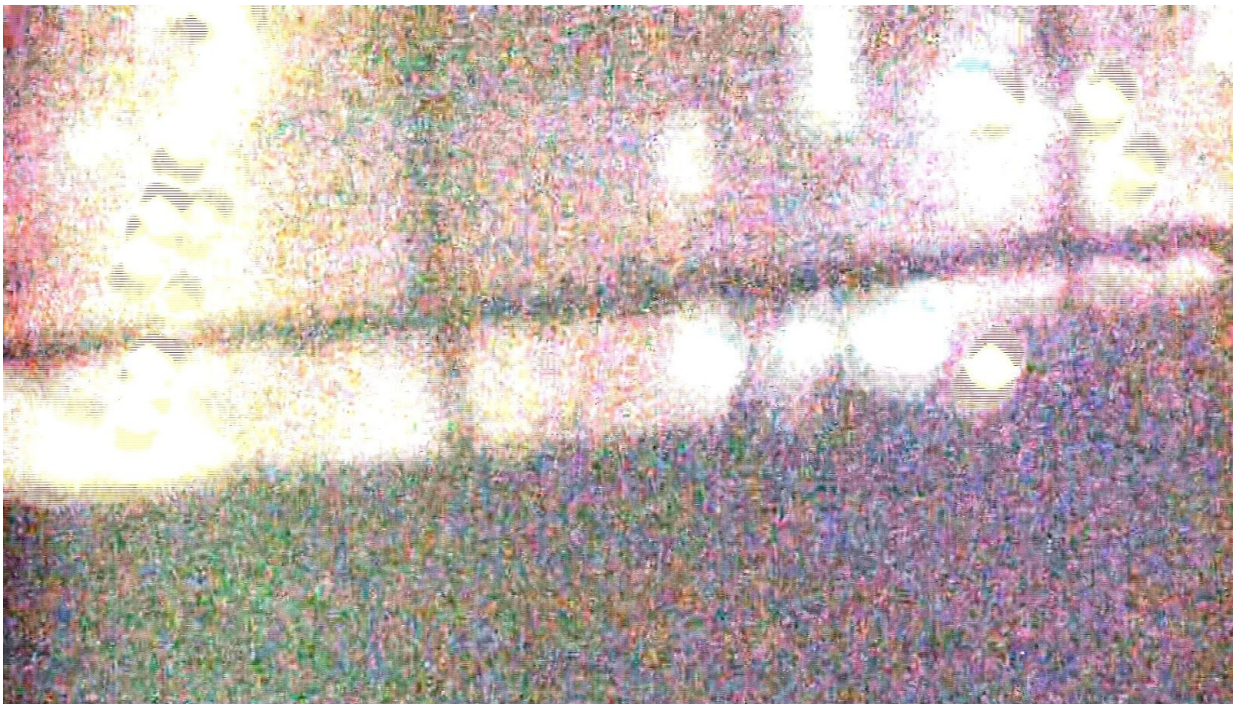
[62]**MPEG-1 or MPEG-2 Audio Layer III**, mais comumente referido como **mp3**, é um codec de formatação de áudio digital.

IV.7 Vídeo *glitch*

Como parte da experimentação com *glitch art*, optamos por desenvolver uma videodança[63]. A premissa era de que, se podíamos editar uma imagem estática para produzir o resultado de *glitch,* possivelmente o vídeo, que é um conjunto de imagens estáticas, teria o mesmo resultado. Editamos vídeos previamente produzidos que continham as filmagens de uma performer no cenário de uma piscina com baixa iluminação à noite. A filmagem original tinha uma característica visual de baixa fidelidade, devido à pouca iluminação do local. Outro detalhe importante é que os arquivos a serem importados para o editor de vídeo possuíam um formato que não era 100% compatível com a plataforma. Logo, aproveitando esse erro técnico de iluminação e usando a própria importação para gerar alguns *glitches*, resolvemos que a melhor forma de editá-lo seria com uma estética de *glitch art*. Abaixo um exemplo do ruído visual das imagens por conta da baixa iluminação e do processo de importação do arquivo.

Figura 12 - Ruído na imagem

[63]"Considerando-se o amplo horizonte conceitual existente com relação à videodança, optou-se, neste artigo, por discutir a produção baseada na relação do corpo que dança, com a câmera, como material a ser manipulado por meio da edição e procedimentos de pós-produção para ser visualizado em uma tela ou projeção" (SCHULZE, 2011, p.1)



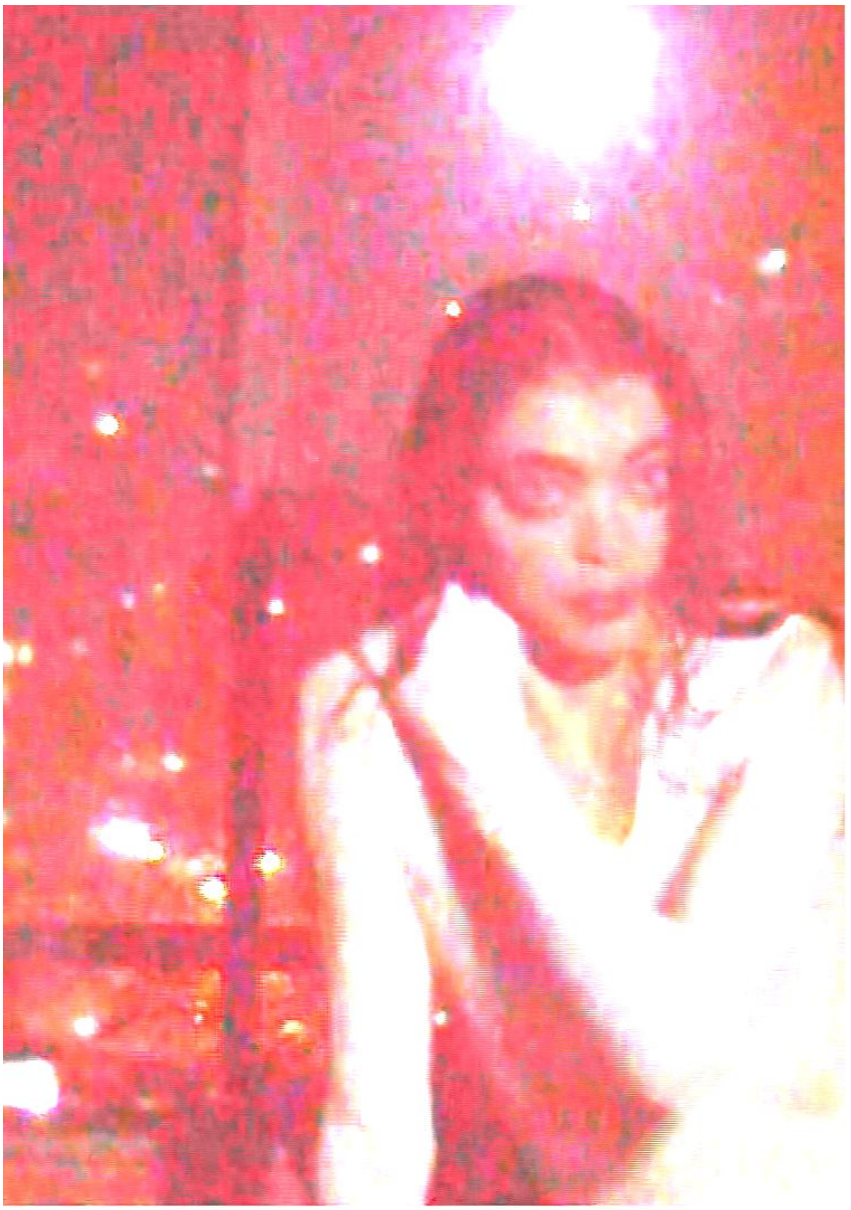
Fonte: Arquivo pessoal (2018)

O ruído digital lembrava a imagem de fitas VHS deterioradas pelo tempo. Optamos por aplicar elementos de *glitch* em alguns arquivos ou pedaços de arquivos utilizando o Frhed como havíamos feito anteriormente. Para a trilha sonora do vídeo, utilizamos a música criada em nosso experimento anterior. Utilizamos o Davinci Resolve 14[64] para editar as imagens e, com isto, pudemos fazer uso de correções de cores exageradas e mantivemos um estilo de cortes das imagens condizentes com a movimentação da performer e os glitches criados, juntando esses elementos de *glitch* aos de *glitch like* [65]. Englobamos nesta obra elementos como o estilo de cortes e a correção exagerada de cores com duas tonalidades principais: as cenas vermelhas e as cenas azuis.

Figura 13 - Cenas vermelhas

[64]In: < https://www.blackmagicdesign.com/br/products/davinciresolve/ >. Editor de vídeo popular com recursos modernos e oferecido gratuitamente.

[65]Parecido com *glitch,* ou imita *glitch*,. (MCCORMACK, 2010, p.15, tradução nossa) pode também ser referido como *glitch alike* termo cunhado por Moradi (MORADI, 2014, p.39)



Fonte: Arquivo pessoal (2018)

Resultando em uma videodança no estilo da glitch art, esse processo de criação baseado em erros de vídeo pode ser revisto no trabalho da pesquisadora Beiguelman:

> Quando cheguei à galeria para colocar a peça toda os artistas já definiram seus próprios espaços e havia apenas um quarto muito pequeno com paredes muito altas sobrando para mim. A única maneira de mostrar a peça era empilhar os dois vídeos em cima um do outro. Esta situação final de "erro" levou-me a re-editar o vídeo que retrata a parte inferior do viaduto de cabeça para baixo, o que produziu o efeito de uma continuidade das linhas verticais na instalação de vídeo. A maneira acidental pela qual eu cheguei à configuração final dessa peça sinalizou não apenas meu primeiro contato com a estética da falha, mas a descoberta de quão semelhante a materialidade da corrupção do código era para os atributos sociais ruidosos de algumas das grandes cidades da América do Sul, sobretudo São Paulo. (BEIGUELMAN, 2015 p.3)[66]

Figura 14 - Cenas azuis

Fonte: Arquivo pessoal (2018)

Por fim, a videodança[67] contém uma construção narrativa visual da morte da performer acompanhando a degradação da imagem e da trilha sonora, a degradação de toda a composição do vídeo fazendo uma integração com a glitch art como uma de suas facetas, a desconstrução, destruição, a sua tendência ao caos como dito anteriormente.

[66]Tradução nossa.
[67]In: < https://vimeo.com/205762559 >.

V. Desenvolvimento Com Pure Data, Live Glitch & Paisagem Sonoras

Verificando que havia uma diferença entre o conhecimento dos desenvolvedores do fórum de *Pure Data,* que utilizamos para sanar algumas dúvidas e buscar sugestões, e o nosso sobre *Pure Data*, resolvemos desenvolver um projeto próprio, em área que pertence ao escopo da pesquisa, que pudesse servir para aprimorar os conhecimentos na linguagem.

Logo surgiu a oportunidade de criar um *patch* com características de *glitch like,* elemento da *glitch art* discutido por autores de nossa bibliografia. O *patch* faria parte de uma performance de dança, fazendo a captura de movimentos da performer e reinterpretando essa captura de imagem para posteriormente projetá-la, dando espaço para o performer trabalhar o espaço e o corpo físico e virtual. Para isso contamos com a participação da performer Luna Dias[68] para o experimento.

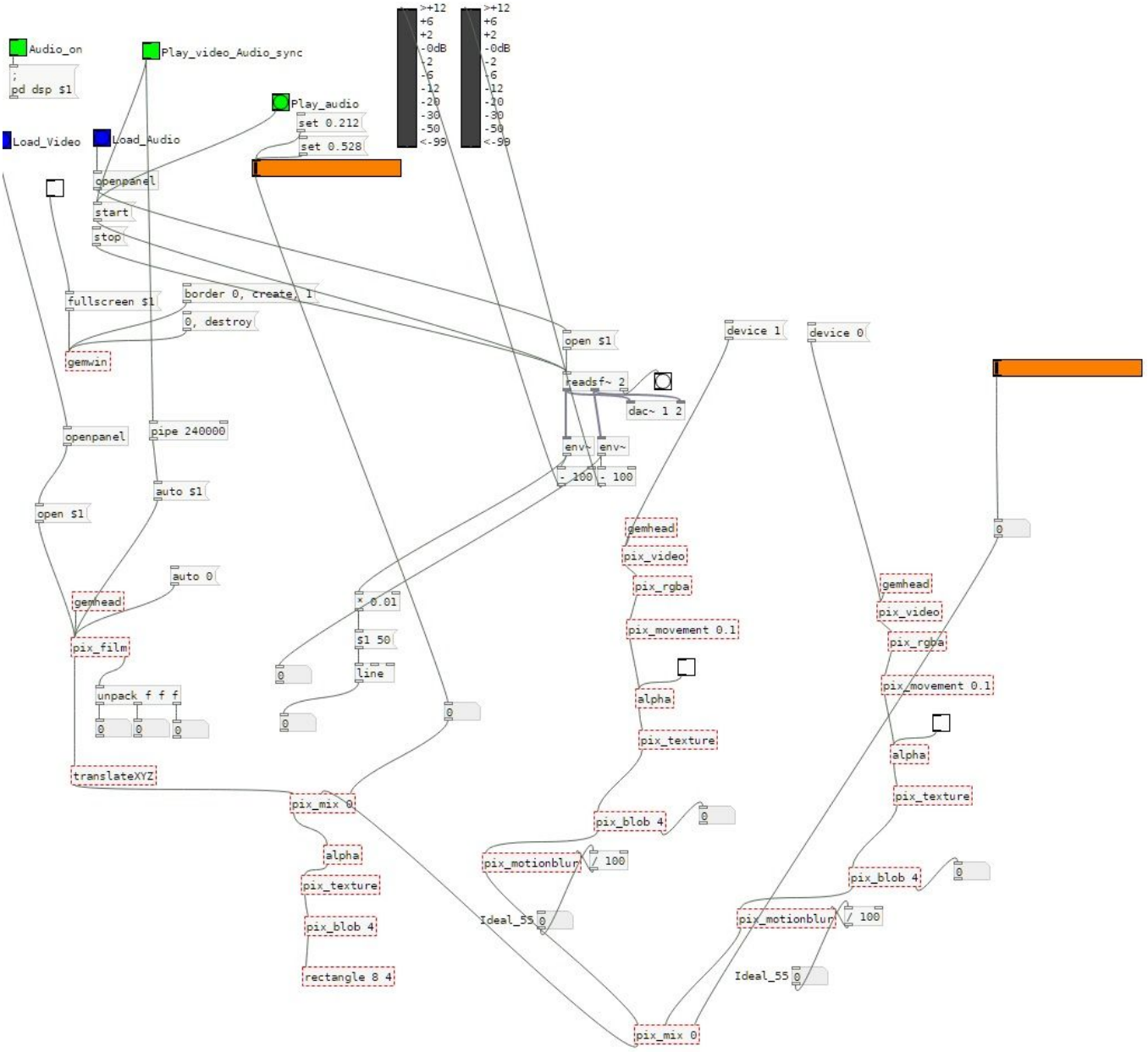
Buscamos criar no *patch glitch like* que gerasse elementos visuais de *glitch art,* porém, não sofresse com erro ou falhas reais, aproveitando apenas a estética do erro.

Fizemos isso com um *patch* que captura as imagens utilizando uma câmera USB, e processa a imagem para uma versão pixelada exibindo as bordas da imagem original. Isso é feito analisando a diferença entre dois frames de um vídeo e exibindo apenas a diferença dos dois, gerando assim as bordas de uma imagem pixelada, utilizando o canal alpha[69] de ambas as imagens para transmitir essa informação. Com essa base estabelecida, uma segunda câmera foi acrescentada ao patch para possibilitar a adição de diversos efeitos manualmente, como imagens sobrepostas. A segunda câmera podia ainda ser utilizada como um controlador das cores e iluminação do que se passava na tela, um controle através de hardware em vez da utilização do *software*. Essa é nossa experiência com *Pure Data* para gerar elementos de *glitch like*.

Figura 15 - *Patch glitch like*

[68]Luna Dias, licenciada em Dança pela Universidade Federal da Bahia UFBA, com especialização em estudos contemporâneos em dança pela UFBA, e mestranda do Programa de Pós-Graduação em Computação, Comunicação e Artes PPGCCA da UFPB.

[69]Canal reservado para transparência da imagem



Fonte: Arquivo pessoal (2018)

Este patch foi desenvolvido ao longo de algumas versões, porém, apresentamos aqui a versão utilizada na performance.

Seu funcionamento consiste nas operações de quatro objetos: o pix_film, pix_movement, pix_video e pix_mix. Seguindo o fluxo de informação na imagem acima, começaremos da esquerda para a direita a descrição do seu funcionamento. [Pix_film] tem a função de executar a videodança que faz parte de uma das camadas de vídeos da apresentação. Esse vídeo e todas as outras camadas são

exibidos na janela criada pelo objeto [gemwin]. Como uma interação a mais no áudio da performance, a música discutida no IV.6 é executada no patch utilizando os objetos [readsf~ 2] e o [dac~ 1 2] sendo que os valores de decibéis da música alteram o posicionamento do vídeo de pix_film, em seus valores X e Y, com os objetos [translateXYZ] e [unpack f f f] , no caso o [valor Z] é referente ao eixo Z cartesiano, designando quem ocupa a camada primária entre [pix_film] e os dois [pix_video].

Construída assim uma base visual e musical para a verdadeira interação através dos objetos [pix-video], ambas as árvores de objetos de [pix-video] no *patch* são cópias idênticas, com o diferencial sendo a forma de utilização da câmera usb conectada a eles através da mensagem [device 1] ou [device 0]. [Pix_video] recebe o stream da imagem seguindo o fluxo de dados, [pix_rgba] separa em canais RGB e o nosso principal interesse o canal alpha, representado por A. O objeto [pix_motionblur] é utilizado durante a performance para criação do rastro das imagens capturadas, podendo-se alterar seus valores para aumentar, diminuir ou anular seus efeitos. Por fim, a combinação das três imagens segue para [pix_mix], que é controlado pela barra vertical laranja à esquerda e que altera os valores de transparência dos fluxos das imagens, permitindo controlar qual imagem se sobressai na sobreposição indo para a janela de saída [gem_win.]

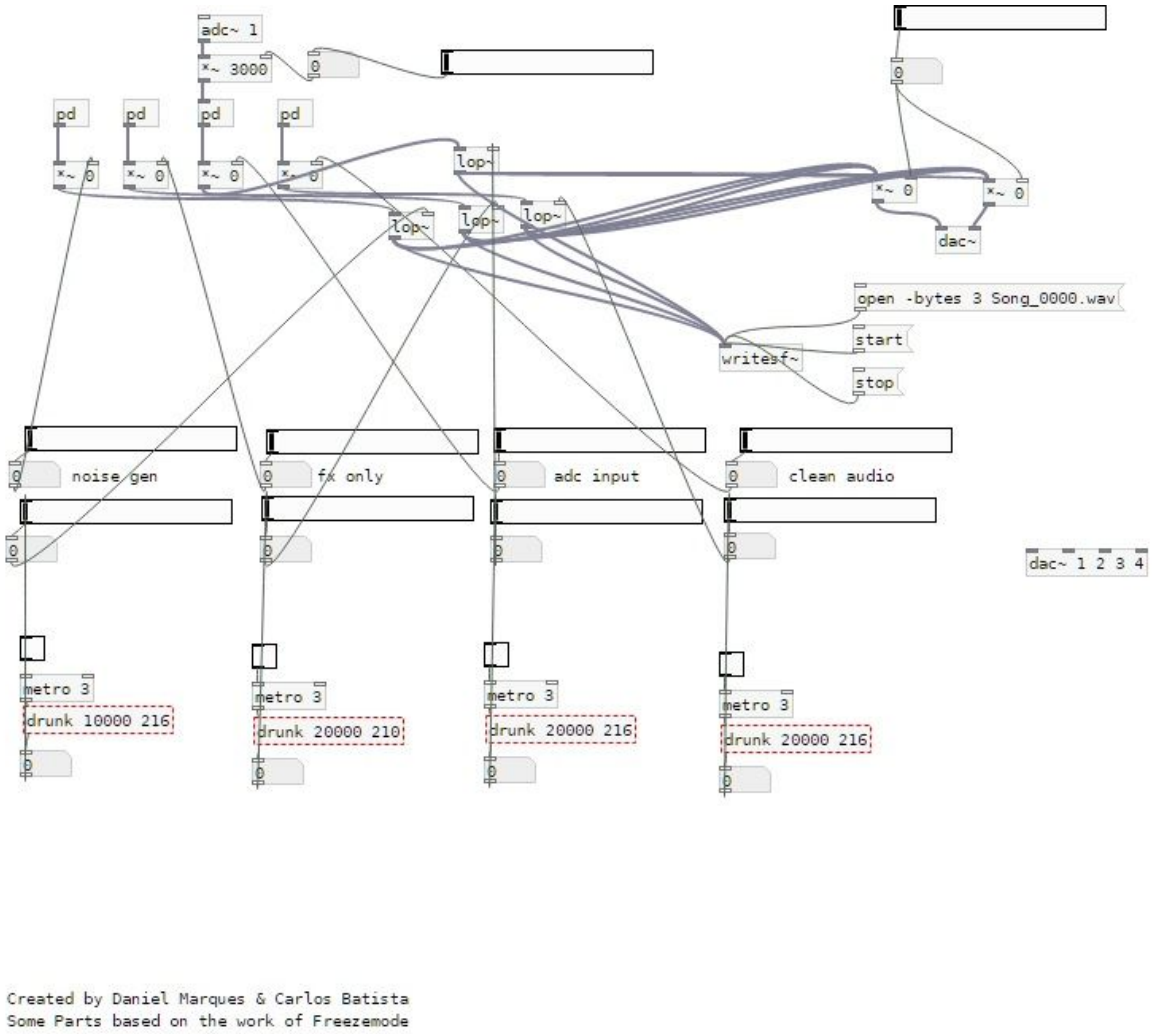
Além do objetivo principal, que era a capacidade de gerar as imagens pretendidas, esse patch pôde executar a música que é trilha sonora da performance e a videodança que foi desenvolvida em uma de nossas pesquisas, desta forma misturando as imagens da videodança com as imagens capturadas em tempo real. Eis os efeitos visuais que eram projetados em cima da performer gerando interação em tempo real.

As imagens exibidas são da performance de dança e tecnologia idealizada junto com a construção do *patch* com a performer Luna Dias. A trilha sonora consistia da música criada em nosso experimento de música *glitch*. A performance foi apresentada no Janela Quebrada: cinema de invenção e risco[70] em que projetamos a imagem em uma parede por trás do Cine Aruanda da UFPB com um projetor 4k, e simultaneamente em um painel de LED.

[70]In: < https://www.facebook.com/janelaquebrada/videos/vb.974396412629418/1124237610978630/?type=2&theater >.

Com o sucesso da performance e do funcionamento do *patch*, passamos para o nosso próximo experimento: um *patch* voltado para música*,* criado durante o desenvolvimento do *Pure Data Script* para ser um de seus testes. A proposta era criar um *patch* que gerasse uma gama de sons automaticamente, em diversas tonalidades, volumes e timbres a fim de gerar uma paisagem sonora. As experimentações com o *Pure Data,* nos levaram a desenvolver o seguinte *patch:*

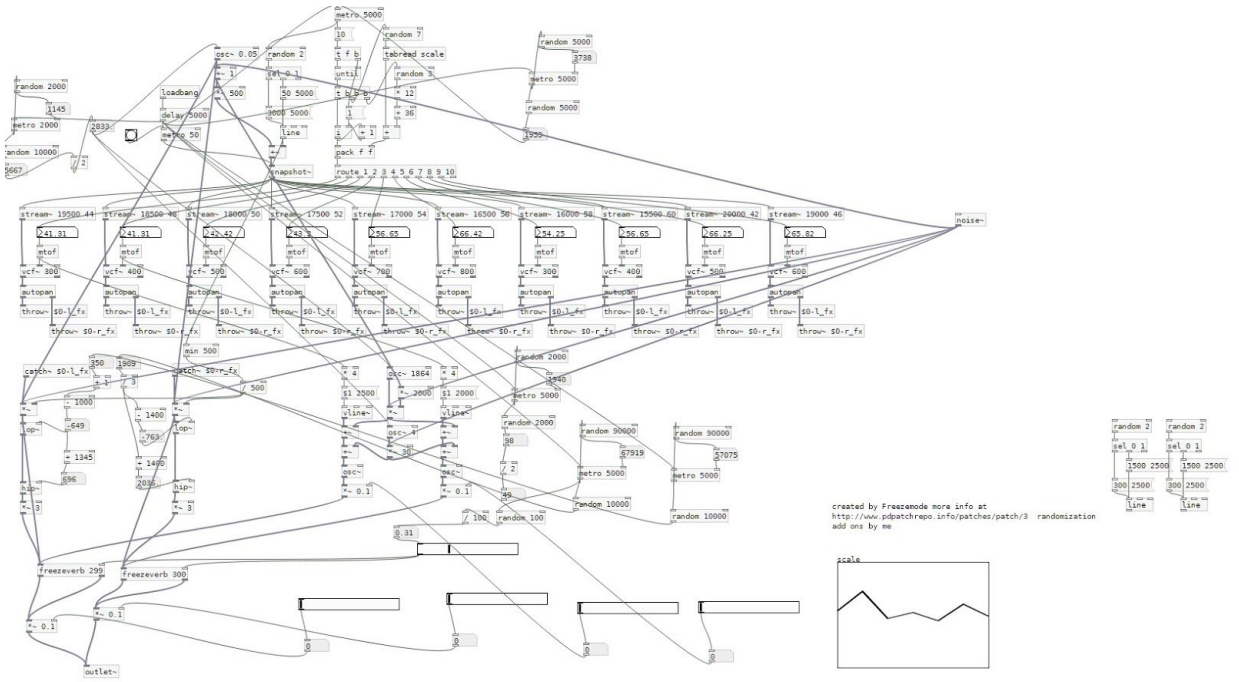
Figura 16 - *Patch* gerador de paisagem sonora interface



Fonte: Arquivo pessoal (2018)

O *patch* é dividido em quatro partes, que geram áudio e que, em si, já são paisagens sonoras. Porém, a soma das partes é que torna o áudio gerado realmente denso. Na interface, temos controladores de volume para cada canal, que permitem controlar a presença de cada elemento a gosto do usuário, e também um filtro de passagem alta[71], para controle de eventuais *feedbacks*[72] de som. Cada canal é composto de um *sub-patch* que será explicado em detalhes abaixo e é representado pelo objeto [pd]. No caso temos quatro: noise gen, fx only, adc input, clean audio. O *patch* também possibilita que, tanto os volumes dos canais como dos filtros, sejam controlados aleatoriamente através do objeto [drunk] que oscila seus valores de 10000 ou 20000 a 216 ou 210 a cada três ciclos determinado pelo objeto [metro 3]. Deixamos à disposição do usuário também a opção de gravar o áudio gerado em formato WAVE, possibilitado pelo objeto writef~ . Abaixo, examinaremos um dos primeiros módulos que compõem o *patch.* Primeiramente temos o gerador de ruído:

Figura 17 - *Patch* paisagem sonora, módulo gerador de ruído.



Fonte: Arquivo pessoal (2018)

[71]Um filtro de passagem-alta é um filtro que permite a passagem das frequências altas, porém atenua ou reduz a amplitude das frequências abaixo de frequência de corte.

[72]Retroalimentação, quando um sinal de áudio é retroalimentado ele gera um som alto capaz de danificar equipamentos e geralmente percebido como desagradável aos ouvintes.

Essa parte do *patch* gera um ruído randômico que oscila com o tempo, se reinventando a cada revolução do *patch*[73].

A nossa modificação consiste basicamente de um objeto principal, o [noise~], que gera ruído para o [osc~] principal, que por sua vez, alimenta todos os fluxos de dados que se encaminham para os objetos [freezeverb]. Estes últimos reverberam nove canais com ruídos diferentes, comandados pelos objetos [metro] e [random] com intervalos e valores diferenciados, que mudam a cada ciclo, com a função de dar aleatoriedade aos valores principais que comandam o gerador de ruído. Esse processo gera o canal de ruído que compõe parte de nossa massa sonora.

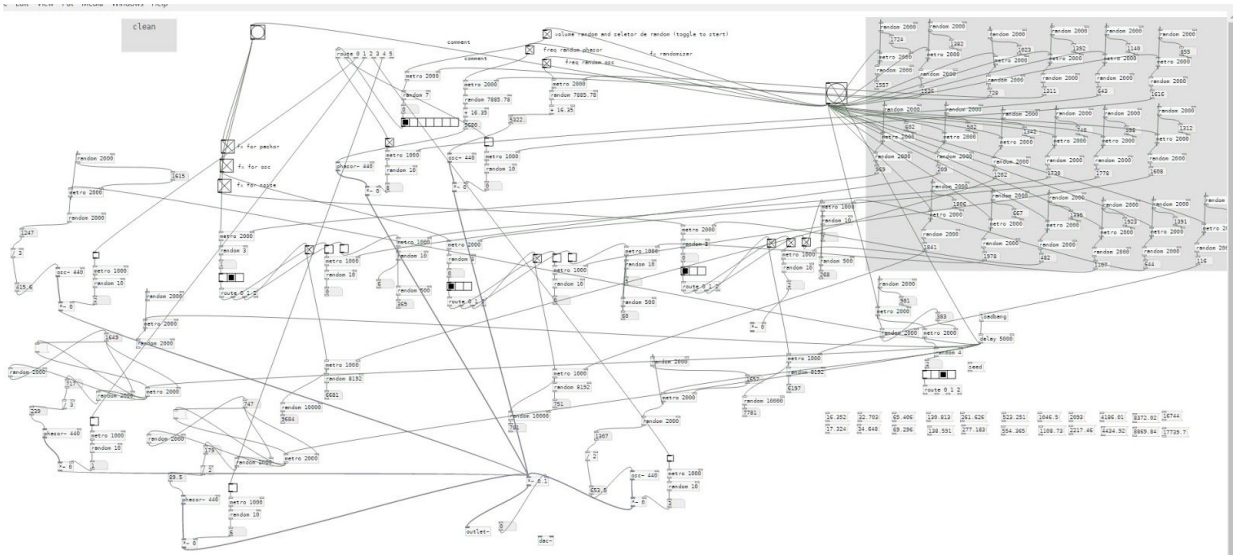
O próximo módulo ou *subpatch*[74] é o gerador de sinal limpo. Ele usa osciladores e *phasers*[75] sem efeitos, gerando sons em diversas frequências e tempos para utilizando novamente os objetos [metro] e [random] para atribuir aleatoriedade para criar a sua parte da massa sonora a partir de sua conexão com os dois objetos do *Pure Data* princiapais neste s*ubpatch,* o [osc~] e o [phasor~] . Abaixo veremos o *subpatch:*

Figura 18 - *Patch* paisagem sonora, módulo gerador de som sem efeitos.

[73]Essa parte é uma alteração do *patch* desenvolvido por um autor não identificado, que o usa o pseudônimo de freezemode.

[74]*Subpatch* é um *patch* criado dentro de outro *patch* com o propósito de manter uma organização visual, ou ocultar elementos.

[75]O efeito *Phaser* combina dois sinais de áudio,o sinal original e uma cópia fora de fase com o original. Isto significa que as amplitudes dos dois sinais, original e duplicado, atingem os seus pontos mais altos e mais baixos no espectro de frequência em momentos diferentes.

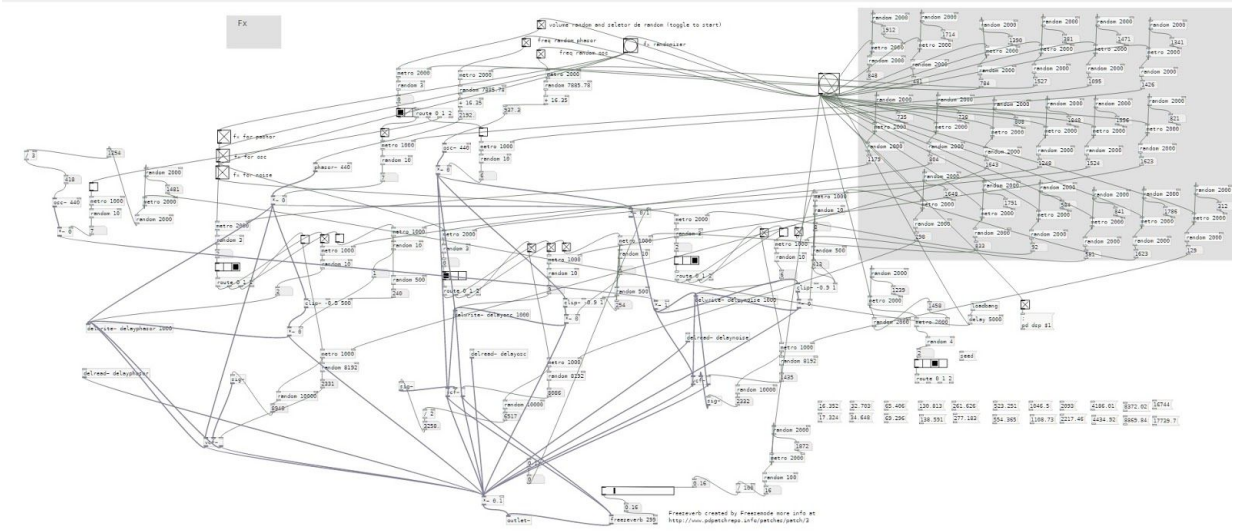


Fonte: Arquivo pessoal (2018)

Os valores das frequências a serem executadas são atribuídos aleatoriamente, dentro do espectro da audição humana, para cada sinal de áudio.

Passemos para o próximo módulo, gêmeo a este, o *subpatch* FX ou de efeito. Este *subpatch* gera o sinal de aúdio exatamente como o anterior, porém, os sinais sofrem alterações pelos objetos 1.[del write~] , 2.[clip~] e 3.[vcf~]. O primeiro atribui o efeito de *delay* ou eco ao sinal original; o segundo distorce o sinal; e o terceiro é um filtro de áudio baseado em voltagem, que na prática é um objeto que altera o timbre original do áudio. Abaixo, a implementação do nosso terceiro canal de áudio, que gera a paisagem sonora.

Figura 19 - *Patch* paisagem sonora, módulo gerador de efeitos.

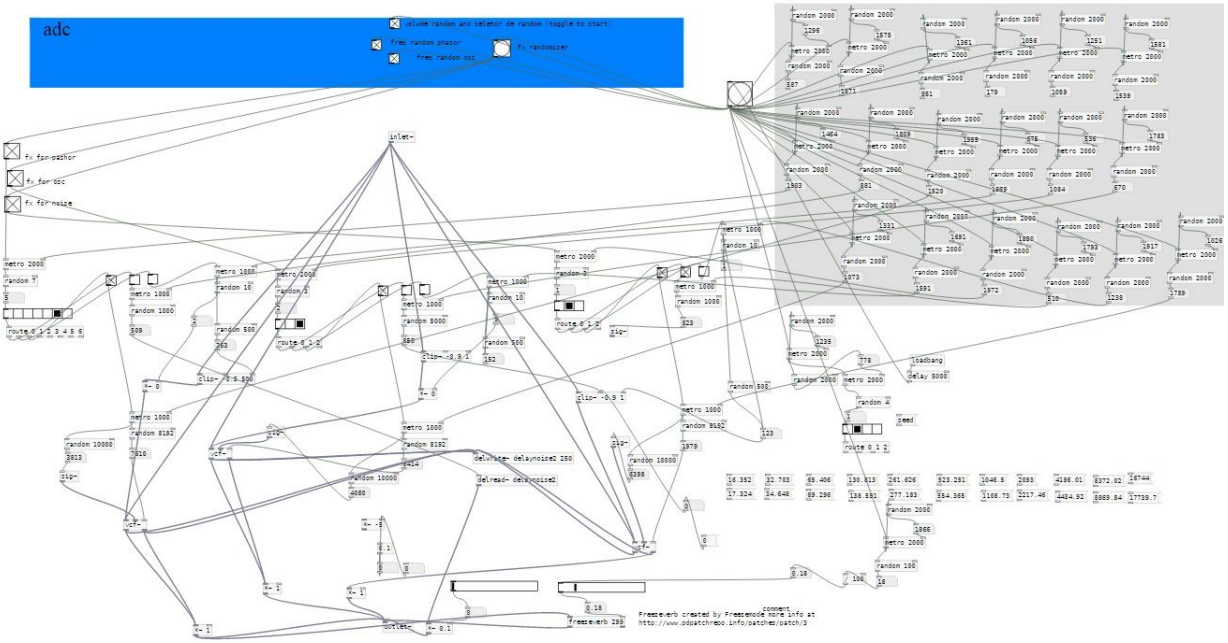


Fonte: Arquivo pessoal (2018)

O *subpatch* gêmeo do gerador de som com efeitos é o módulo ADC[76]. Ele serve de input para instrumentos, sintetizadores ou microfones que sofreram as mesmas alterações pelos objetos de efeitos citados no módulo anterior, porém, este módulo não possui nenhum [osc~] nem [phasor~], sendo assim, incapaz de gerar som sem ter algo conectado ao *patch*.

Figura 20 - *Patch* paisagem sonora, módulo ADC

[76]Analog Digital Converter, conversor de analógico para digital. Objeto do *Pure Data* que converte sinais analógicos em digitais.



Fonte: Arquivo pessoal (2018)

Tivemos sucesso no desenvolvimento deste *patch* produzindo o resultado sonoro que esperávamos, uma paisagem sonora randomicamente controlada pelo *patch*. Sua interação com o *Pure Data Script* também nos rendeu resultados positivos, pois, através dele, conseguimos fazer funcionar várias instâncias deste mesmo *patch,* assim gerando uma paisagem sonora ainda mais complexa.

V.1 *Pure Data* desenvolvimento Protótipos

O desenvolvimento do nosso primeiro *patch* constitui basicamente da tentativa de importar uma imagem para dentro do *Pure Data,* a fim de criar o nosso primeiro *glitch,* utilizando o conhecimento adquirido com os experimentos anteriores. O raciocínio utilizado nesse primeiro *patch* foi que utilizar o objeto [open panel][77] conectado a [open $1] nos permite importar qualquer arquivo selecionado ao se executar um "bang"[78]. Importamos o arquivo selecionado para um objeto do GEM[79] uma extensão do *Pure Data Extended*.

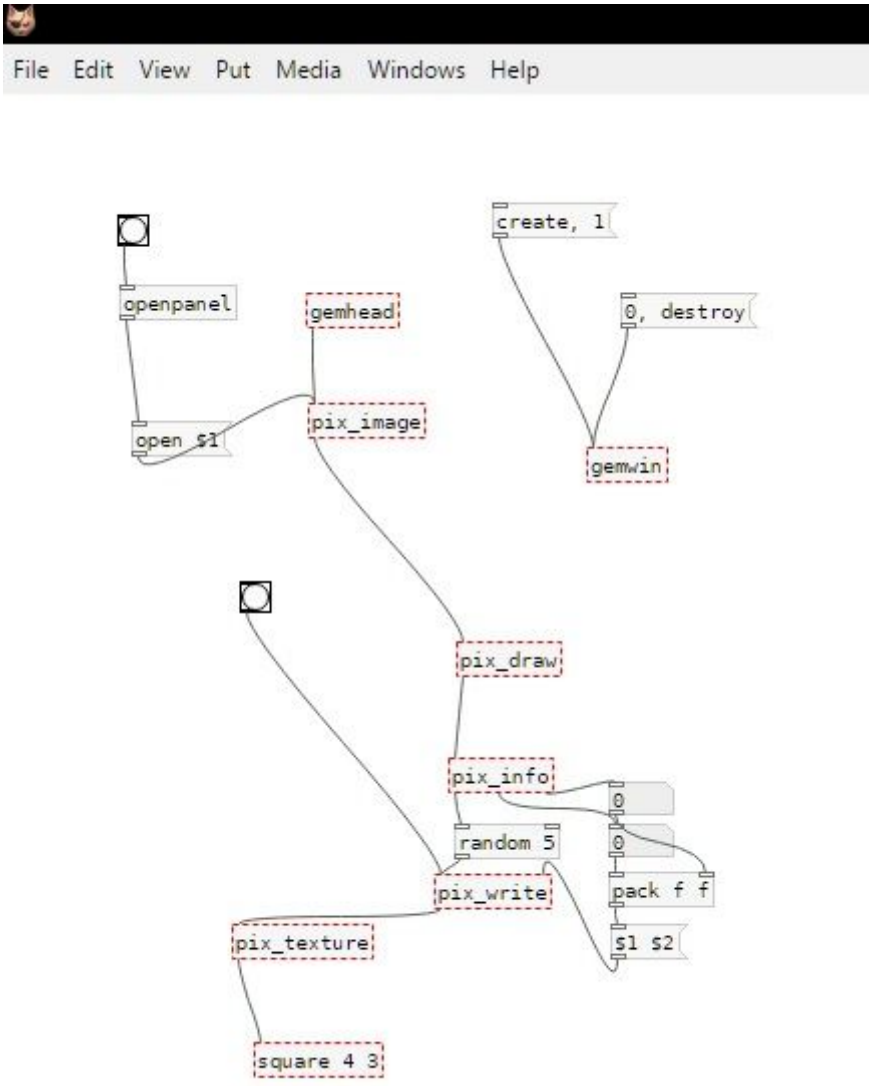
[77]Costumeiramente no fórum do *Pure Data,* ao se referir ao objeto dentro de um *patch* se utiliza colchetes.
[78]Bang uma mensagem do *Pure Data* que manda executar as informações que estão conectadas a ele.
[79] In: < https://puredata.info/downloads/gem/ >. Tradução nossa.

> GEM é o ambiente gráfico para multimídia. Foi originalmente escrito por Mark Danks para gerar gráficos em computação em tempo real, especialmente para composições audiovisuais. Como o GEM é um ambiente de programação visual, os usuários não precisam de nenhuma experiência em linguagens de computadores tradicionais. GEM é uma coleção de externos que permitem ao usuário criar gráficos OpenGL dentro do Pd , um programa para processamento de áudio em tempo real por Miller Puckette (da fama Max). (YE OLDE GEM MANUAL, 2018)[80]

Resumidamente, o GEM é responsável por exibir imagens, uma forma de *output* do *Pure Data* para a informação de forma visual. Em nossa primeira tentativa, a ideia da visualização da imagem importada e do resultado do *glitch* sendo exibidos em uma mesma janela, durante o processo, parecia tentadora. Geramos a janela necessária para se exibir uma imagem com [gemwin]. Os objetos [gemhead] e [pix_image] geram a imagem, efetivamente informamos ao *patch* que queremos que ele a desenhe na janela com [pix draw], e [pix info] é um objeto para se extrair informações da imagem - basicamente ele nos informa tamanho, resolução, cores, algo similar às informações do *header* do arquivo. Esperávamos que, com esse fluxo de dados, inserir o objeto [random 5] que literalmente elege um número randômico de 1 a 5 interferisse na informação a qual ele está conectado. Esperávamos que o [random 5] forçasse sua integral pseudo randômica no processo que o [pix_write] executa, que é retirar a imagem do *frame buffer* e escrevê-la. Com o acréscimo do [pix_texture], ele desenha a imagem no formato que está abaixo dela, no caso [square 4 3] um quadrado quatro por três.

[80]Tradução nossa.

Figura 21 - *Pure data patch* versão 1.



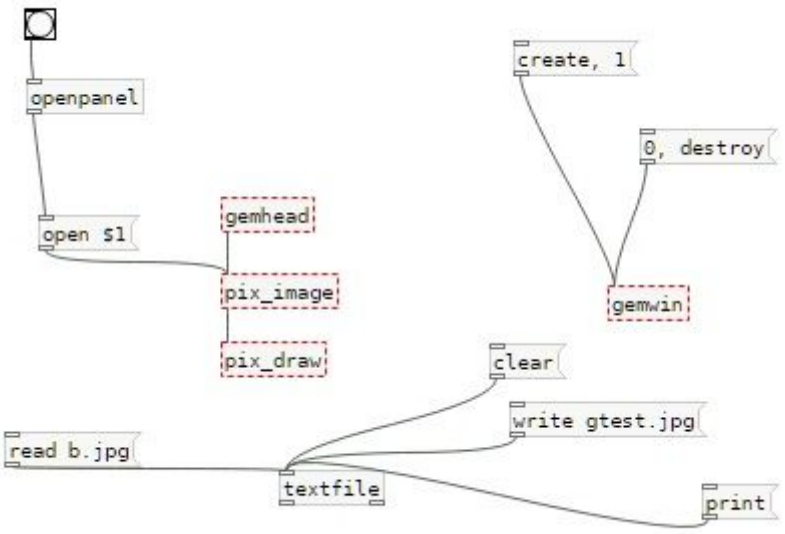
Fonte: Arquivo pessoal (2018)

Os objetos [open $1] e [open panel] são utilizados para importar a imagem para dentro do *patch.* A intenção era que [random 5] fosse o gerador de glitch em nosso patch, atribuindo o valor de zero a cinco randomicamente ao fluxo de dados da imagem entre [pix_info] e [pix_write], assim acrescentando o valor randômico ä exibição da imagem. Os objetos de prefixo PIX nos permitem exibir a imagem importada e manipulá-la, mas a imagem não sofreu nenhuma alteração.

Para um primeiro protótipo, o resultado era satisfatório do ponto de vista do aprendizado da utilização de *Pure Data*, mas falho em relação aos resultados esperados.

Esse primeiro *patch* tem algumas falhas que perceberemos com maior clareza no decorrer do texto, mas um dos principais erros é o tratamento do fluxo de dados. Não há um algoritmo que traduza a imagem em símbolos ou texto, por isso não procedemos de acordo com os experimentos que antecedem o *Pure Data,* uma falha no período da idealização do projeto do *patch*. Seguindo as etapas dos experimentos anteriores à risca, desenvolvemos o segundo *patch:*

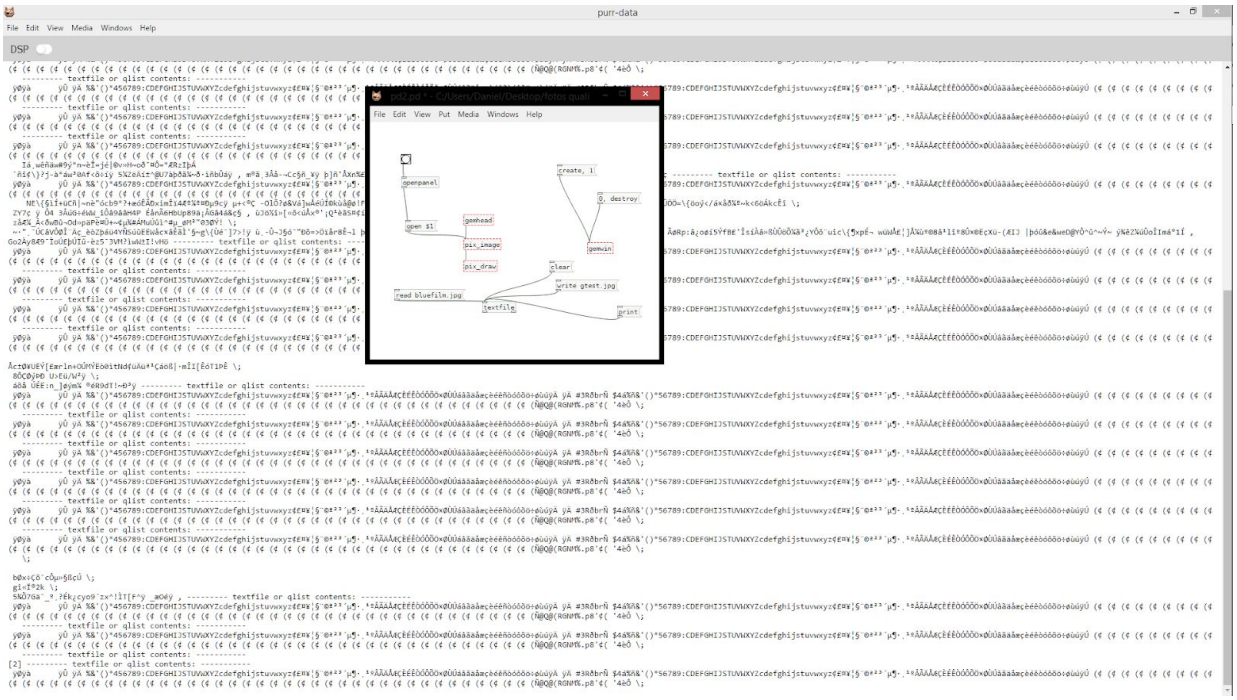
Figura 22 - *Pure data patch* versão 2 novos objetos.



Fonte: Arquivo pessoal (2018)

Com este segundo protótipo, instruímos ao *Pure Data* que ele deve ler o arquivo teste b.jpg com a mensagem [read b.jpg] como um arquivo de texto, utilizando o [textfile]. Depois de lida a imagem em formato de texto, instruímos para o mesmo gerar o arquivo chamado gtest.jpg com o conteúdo do b.jpg traduzido pelo objeto [textfile]. A mensagem [print] serve para vermos o que foi lido pelo [textfile] no terminal do *Pure Data*. Pela primeira vez, podemos ver o código ASCII similar ao que conhecemos dos experimentos anteriores ao *Pure Data,* porém, o arquivo gerado pelo *patch* não pode mais ser aberto como imagem apresentando erro por corrupção do arquivo. A informação do ASCII só ser apresentada no terminal nos incomodava, e percebemos que o ASCII gerado no terminal era muito pequeno para ser todo o conteúdo do arquivo.

Figura 23 - *Pure data* alteração *patch* ASCII no terminal



Fonte: Arquivo pessoal (2018)

A parte construída com GEM serve para o mesmo propósito inicial, apenas abrir a imagem a ser submetida a *glitch* em uma janela à parte, porém, desta vez o GEM não faz parte do fluxo que lê e traduz a imagem. Com esse segundo protótipo em mãos, recorremos a comunidade do *Pure Data* para apontar na direção correta no desenvolvimento do *patch* ou ao menos correções dos erros. Um dos motivos que elencamos trabalhar com *Pure Data* é a sua comunidade ativa, então esse era um recurso que já antecipamos usufruir. A forma mais comum de receber auxílio da comunidade é através do fórum[81] do *Pure Data*. As primeiras questões apresentadas ao fórum sobre acessar o código ASCII de uma arquivo JPEG foram respondidas com links para *patches* que já conhecíamos.

Apesar de nosso *patch* ainda não gerar *glitch,* a mudança de método para trabalhar com o código ASCII, como havíamos aprendido nos experimentos anteriores, em forma de linha de código, parecia mais promissora do que tentar gerar *glitch* no fluxo de dados da geração da imagem dentro do *Pure Data*.

Apresentamos as dificuldades com o segundo *patch* a comunidade do *Pure Data*. As primeiras sugestões eram de fontes que já conhecíamos, mas com uma nova investigação do sitio

[81]In: < https://forum.pdpatchrepo.info/ >.

www.hellocatfood.com - que apresenta alguns trabalhos *glitch,* mas nem sempre os links para download funcionam - conseguimos identificar um *patch* criado em 2012, por Antonio Roberts[82], sob o pseudônimo hellocatfood.

O *patch* vinha com uma mensagem que possuía todo o código binário para gerar um arquivo no formato JPEG. Essa informação do JPEG em formato binário sofria *glitch* pelo mesmo método que havíamos tentado anteriormente: utilizando o objeto [random]. Os diferenciais eram o [random 255], que representa a possibilidade de eleger um número de 0 a 255, valores que um bit pode apresentar, conectado a um objeto de número que identifica para o *patch* que esse valor é um número e não um símbolo ou outra forma de codificação.

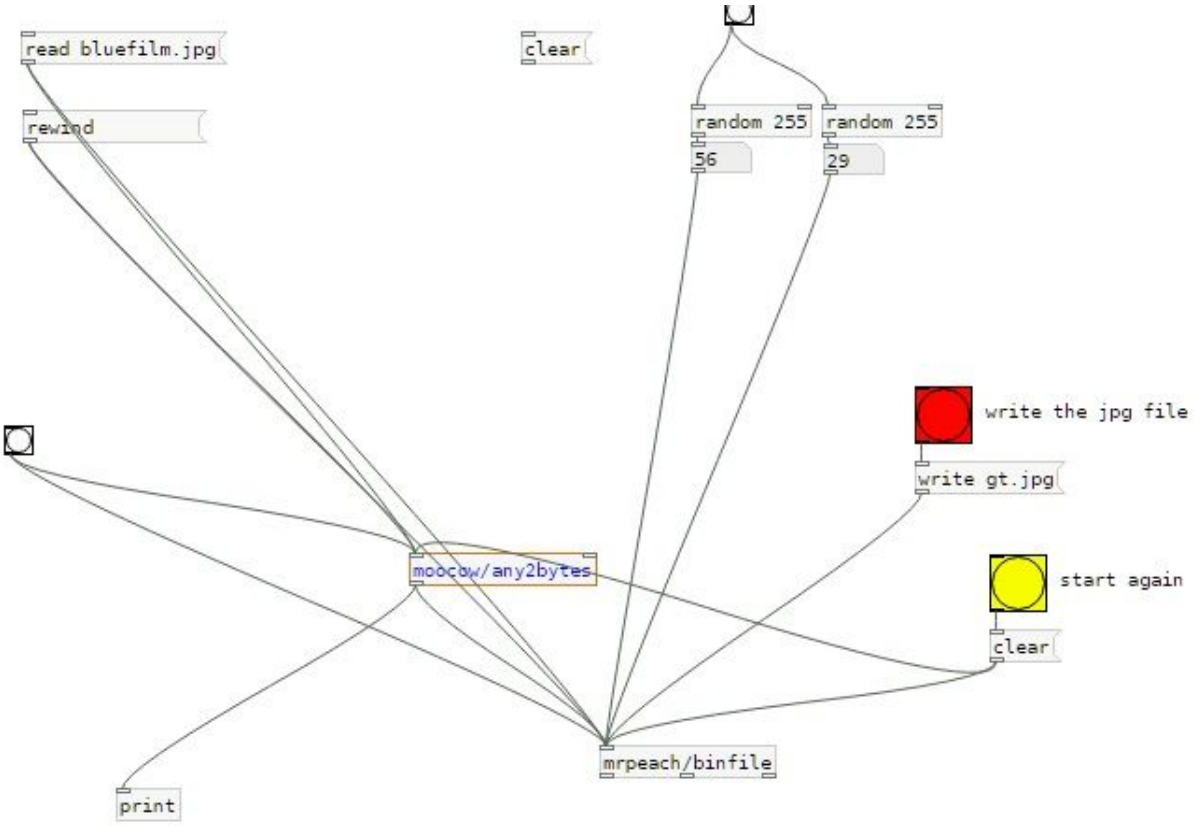
O diferencial mais importante é o objeto [mrpeach/binfile], capaz de entender código binário e gerar um arquivo final a partir dele. Para finalizar, o arquivo é escrito com o objeto [write], exatamente como fazíamos em nosso *patch* anterior. O resultado era um arquivo de JPEG com *glitch*. Encontramos um forte contribuidor para nossa meta final. O diferencial é que não existe forma de importação de um arquivo para esse *patch,* o binário do JPEG que ele utiliza não tinha origem apresentada pelo autor, apesar de ele nos oferecer a opção de manipular o binário manualmente, alterando sem recorrer aos objetos [random]. O método de tradução de uma imagem JPEG para o binário não foi apresentado aqui, e isso é importante para podermos aplicar o *glitch* em qualquer imagem que seja de nosso crivo artístico e não partir de um molde previamente estabelecido.

Para facilitar a compreensão do desenvolvimento da ferramenta, resumimos a descrição do seu desenvolvimento, discorrendo sobre as etapas mais significativas. O processo completo de todas as versões desenvolvidas durante a pesquisa experimental estão no apêndice A desta dissertação.

[82]Antonio Roberts é um artista da New Midia e curador com sede em Birmingham, no Reino Unido. Faz uso processos orientados pela tecnologia para explorar questões relacionadas ao *software* de código aberto, cultura livre e práticas colaborativas. O seu trabalho visual e de performance foi apresentado em galerias e festivais, incluindo databit.me em Arles, França (2012), Glitch Moment / ums em Furtherfield Gallery, Londres (2013), Loud Tate: Código em Tate Britain (2014), glitChicago em O Instituto Ucraniano de Arte Moderna em Chicago, EUA (2014), permissão tomada em Birmingham Open Media e Universidade de Birmingham (2015-2016), Common Property at Jerwood Visual Arts, Londres (2016), Green Man Festival, País de Gales (2017) e Barbican, Londres (2018). Ele organizou exposições e projetos, incluindo GLI.TC/H Birmingham (2011), as edições de Birmingham do Bring Your Own Beamer (2012, 2013), o uChip 3 (2015), Stealth (2015) e a violação de direitos autorais prevista (2017) Ele é Curador em Vivid Projects, Near Now Fellow, e faz parte do Advisory Group for New Art West Midlands. (ROBERTS, 2011, tradução nossa).

Juntamos os resultados da importação do nosso patch com os resultados de glitch e geração de arquivos do Antonio Roberts. Conseguimos importar um arquivo a nossa escolha, traduzi-lo para binário e gerar um novo arquivo sem alterações, mas não conseguimos aplicar o *glitch*.

Figura 24 - *Patch* versão 6 objeto mrpeach/binfile

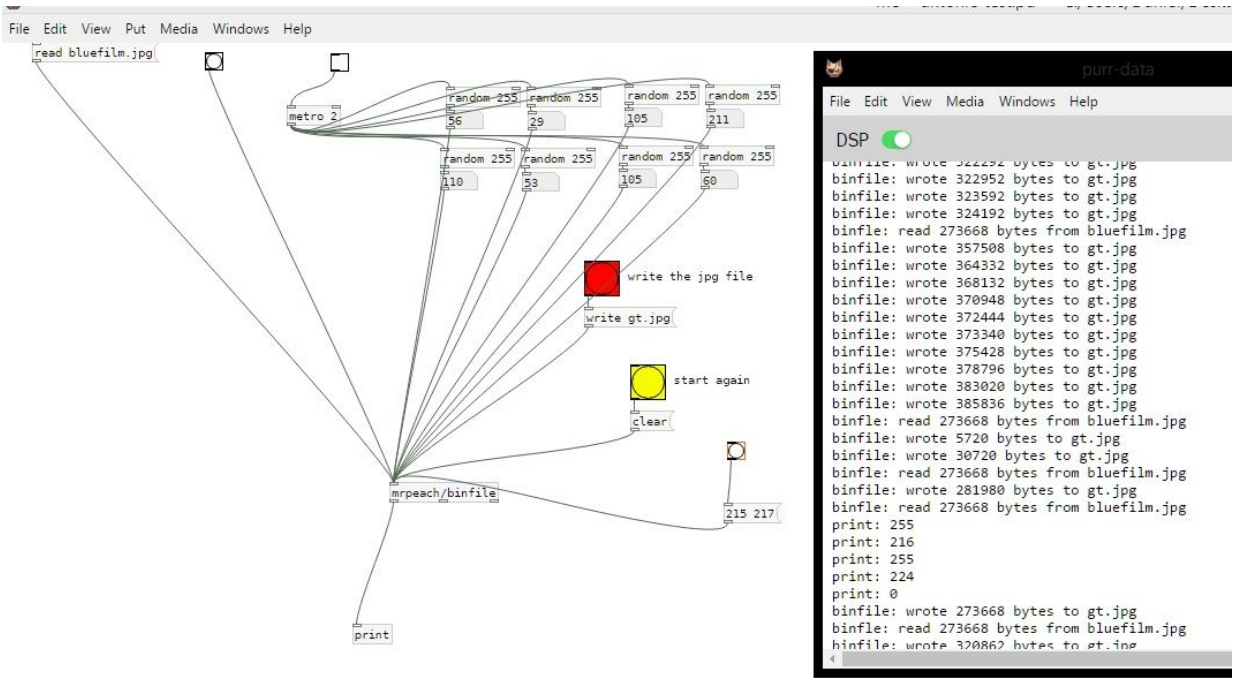


Fonte: Arquivo pessoal (2018)

Por fim, através de alguns testes, verificamos a incompatibilidade entre esses dois objetos, provavelmente por serem dois objetos adicionais ao *Pure Data vanilla* de autores diferentes. Optamos por eliminar o menos citado no fórum, o [moscow/any2byte]. Na imagem 24 percebemos que todos os conectores são recebidos pelo objeto [mrpeach/binfile], e utilizamos o objeto [moscow/any2byte] apenas para introduzir mais uma leitura do fluxo de dados da imagem através da mensagem [read bluefilm.jpg] e exibir o print desses dados. Porém a interação entre [moscow/any2byte] e [mrpeach/binfile] não era possível, o fluxo de dados da imagem não era transmitido de uma para outra e [mrpeach/binfile] tinha todas as funcionalidades que precisávamos.

Procedemos os testes a fim de verificar se era [moscow/any2byte] que prevenia o *glitch*, mas, mesmo com sua eliminação, não obtivemos resultados positivos. Na imagem 25, importamos as imagem para [mrpeach/binfile], e introduzimos oito bits de 0 a 255 elegidos randomicamente com o objeto [random 255] junto com o objeto [metro 2]. Conseguimos recriar a imagem sem *glitch,* com a vantagem de poder observar no console do *Pure Data* que o número de bytes que estavam sendo introduzidos no arquivo estava aumentado, porém, visualmente não houve mudança na imagem.

Figura 25 - *Patch* versão 7, bytes aumentando



Fonte: Arquivo pessoal (2018)

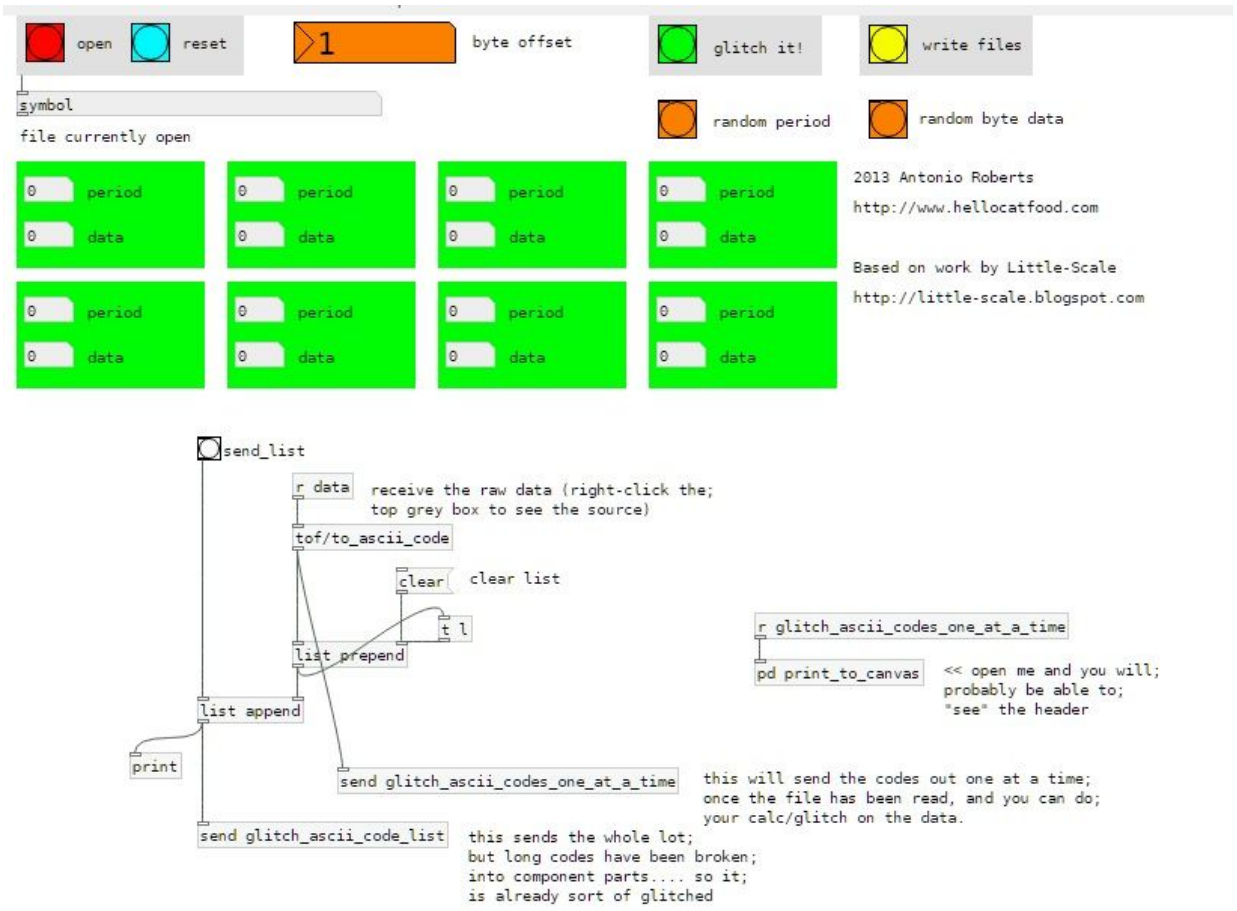
V.2 *Pure Data* Adequando Soluções

O fórum apresentou uma solução, que partia de um *patch* de autoria de Antonio Roberts, que era exatamente o que buscávamos: um *patch* que gerava *glitch* em uma imagem. O usuário whale-av modificou o *patch* original de Roberts a fim de se encaixar nas características do código ASCII que havíamos comentado em postagens anteriores. (Vide apêndice B)

Analisando o *patch* alterado (fig 31), percebemos a complexidade do *patch* original de Roberts, que foi comentado em cada elemento adicionado para mudar a funcionalidade do *patch* original, de trabalhar com binário para trabalhar com ASCII. Essa versão alterada oferece opções como: receber os dados da imagem importada em linhas longas de código ASCII, divididas em partes, que já geram um

glitch, pois alteram a quebra de linha do código original da imagem. Outra opção é emitir os códigos um por vez para que posteriormente possam ser manipulados em um *patch, sub-patch,* ou mesmo fora do Pure Data.

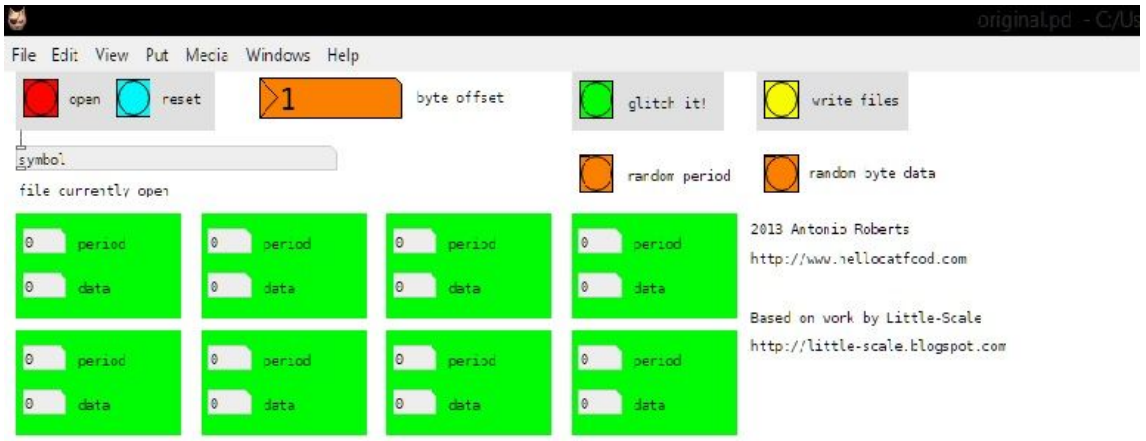
Figura 26 - *Patch* alterado por whale-av



Fonte: Arquivo pessoal (2018)

Apesar de bem intencionada, a solução apresentada não funcionava em nossa máquina utilizada para teste em ambiente Windows 8.1, mas o *patch* que havia sido apresentado era de muita importância para nossa pesquisa, pois já havíamos lido sobre o mesmo, porém o link para download não funcionava. Eis o *patch* original:

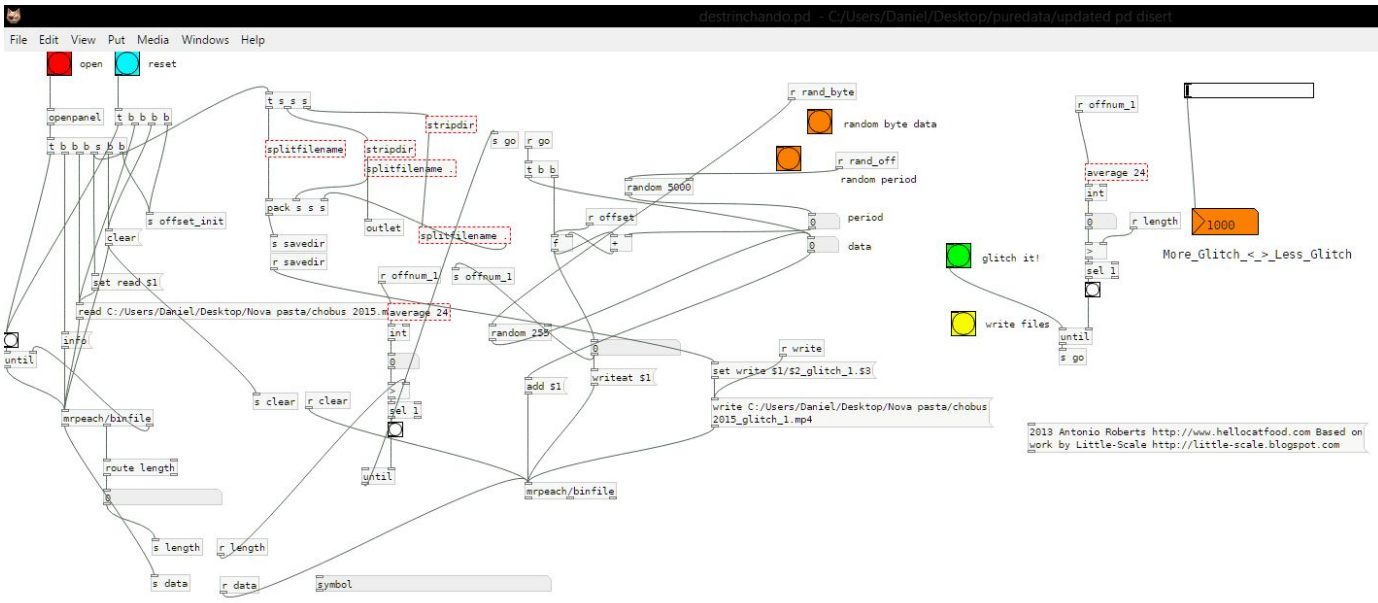
Figura 27 - *Patch* criado por Antonio Roberts interface



Fonte: Arquivo pessoal (2018)

O *patch* de Roberts apresentava uma solução de *glitch* extremamente elegante e diferente da versão alterada feita por whale-av e de nossa ideia original de trabalhar com a criação de *glitch* com ASCII. Ele desenvolveu seu *patch* para trabalhar com código binário e fazia o que a pesquisa intenciona fazer. Buscamos analisá-lo para ganhar conhecimento acerca do tema, da funcionalidade e da interface gráfica criada dentro do próprio *Pure Data,* que ainda não havíamos dedicado tempo para desenvolver, pois as soluções primárias de funcionalidade ainda não haviam sido resolvidas. Começamos a desmontar o *patch* para ver seu funcionamento interno, e principalmente quais objetos haviam sido utilizados na construção do mesmo.

Figura 28 - *Patch* Antonio Roberts análise e desconstrução



Fonte: Arquivo pessoal (2018)

A análise e desconstrução do *patch* mostram que o elemento chave é o objeto [mrpeach/binfile] e, como havíamos chegado a conclusões parecidas em nossos próprios experimentos, esse *patch* confirmava que estávamos em um caminho correto. Esse *patch* também oferece soluções para a implementação do binário gerado com o objeto [random] conectado ao objeto [mrpeach/binfile] através de mensagens add $1 e writeat $1 mensagens chaves para a introdução dos bits e no local indical pela variavel $1 . Mais uma solução foi apresentada de forma simples: a proteção do *header*. Diferente da solução que havíamos idealizado, em que seria necessário identificar o *header* de forma textual ou visual, como vimos antes nos experimentos, a solução implementada por Roberts foi simplesmente utilizar um *offset*, ou seja, deslocar o ponto da interferência que era feita no arquivo com binários, através de um algoritmo simples que basicamente instrui o *patch* a pular os binários do local zero para o local mil, assim preservando o *header* sem necessidade de identificá lo.

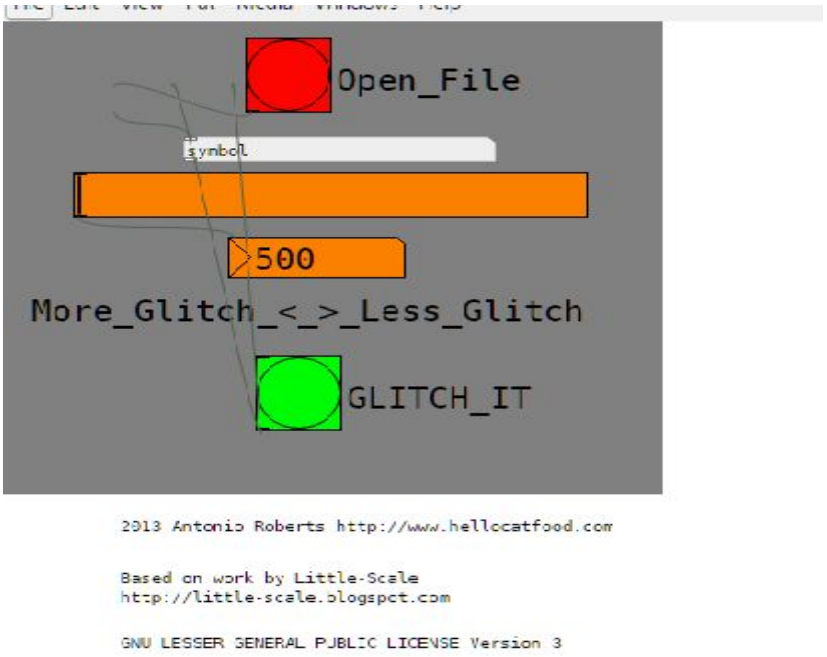
Com a parte interna do *patch* analisada, passamos a analisar a interface gráfica criada utilizando o *Pure Data*. Percebemos que o *patch* do Roberts deixa bem claras as informações necessárias para sua manipulação por um usuário de *Pure Data*, porém, nosso foco é facilitar o processo de criação de *glitch art,* eliminando a necessidade de uma carga técnica elevada. Para isso, assim como temos no próprio *Pure Data* um excesso de elementos que podemos trabalhar, a interface de Roberts acaba

sendo contra intuitiva para gerar um software final, pelo excesso de informação para um usuário que não necessariamente tenha domínio sobre o *Pure Data*. Assim, apresentamos uma solução simplificada da interface gráfica do *patch* de Roberts, acrescido de nossa própria simplificação do processo interno de *glitch.* Agimos de forma reducionista com o *patch* para melhor compreendê-lo e para facilitar a sua implementação e possíveis processos futuros, pois até aquele momento esta era a melhor solução para o problema tema de nossa pesquisa.

Podemos resumir o *patch* de Roberts como uma solução que acrescenta, ao binário do código da imagem, oito interferências em oito pontos eleitos de forma randômica no código e insere oito binários gerados de forma randômica, sendo que esses oito pontos no código respeitam o posicionamento da barra "*byte offset".* O processo de seleção randômica dos locais e bites é feito ao se apertar "*random period" e* "*random byte data"* e o resultado é exibido nas caixas verdes, sendo a próxima etapa apertar "*glitch it",* que escreve as informações que foram selecionadas randomicamente no arquivo, restando apenas exportar o arquivo alterado com "write file".

Abaixo a interface refeita:

Figura 29 - Interface refeita baseado no trabalho de Roberts



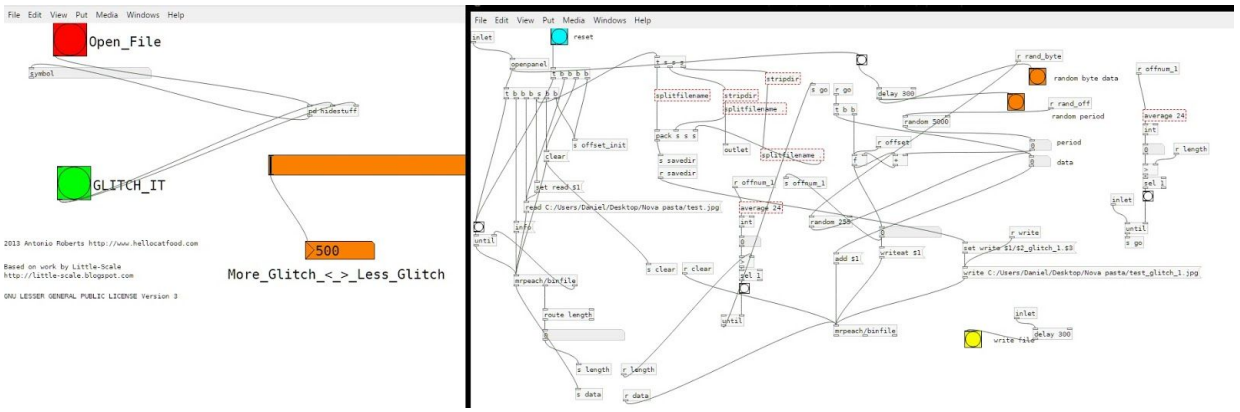
Fonte: Arquivo pessoal (2018)

Esta interface busca esconder elementos não necessários ao usuário final, levando em conta uma possível versão como aplicativo, com funcionamento em um ambiente que não necessariamente seja o *Pure Data*, como celulares, por exemplo, ou uma versão executável de forma independente para o sistema operacional Windows.

Aqui apresentamos a parte interna do *patch* (fig. 35), que é uma versão simplificada do original de Roberts, feita no processo de análise do *patch,* que tinha como uma das principais funções gerar o *glitch* por binário. Vale destacar que, para o nosso desenvolvimento, não havia necessidade de demonstrar, por exemplo, os binários eleitos, mas de ter uma interface com menos elementos visuais e funcionamento mais simples, com menos etapas, pois, diferente do *patch* de Roberts desenvolvido

para usuários de Pure Data, um dos nossos objetivos era diminuir a demanda de conhecimento computacional.

Figura 30 - Funcionamento interno da interface (fig. 34)



Fonte: Arquivo pessoal (2018)

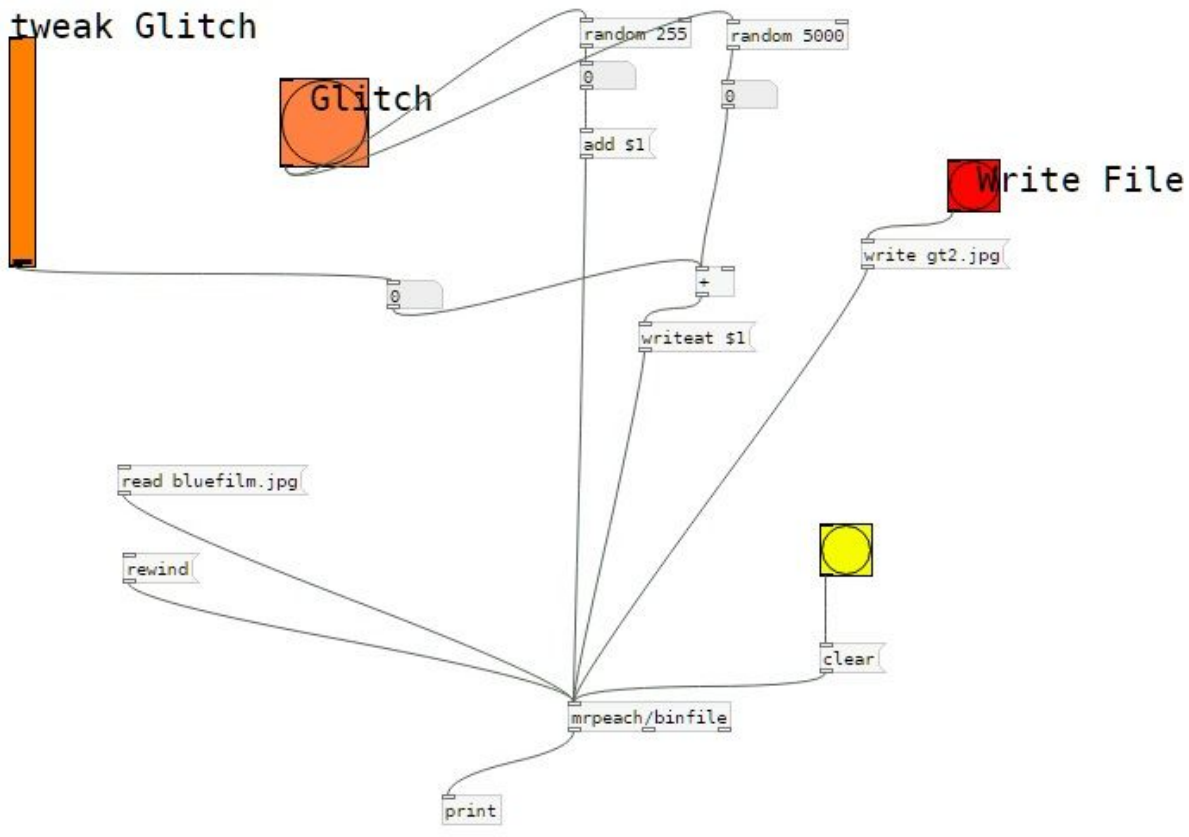
V.3 Pure Data Protótipos 2

Com as informações adquiridas nos experimentos anteriores, fizemos uma nova análise do nosso protótipo mais promissor, a sua sétima versão. Reconstruindo o *patch,* chegamos em sua oitava versão, que tem uma construção eficiente, simples e funcional, apresenta bons resultados ao gerar *glitch* em imagens JPEG com opções de proteção do *header* e com facilidade de utilização. O objeto principal é o [mrpeach/binfile], que faz o trabalho de transcrição da imagem para binário, recebendo a informação de dois objetos [random] - o primeiro gerando números randômicos de 0 a 255 [random 255] e o segundo gerando valores randômicos de 0 a 5000 [random 5000]. O primeiro gera o bit a ser introduzido no código binário original da imagem e o segundo determina a localização. Por exemplo, se o valor 345 é sorteado, o bit previamente sorteado pelo [random 255] altera o bit na localização 345 do binário original da imagem. A proteção do *header* é feita adicionando um valor através de uma barra que permite interação manual do usuário e gera um número de 0 a 5000, de acordo com a altura da manivela dentro da barra que criamos como interface. Esse valor é adicionado ao valor do [random 5000], assim permitindo ao usuário controlar a localização[83] do *glitch* a ser feita na imagem. Se selecionado o valor zero, deixando a barra em sua extremidade mais baixa, o *header* será alterado. O

[83]A localização do *glitch* no código binário pode ou não ter uma relação direta com o local do *glitch* na imagem (linha de pixels no quadrante X e Y). Isto varia com cada arquivo de imagem importado para o *patch*.

botão de *Glitch* pode ser utilizado mais de uma vez, permitindo ao usuário apertá-lo quantas vezes desejar, introduzindo mais bits randômicos em locais randômicos até o eventual não funcionamento da imagem por excesso de bits aleatórios. Este *patch* também permite que cada vez que o botão *Glitch* seja apertado, a barra de seleção de *tweak Glitch* seja posicionada de diferentes formas, assim o usuário pode adicionar mais erros quando ele aperta o botão *Glitch.* Ao clicar o botão vermelho *Write file,* o usuário gera o arquivo com os bits randômicos introduzidos em seu conteúdo. A mensagem *clear* e o objeto [print] foram utilizados para *debugging*[84], e a mensagem *rewind* faz papel importante no sistema, pois instrui o *patch* a voltar para o ponto inicial do *buffer* criado na leitura e tradução da imagem, podendo assim, introduzir os bits randômicos e adicionar os valores para proteção do *header* criados com a barra *tweak Glitch* do início do *buffer*.

Figura 31 - *Patch* versão 8 apresentada na qualificação



Fonte: Arquivo pessoal (2018)

[84]Correções de erros do *patch*, *debugging* ou o ato de remover *bugs* é um termo comum ao se referir a correção de erros em um programa dentro da computação.

Com a funcionalidade do *patch* verificada, passamos a utilizá-lo para compor algumas obras de *glitch art* como um teste final, utilizamos as obras do artista plástico Thiago Trapo que foram doadas para a pesquisa como base de nosso trabalho, abaixo alguns resultados:

O que podemos observar nesses testes é que os arquivos com resolução de acima de FHD[85] tendem a ser mais resilientes ao *glitch* do *Pure Data.* A hipótese que sugerimos é que a alteração é minúscula comparada a todo o conteúdo de bits dentro de um arquivo desse tamanho, sendo assim, o atual *patch* funciona melhor com imagens de resolução menor ou igual a FHD. Também percebemos a importância da compressão dos arquivos, já que, quanto mais comprimido era o arquivo, maior era o efeito do *glitch* na imagem final. Supomos que a maior interdependência dos *bits,* criada pela compressão dos arquivos, faça com que a mudança de um *bit* altere um número maior de informações relacionadas a ele, apresentando resultados mais visíveis na imagem. Outra hipótese é que a menor quantidade de *bits* dentro do arquivo, devido a compressão, faça com que a alteração de poucos *bits* seja mais significante na corrupção do arquivo. Para mudanças de resolução das imagens originais e compressão dos arquivos, utilizamos o *software* GIMP[86]. Com este resultado, chegamos ao objetivo que pretendíamos no início da pesquisa.

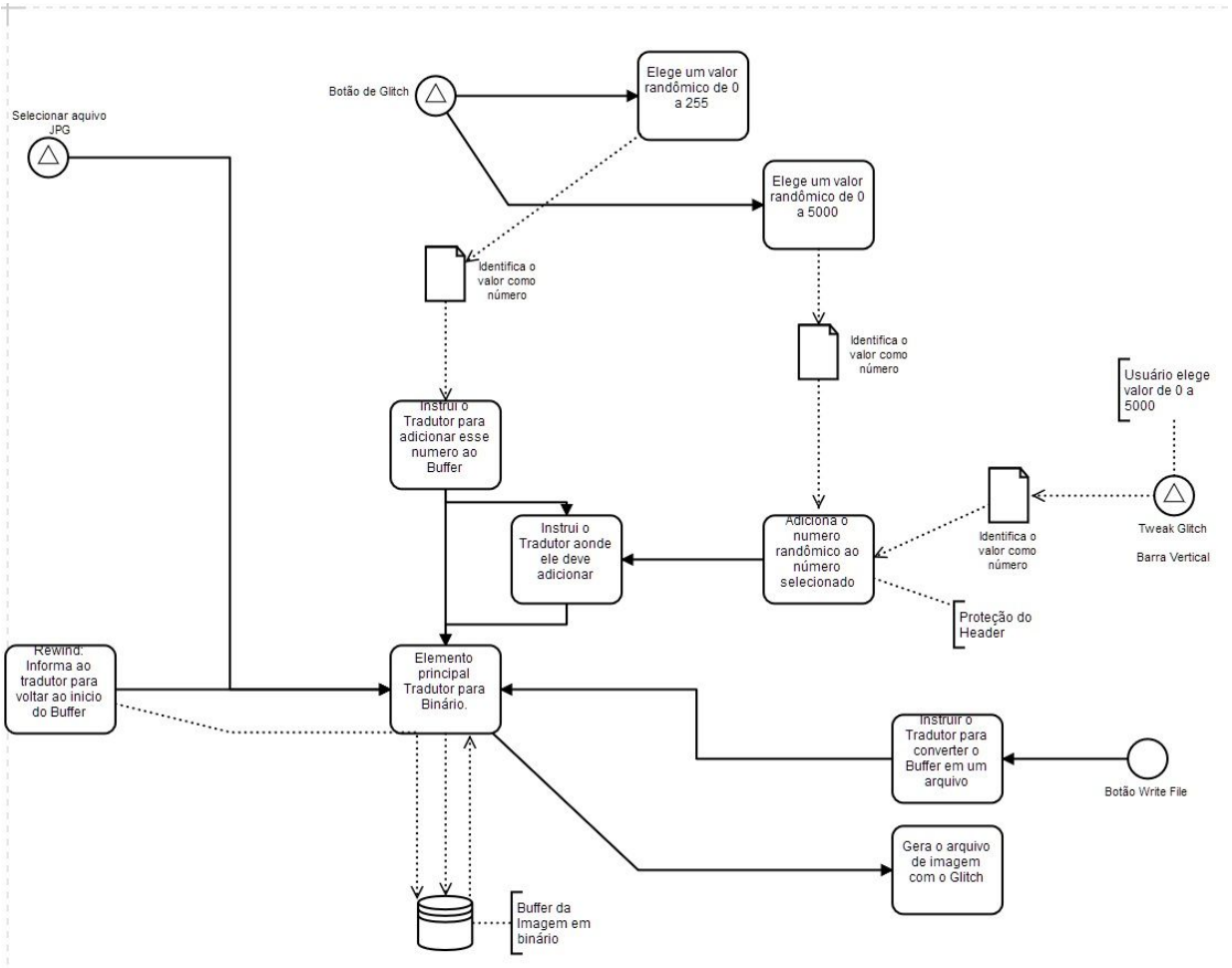
No segundo teste com o mesmo *patch,* importamos uma imagem em PNG[87] e exportamos a informação com o *glitch* em JPEG, uma corrupção adicional no formato do arquivo, além da introdução de bits.

Finalizamos a obra como um GIF[88], unindo a obra original com o nosso *glitch,* demonstrando mais uma possibilidade dentro do processo criativo. Esta obra faz uso de três formatos de imagem diferentes, de forma subversiva, em seu processo de criação da *glitch art*.

Com esses resultados passamos agora a dar atenção a outros aspectos do *patch*. Abaixo um diagrama dos algoritmos do *patch* desenvolvido:

Figura 32 - Diagrama

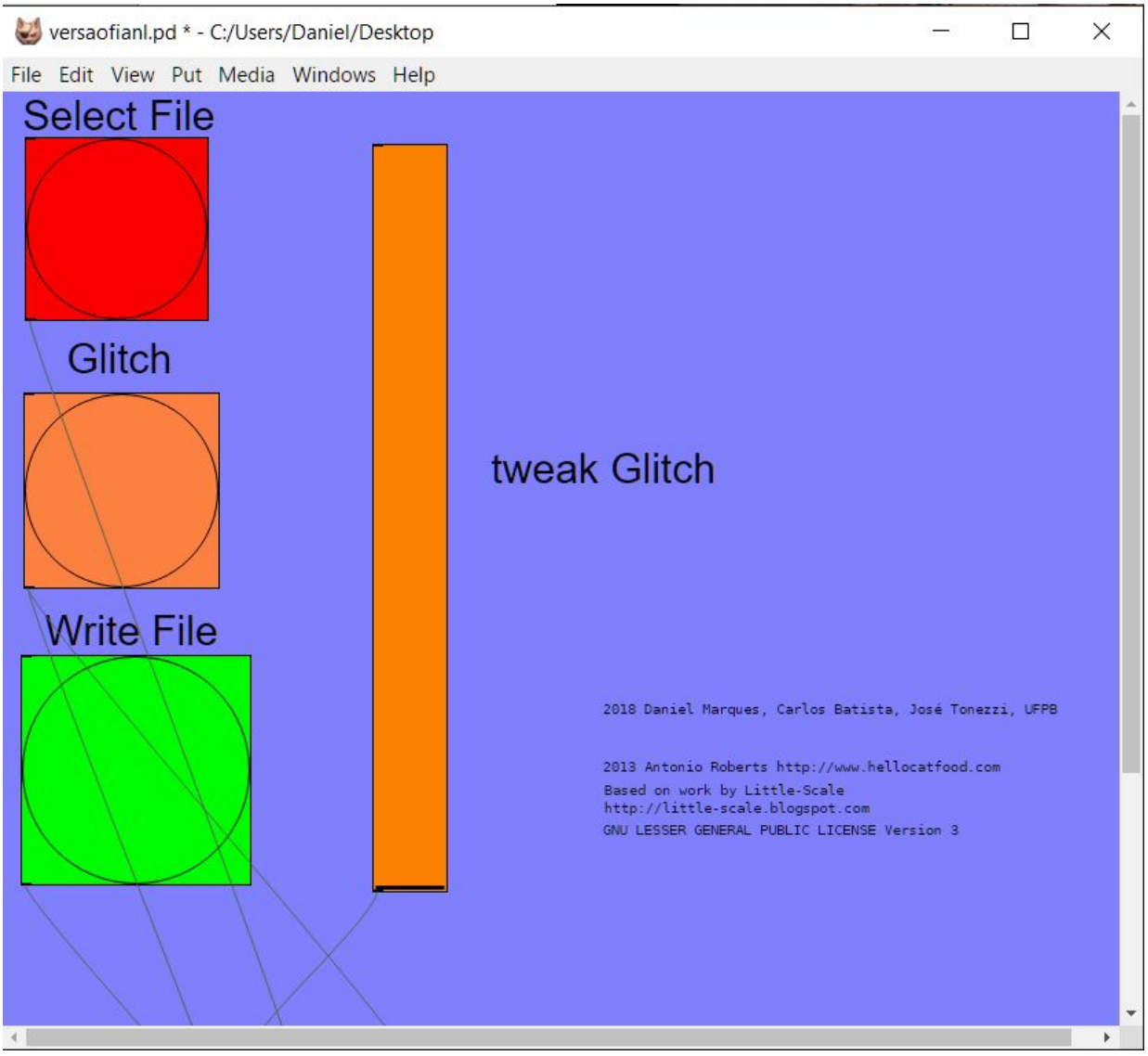
[85]Full High Definition (FHD) é a resolução de 1920x1080 pixels
[86]GNU Image Manipulation Program, um editor de imagem de licença pública https://www.gimp.org/about/
[87]PNG *Portable Network Graphics* um formato de imagem comumente utilizado na internet.
[88]Graphics Interchange Format, um formato de arquivos de imagem que permite animações simples.



Fonte: Arquivo pessoal (2018)

A versão final do *patch* possui o mesmo método de funcionamento da versão 8, assim como explicitado no diagrama acima, além da interface melhorada, demonstrada na figura 42:

Figura 33 - Versão final com nova interface.



Fonte: arquivo pessoal (2018)

VI. CONCLUSÃO

Durante o processo de pesquisa para elaboração do trabalho que apresentamos como dissertação, percorremos com análise experimental e teórica as práticas acerca da *glitch art*, discorremos sobre a cultura em seu entorno e sua abordagem prática, verificamos variados métodos de desenvolvimento da *glitch art* descritos na literatura*,* tanto em *glitches* reais *como glitch like,* e trazemos isso para a dissertação, como um manual acadêmico de suas possibilidades criativas artísticas e de seu funcionamento perante aspectos técnicos, sejam eles técnicas de edição ou de programação em Pure Data de código aberto, trazendo assim abertamente a sua forma de funcionamento que como vimos antes é importante para a *glitch art*.

Com esse conhecimento desenvolvemos todos os nossos objetivos: criamos a ferramenta geradora de *glitch* em imagens JPEG utilizando a linguagem de programação visual Pure Data; transitamos entre metodologias e experimentos das áreas da computação e da arte e expomos um guia acadêmico para desenvolvimento, tanto de *patches* para Pure Data quanto de *glitch art* em alguns formatos de mídias digitais, sendo essa abrangência permitida pela própria sinestesia que a digitalização das mídias nos oferece. Também durante o decorrer da pesquisa percebemos possíveis relações entre *glitch art* e o grotesco. Concluímos avaliando os resultados de forma positiva e almejando, em futuras pesquisas desenvolver novas versões aprimoradas de alguns dos *patches* apresentados, e buscar ainda novas interações com a arte computacional. Glitch art por sua natureza é subversiva: desconstruir, interferir, repensar e revisualizar são caracteristicas de seu movimento artístico, fazendo isso de forma invasiva dentro dos meios digitais, corrompendo, alterando, desmistificando áreas que antes eram opções binárias, como estados funcional ou não funcional. A glitch art traz esse processo ä tona na obra finalizada, ou sequer finaliza a obra, deixando a máquina, os erros, contribuírem para as obras, trazendo a dualidade em tudo que lhe engloba, funcional e não, distorcido e puro, caótico e ordenado.

VII.1 Trabalhos futuros

Como futuros experimentos e pesquisas pretendemos fazer o desenvolvimento de versão do *patch* gerador de *glitch* como aplicativo para dispositivos móveis, como IOS e Android, utilizando o libpd, além de uma possível versão *stand-alone*[89] para Windows, Linux e Mac-OS, algo que seria possível por meio de uma tradução do *patch* de Pure Data para MAX/MSP[90], ou ainda uma versão do *patch* apresentada em formato de uma página da internet.

Outra possibilidade é a conversão do *patch* de *glitch* para um plug-in[91], assim podendo ser incorporado como ferramenta de edição de forma mais compatível com o *workflow* de profissionais que trabalham com edição de áudio, vídeo e imagens. A versão do plug-in para VST[92], a fim de ser utilizada em DAWS como ferramenta de edição de áudio, poderia ser feita através do projeto Camomile[93] uma ferramenta que permite incorporar *patches* como plug-ins VST .

Além da possibilidade de trabalhar com a geração de *glitch* em tempo real que não foi explorada nesta pesquisa, porém traz um elemento interessantíssimo para performances que envolvam projeção, captura ou transmissão de vídeos e imagens, e também para eventos de música, como um efeito para instrumentos ou DJs executando suas performances ao vivo.

Todos os *patches* desenvolvidos durante esta dissertação serão apresentados tanto a comunidade do Pure Data, através de uma publicação no fórum oficial demonstrando os resultados obtidos, como para a comunidade de artistas locais interessados, com o objetivo de difundir tanto o uso da ferramenta, como a utilização de Pure Data. Também a partir desta dissertação podemos apresentar palestras tanto do uso do Pure Data quanto do criação de *glitch art* para artista locais. Como demonstrado, o *patch* em Pure Data abre uma porta para uma imensidade de novas interações que ainda podem ser exploradas por sua ativa comunidade de desenvolvedores

[89] Uma operação computacional que funciona de forma independente, no caso o *patch* funcionaria sem ser necessário o Pure Data.

[90] In: < https://cycling74.com/ >

[91] Elemento computacional que adiciona uma função nova a um programa.

[92] Tecnologia de estúdio virtual

[93] In: < https://github.com/pierreguillot/Camomile/wiki >

VIII. REFERÊNCIAS:

BAKHTIN, Mikhail. **A Cultura Popular na Idade Média e no Renascimento::** O Contexto de François Rabelais. Brasília: Editora Universidade de Brasília, 1987. 385 p.

BAX, Marcello. XV ENANCIB 'ALÉM DAS NUVENS: EXPANDINDO AS FRONTEIRAS DA CIÊNCIA DA INFORMAÇÃO', 15., 2014, Belo Horizonte. **DESIGN SCIENCE:** FILOSOFIA DA PESQUISA EM CIÊNCIA DA INFORMAÇÃO E TECNOLOGIA. Belo Horizonte: Encontro Nacional de Pesquisa em Ciência da Informação, 2014. 3903 p.

BRECHT, George. **Chance Imagery.** New York: Great Bear Pamphlet By Something Else Press, 1966. 30 p.

BRINKMANN, Peter et al. **Embedding pure data with libpd**. In: Proceedings of the Pure Data Convention. 2011.

BRIZ, Nick et al (Ed.)**. GLI.TC/H READER[ROR] 20111.** Amsterdam: Unsorted Books, 2011. 161 p. Disponível em: <HTTP://GLI.TC/H/READERROR>. Acesso em: 6 ago. 2016.

COUCHOT, Edmond. **A tecnologia na arte:** da fotografia à realidade virtual. Tradução Sandra Rey. Porto Alegre: Editora da UFRGS, 2003.

CONTRETAS, Scott; MIROCHA, Lukasz. **THE NEW AESTHETIC AND ART:** CONSTELLATIONS OF THE POSTDIGITAL. Amsterdam: Institute Of Network Cultures, 2016. 280 p. Disponível em: <http://www.networkcultures.org/publications/#tods>. Acesso em: 7 fev. 2017.

Chiptune. Dísponivel em: https://pt.wikipedia.org/wiki/Chiptune Acessado em: 22 abr. 2016

DANKS, Mark. **Real-time Image and Video Processing in GEM**. PROCEEDINGS ICMC97, fUniversity Of California, San Diego, p. 1-3, jan. 1997.

Delay GSI WatKat Dísponivel em: http://www.genuinesoundware.com/?a=downloads&b=36 Acessado em 8 de junho 2016

DRYMONITIS, Alexandros. **Drymonitis**: Bio. Disponível em: <http://drymonitis.me/bio/>.Acesso em: 22 fev. 2018.

FERNANDES, José Carlos Silvestre. **A ESTÉTICA DO ERRO DIGITAL.** 2010. 154 f. Dissertação (Mestrado) - Curso de Mestrado em Tecnologias da Inteligência e Design Digital, Pontifícia Universidade Católica de São Paulo, São Paulo, 2010.

GAZANA, Cleber. **GLITCHART: :ESTÉTICADOERRODIGITAL**. 2015. GLITCHART:ESTÉTICADOERRODIGITALCleber Gazana/ PPGA –Uni versidade Estadual PaulistaComitê de Poéticas Artísticas. Disponível em: <chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/http://anpap.org.br/anais/2015/comites/cpa/cleber_gazana.pdf>. Acesso em: 16 nov. 2018.

GAZANA, Cleber. **Glitch:: A arte visual do erro digital**. 2014. Poéticas Visuais , Bauru, v 5 n. 1 p.67-82. Disponível em: <https://www.academia.edu/30463590/Glitch_a_arte_visual_do_erro_digital_Glitch_the_visual_art_of_the_digital_error>. Acesso em: 20 nov. 2018.

Glitch: a noisy fusion of math and music. 2014. Disponivel em: <https://medium.com/@naive_sound/glitch-a-noisy-fusion-of-math-and-music-6a9b24e7f5b5#.vyycpmt7p> Acessado em 20 abril 2016.

Glitch: a noisy fusion of math and music. 2014. Disponivel em: < http://naivesound.com/glitch/#> Acessado em 20 abril 2016.

GLITCH. in: Cambridge dictionaries online, 2016. Disponivel em: <http://dictionary.cambridge.org/pt/dicionario/ingles/glitch?q=Glitch>. Acessado em 14 de out. 2016.

GROTESCO . In: ENCICLOPÉDIA Itaú Cultural de Arte e Cultura Brasileiras. São Paulo: Itaú Cultural, 2019. Disponível em: <http://enciclopedia.itaucultural.org.br/termo4981/grotesco>. Acesso em: 05 de Fev. 2019. Verbete da Enciclopédia.

ISBN: 978-85-7979-060-7

HELLOCATFOOD Disponível em: < https://www.hellocatfood.com/pure-data-file-killer/> Acessado em: 26 fev. 2018

HOLMES, Danny. **Mobile Instruments Made Easy:** Creating Musical Mobile Apps with LIBPD and iOS, No Experience Necessary. Proceedings ICMC|SMC|2014, Athens, Greece, p. 1-1803, set. 2014.

Illformed.org Disponível em: http://illformed.org/plugins/ Acessado em 20 de maio 2016.

JOHNSON, Steven. **Cultura da Interface:** Como o Computador Transforma nossa Maneira de Criar e Comunicar. Rio de Janeiro: Jorge Zahar Editor, 2001.

LEHTOMÄEN, Sakarias **Countercomplex:** Some deep analysis of one-line music programs. 2011. Disponível em: http://countercomplex.blogspot.com.br/2011/10/some-deep-analysis-of-one-line-music.html acessado em 18 abril 2016.

LÉVY, Pierre. **CIBERCULTURA.** São Paulo: Editora 34, 1999. 264 p.

LEMOS, André. Arte Eletrônica e Cibercultura. **Famecos**: TECNOLOGIAS DO IMAGINÄRIO, Porto Alegre, n. 6, p.1-31, jun. 1997. Semestral.

LØFGREN, Tobias. Glitch Art: - Adventures in databending. 2018. Disponível em: <https://tobloef.com/fun/glitch-art>. Acesso em: 21 dez. 2018.

MANOVICH, L. **The language of new media**. Cambridge: MIT Press, 2001.

MANON, Hugh S.; TEMKIN, Daniel. Notes on Glitch. 2011. Daniel Temkin makesstilland interactivepieces,often consistingof uneasy collaborations with the computer.His work has been exhibited in museums and galleries inNorth America and Europe, and at glitch-related festivals such as Bent Fest. He has presented at such conferences as GLI.TC/H, Rewire (Media Art Histories)andHackers on Planet Earth, and served as artist-in-residence in Budapest and in Southern Italy.He is currently an MFA Candidate at the International Center of Photography. Disponível em: <chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/http://www.worldpicturejournal.com/WP_6/PDFs/Manon.pdf>. Acesso em: 26 dez. 2018.

MANGPER. **Opened a JPG picture with notepad, pasted all the “text” to a new notepad file, changed to .JPG and it no longer opens. Why?** 2014. Disponível em:

<https://superuser.com/questions/782692/opened-a-jpg-picture-with-notepad-pasted-all-the-text-to-a-new-notepad-file>. Acesso em: 20 jun. 2018.

Making music with a C compiler [SIGINT13] Chaos Communication Congress. 2013. Disponível em:https://media.ccc.de/v/saal_mp7_og_-_2013-07-06_15:00_-_making_music_with_a_c_compiler_-_erlehmann_-_5067 Acesso em 22 out 2017.

PLAZA, Julio. **Arte e Interatividade**: autor – obra – recepção. Concinnitas, Rio de Janeiro, ano 4, n. 4, p. 7-34, mar. 2003.

MCCORMACK, Tom et al. Code Eroded: : At GLI.TC/H 2010, RHIZOME. In: BRIZ, Nick et al. **GLI.TC/H READER[ROR] 20111**. [S.l.: s.n.], 2011. p. 15-20. Disponível em: <http://HTTP://GLI.TC/H/READERROR>. Acesso em: 12 mar. 2018.

MENKMAN, Rosa. **The Glitch Moment(um).** Amsterdam: Institute Of Network Cultures, 2011. 70 p. Disponível em: <http://www.networkcultures.org/networknotebooks>. Acesso em: 6 ago. 2016.

MENKMAN, Rosa. **GLITCH STUDIES MANIFESTO**. 2009. /As presented / performed at- Blip Festival, New York, US. 18–12'09.I performed parts of the manifesto during my Little-Scale visual set. Disponível em: <chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://amodern.net/wp-content/uploads/2016/05/2010_Original_Rosa-Menkman-Glitch-Studies-Manifesto.pdf>. Acesso em: 26 dez. 2018.

MORADI, Iman. **Glitch aesthetics.** 2004. 85 f. Dissertação (Mestrado) - Curso de Multimedia Design, Department Of Architecture, University Of Huddersfield, Huddersfield UK, 2004. Disponível em: <http://www.organised.info/wp-content/uploads/2016/08/Moradi-Iman-2004-Glitch-Aesthetics.pdf>. Acesso em: 14 out. 2016.

NELSON, David Erik. **Circuit Bending For Beginners.** 2015. Disponível em: <http://performermag.com/best-instruments/circuit-bending-for-beginners/>. Acesso em: 02 ago. 2018.

OSWALD, John. Plunderphonics, or Audio Piracy as a Compositional Prerogative. In: WIRED SOCIETY ELECTRO-ACOUSTIC CONFERENCE, 34., 1985, Toronto. **Musicworks #34.** Toronto: Recommended Quarterly, 1985. p. 1 - 100. Disponível em: <www.plunderphonics.com/xhtml/xplunder.html>. Acesso em: 10 set. 2017.

OZCITAK, Ozgur . **ExifLibrary for .NET** ,2009. Disponível em: <https://www.codeproject.com/Articles/43665/ExifLibrary-for-NET>. Acesso em: 22 mar. 2018.

PUCKETTE, Miller et al. **Pure Data: another integrated computer music environment**. Proceedings of the second intercollege computer music concerts, p. 37-41, 1996.
PUCKETTE, S. Miller (1997). Pure Data: Recent Progress. Keio University. Reprinted from Proceedings, Third Intercollege Computer Music Festival, Tokyo, pp 1-4.

PURE Data. Disponível em: <https://puredata.info/>. Acesso em: 19 jul. 2016.

PROCEEEDINGS OF THE 21ST INTERNACIONAL SYMPOSIUM ON ELETRONIC ART, 21., 2015, São Paulo. **Aesthetics of the Digital Ruins and the Future of Art Conservation:** Giselle

Beiguelman. São Paulo: School Of Architecture And Urbanism (design Course) – University Of São Paulo, 2015. 6 p.

PRODANOV, Cleber Cristiano. Metodologia do trabalho científico [recurso eletrônico] : métodos e técnicas da pesquisa e do trabalho acadêmico / Cleber Cristiano

Prodanov, Ernani Cesar de Freitas. – 2. ed. – Novo Hamburgo: Feevale, 2013.

ROBERTS, Antonio. **Create jpgs in Pure Data**: In search of a jpg header. 2012. Disponível em: <http://www.hellocatfood.com/create-jpgs-in-pure-data/>. Acesso em: 22 fev. 2018.

ROY, Mallika. GLITCH IT GOOD:: UNDERSTANDING THE GLITCH ART MOVEMENT. 2014. Disponível em: <http://www.theperipherymag.com/on-the-arts-glitch-it-good/>. Acesso em: 26 dez. 2018.

SAKR, Laila Shereen; SAKR, Laila Shereen. **Glitch Art Resources.** 2016. Disponível em: <https://famst109ga.wordpress.com/>. Acesso em: 1 fev. 2017.

SCHULZE, Guilherme Barbosa. **Videodança como prática colaborativa**. 2011. 4 p. Videodança como prática colaborativa (VI Reunião Científica do ABRACE)- POrto, Universidade Federal da Paraíba, Porto Alegre, 2011. Disponível em: <https://www.google.com.br/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwiOlbmGydzZAhXMh5AKHRXQAU0QFggnMAA&url=http%3A%2F%2Fwww.portalabrace.org%2Fvireuniao%2Fpesquisadanca%2F8.%2520SCHULZE%2C%2520Guilherme%2520Barbosa.pdf&usg=AOvVaw0YCssaXaNreiBAUFER0cqH>. Acesso em: 02 fev. 2018.

STALLIO!. **Databending and glitch art primer, part 1:** the wordpad effect. 2008. Disponível em: <http://blog.animalswithinanimals.com/2008/08/databending-and-glitch-art-primer-part.html>. Acesso em: 1 fev. 2017.

TOMCZAK, S. (2009). **Authenticity and emulation:** Chiptune in the early twenty-first century. In *International computer music conference proceedings*. 2009. Disponivel em: http://milkcrate.com.au/_other/downloads/writing_stuff/tomczak.icmc2008.pdf acessado em 6 de maio 2016.

TONEZZI, José. **A cena contaminada:** um teatro das disfunções. São Paulo: Editora Perspectiva S.a, 2011.

TEMKIN, Daniel. **Glitch && Human/Computer Interaction.** 2014. Disponível em: <http://nooart.org/post/73353953758/temkin-glitchhumancomputerinteraction>. Acesso em: 20 fev. 2018.

VENTURELLI, Suzete. ESTÉTICA E ARTE COMPUTACIONAL. Artefactum: – REVISTA DE ESTUDOS EM LINGUAGEM E TECNOLOGIA, Belo Horizonte, v. 3, n. 1, p.1-74, fev. 2010. Anual.

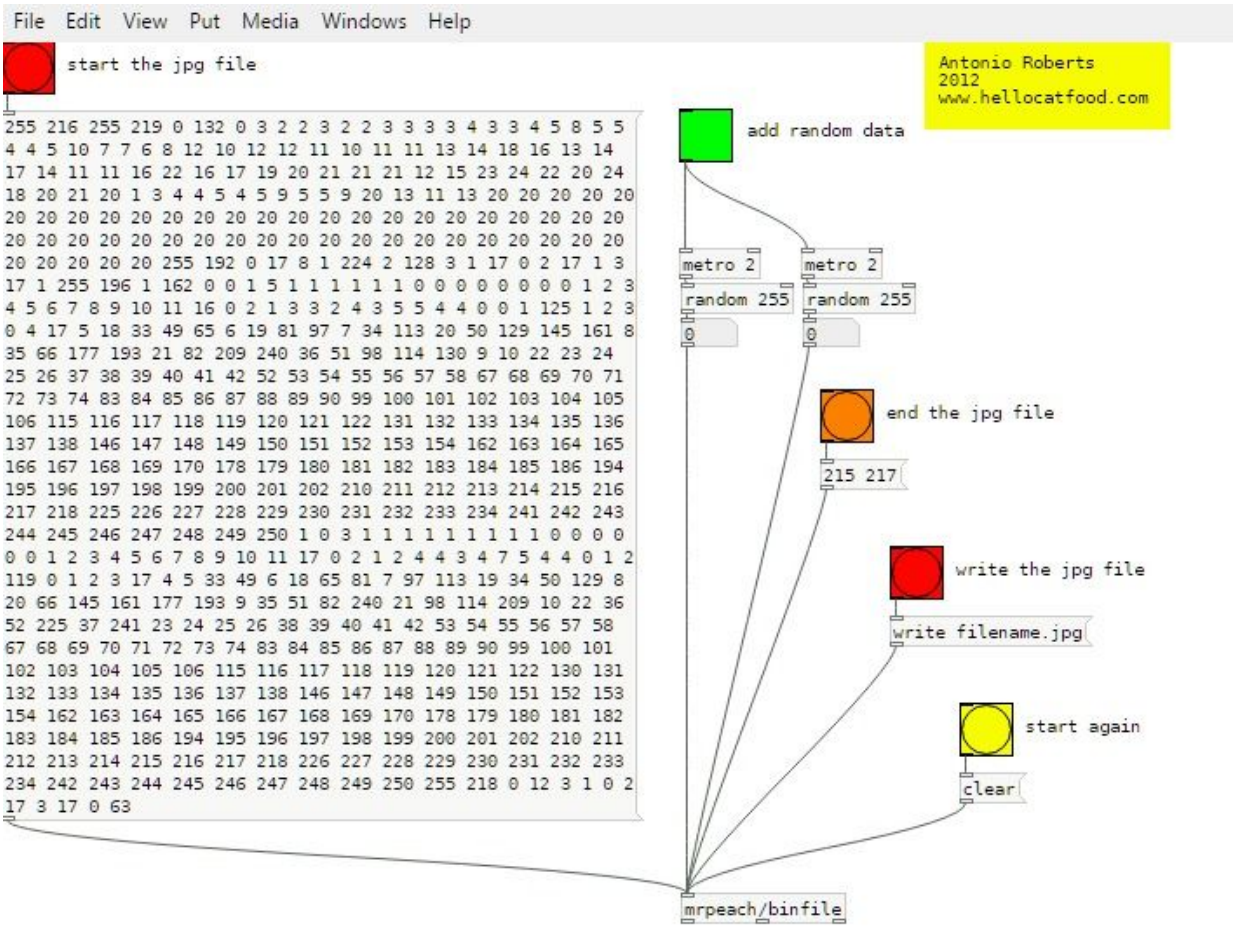
WILSON, Stephen. **Information arts**: intersections of art, science, and technology. MIT press, 2002.

YE olde Gem Manual . Disponível em: <https://puredata.info/downloads/gem/documentation/manual/manual>. Acesso em: 21 fev. 2018.

APÊNDICE A - Detalhamento do desenvolvimento do *patch*

Aqui relataremos o detalhamento de todas as versões do *patch* desenvolvido para a pesquisa.

Figura 1 - *Patch* Antonio Roberts

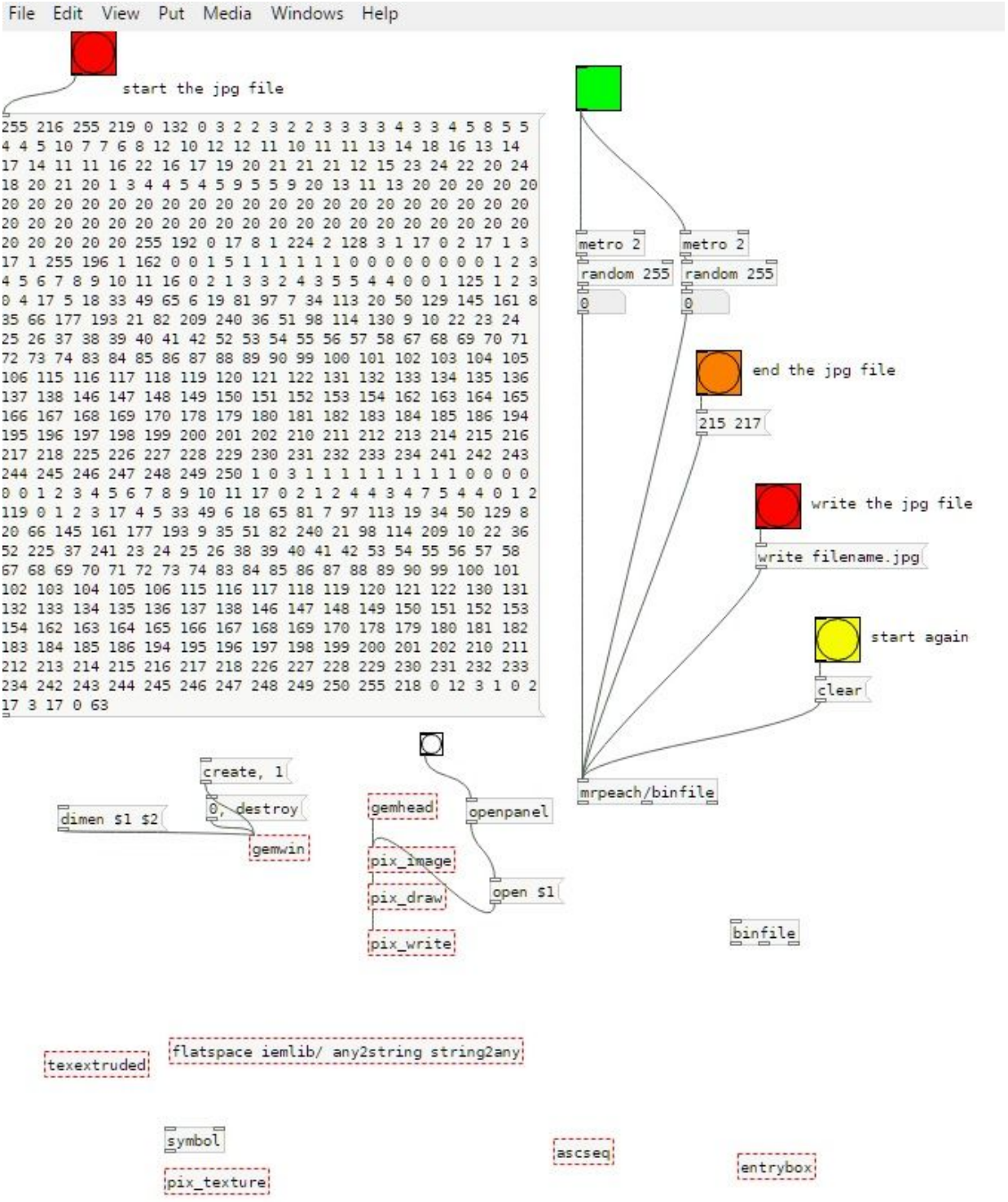


Fonte: Arquivo pessoal (2018)

Tentamos fazer as nossas modificações para que uma possível alteração do *patch* original nos rendesse ainda maiores recursos que as informações já coletadas. Acrescentamos o nosso conjunto de objetos habituais, voltados para geração da janela e leitura de uma imagem através do GEM com seus objetos de prefixo “pix” e o [gemwin]. Experimentamos com o objeto [symbol] para tentar importar arquivos ou converter o binário a ASCII. Apesar de a análise do *patch* ter nos trazido informações importantes, as nossas alterações não produziram um novo *patch* que pudesse importar imagens e gerar *glitch* a partir delas. Mas o acréscimo do objeto [mrpeach/binfile] como um de nossos recursos é

de suma importância, como veremos posteriormente. Uma informação extra é que mrpeach é o pseudônimo do desenvolvedor Martin Peach, que criou o objeto [mrpeach/binfile], em 27 de janeiro de 2014.

Figura 2 - *Patch* Antonio Roberts alterado



Fonte: Arquivo pessoal (2018)

Apesar da contribuição desse *patch,* ele ainda parecia distante dos nossos métodos adquiridos pela pesquisa bibliográfica e os experimentos que antecedem o *Pure Data*. A não utilização do código

ASCII durante a geração do *glitch* e a incapacidade de importar novos arquivos, no momento da análise deste *patch,* parecia distante dos nossos intuitos originais de automatizar o processo de geração de *glitch* com o *Pure Data*. Ainda insistimos, com o nosso *patch,* na utilização de objetos relacionados a texto o [textfile] e com ASCII o [symbol].

Retornamos ao fórum para relatar essas interessantes novas informações e vimos que temos novas contribuições do usuário Alexandros[94] , através de seu site o identificamos como:

> Alexandros Drymonitis é um músico, ativo no campo da música eletrônica experimental e música noise. Ele é formado em MMus no Conservatorium van Amsterdam, onde estudou guitarra e composição. Sua prática musical se concentra na textura do som e do ruído, explorando as fronteiras de controle, com a forma como objetivo principal. Seu objetivo educacional é fornecer conhecimentos adquiridos em programação multimídia. Ele colaborou com vários artistas de diferentes disciplinas de arte, além de vários conjuntos, ou conjuntos interdisciplinares de música. Ele é atualmente membro da Medea Electronique e colaborador do ARTéfacts Ensemble. Como membro da Medea Electronique, ele é o principal organizador da 'Electric Nights', um festival de música eletrônica ao vivo, com sede em Atenas. Ele também é membro da Ubique Media e membro fundador do 'Patching Circle Athens. Ele ensinou guitarra na Escola de Música de Amsterdã e no Conservatório "Philippos Nakas" em Atenas, e a programação de música eletrônica no Conservatório Musical Praxis em Atenas. Ele é atualmente um freelancer no campo da música eletrônica e programação multimídia, ensinando vários workshops em vários locais e realizando programação multimídia em vários eventos. Ele é o autor do livro 'Digital Electronics for Musicians', publicado pela Apress.(DRYMONITIS, 2018)[95]

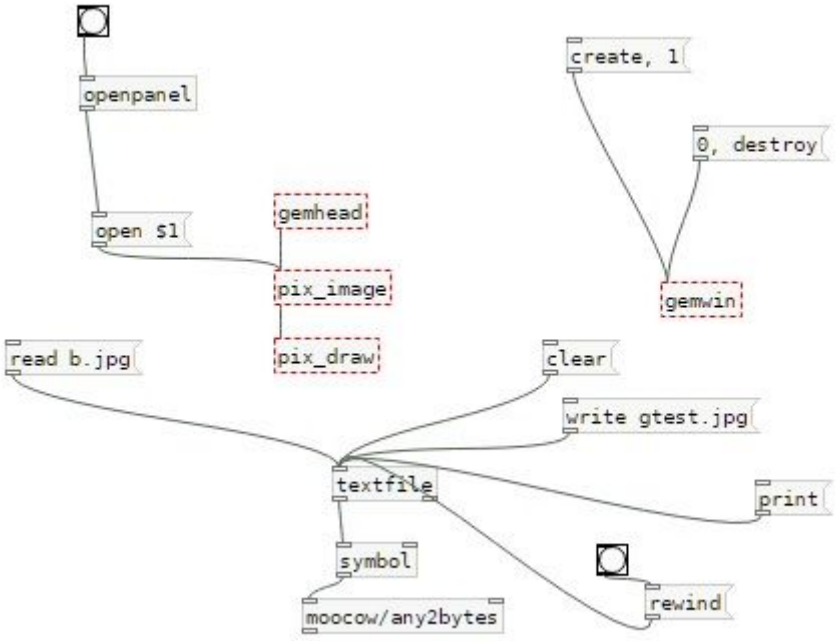
Através de suas sugestões ao nosso *patch* fizemos algumas modificações sem muito sucesso. O usuário do fórum LiamG[96] nos recomendou o uso do objeto [moocow/any2bites] originalmente.

Figura 3 - *Patch* versão 3

[94]In: < http://drymonitis.me/bio/ >.

[95]Tradução Nossa.

[96]Diferente dos outros outros usuários do fórum que contribuiram para nossa pesquisa, não conseguimos maiores informações sobre LiamG. Disponível em https://forum.pdpatchrepo.info/topic/10176/need-help-with-patch-for-a-glitch-project Acessado em 22 de mar de 2018.



Fonte: Arquivo pessoal (2018)

Fizemos os primeiros testes com o [moocow/any2bytes], porém, não obtivemos bons resultados. O *patch* não traduzia nada para bytes na forma que o havíamos construído. Através de novas indagações e orientações do fórum, chegamos a um resultado parecido com o de Antonio Roberts em relação a sua utilização de binário para *glitch*. Veremos nas imagens a seguir que, desta vez, eliminamos o objeto [symbol] abandonando por hora o trabalho com ASCII e testando o uso do objeto [moocow/any2byte][97], que lida com código binário, no *patch* mostrado na figura 23. Seguimos a dividir nossas dúvidas ao fórum. Onde a dificuldade que tínhamos uma grande quantidade de informação em formato de bit, e nesta etapa possuíamos interesse em trabalhar com o código ASCII, além dessa informação está apenas presente no console do *Pure Data* e não no canvas. Recebemos as seguintes informações do fórum:

> [textfile] produz uma linha de texto quando recebe um bang. Mas acho que as linhas devem ser separadas por um ponto e vírgula. Eu tentei o seu patch com um .jpg que eu tinha e consegui algumas linhas longas e curtas. Eu realmente não sei como o formato .jpg é, então eu realmente não posso responder a essa pergunta.Para converter em ASCII use [list fromsymbol] e você obterá o ASCII. Você pode então manipular os valores como desejar, e enviá-los para [listar o simbolo] antes de

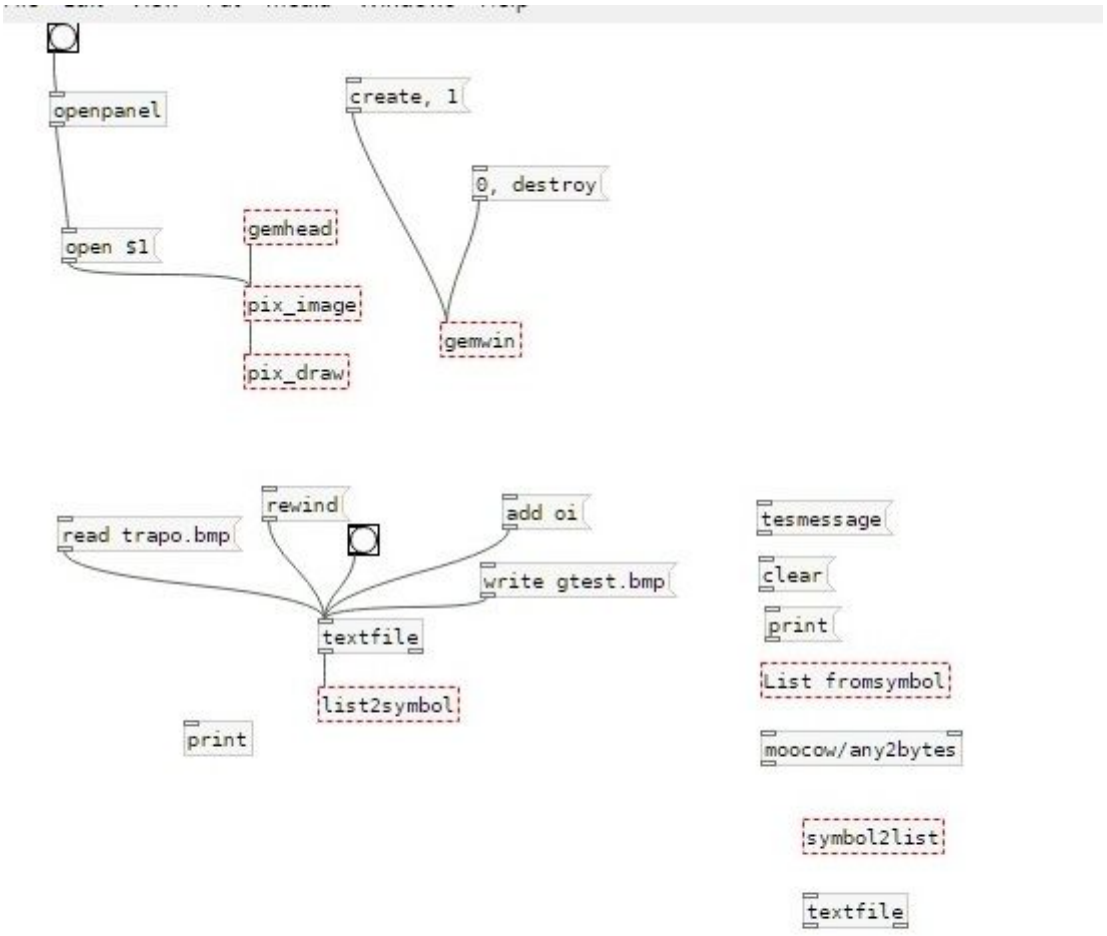
[97]Em uma tradução livre any2byte quer dizer "de qualquer coisa para bytes"

> escrevê-los em um arquivo usando [arquivo de texto] (acho que isso deve funcionar ...). (Alexandros[98], In: < https://goo.gl/YRvnbS >, 2016)[99]

Conseguimos, com o conjunto de nossas experiências e o auxílio do fórum, alterar o nosso *patch* de forma que era possível ver que o objeto [moocow/any2byte] traduzia a imagem para código binário. Exploramos um conjunto de outros objetos, apresentados abaixo na figura 24, sem grande sucesso. A maior parte dos objetos foram sugeridos pelo fórum e dedicamos tempo para ver os resultados e as formas de conexão entre eles, aprofundando também um pouco o nosso conhecimento de programação do *Pure Data*, explorando a diversidade de extensões que nos eram oferecidas pelo *Pure Data Extended*.

Figura 4 - versão 4

[98]Pseudônimo do autor do texto no fórum.
[99]Tradução nossa.

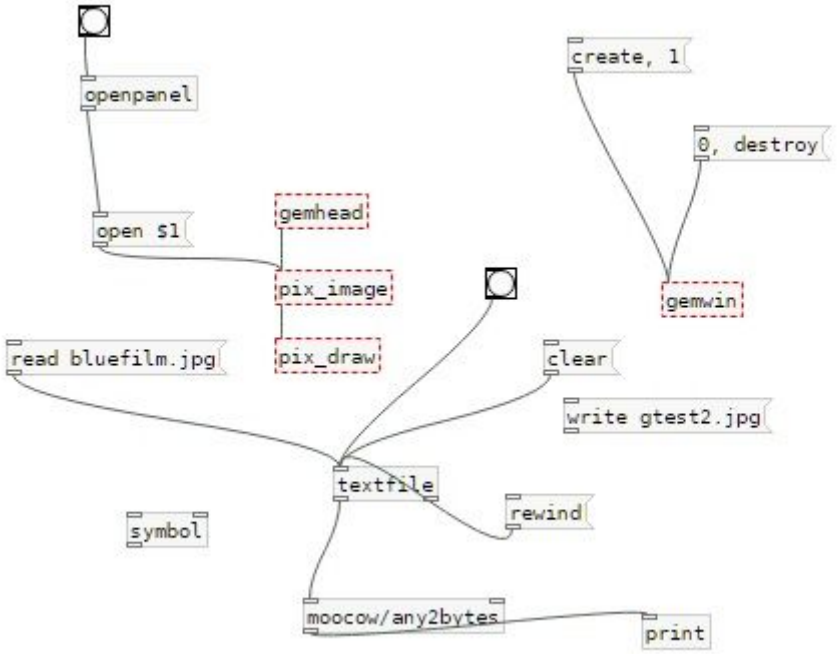


Fonte: Arquivo pessoal (2018)

Mudamos a forma de elaborar o *patch,* percebendo que talvez o código ASCII fosse realmente interessante para o desenvolvimento de *glitch* em linha de código, porém, não o ideal para o ambiente de desenvolvimento e linguagem de programação que havíamos escolhido. A alteração da imagem para binário parecia ser uma forma mais natural de se trabalhar com o fluxo de dados, não estávamos presos a trabalhar com ASCII. Mesmo o experimento mais bem sucedido com ASCII, o experimento com o Frhed, oferecia a opção de editar o arquivo em hexadecimal e, por vezes, fizemos uso do mesmo com resultados positivos. A mudança para código binário seria apenas mais uma forma de linguagem adotada em nossas experimentações. Passamos a escolher os melhores objetos, tanto pelo nosso entendimento de seu funcionamento, quanto por sua simplicidade de aplicação dentro do

objetivo que pretendíamos para trabalhar com o binário. O *patch* da figura 25 mostra esse aprimoramento e tivemos bons resultados com o mesmo.

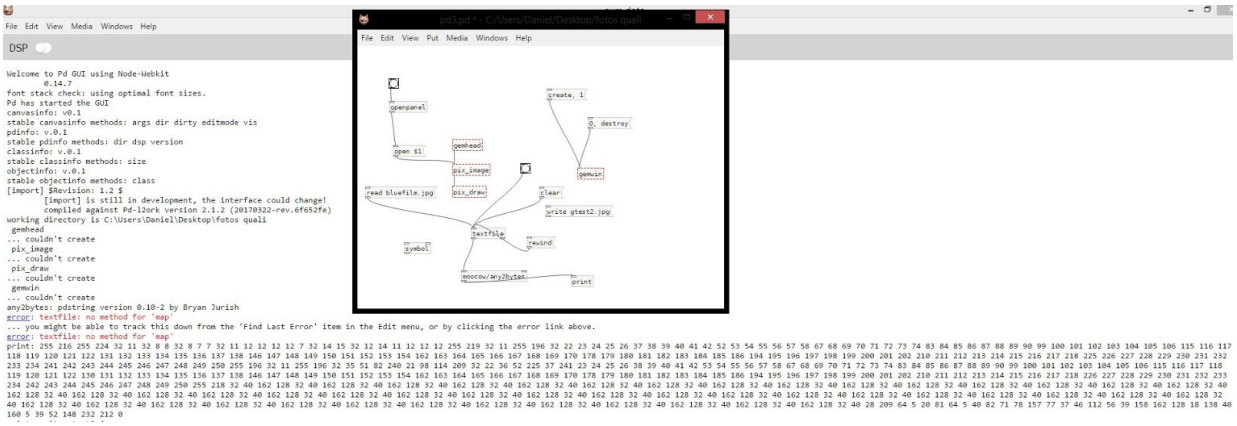
Figura 5 - versão 5



Fonte: Arquivo pessoal (2018)

Esse patch conseguia receber qualquer imagem JPEG que utilizamos como teste, fazia a transcrição da imagem para byte e exibia essa informação no console do Pure Data. Abaixo o resultado do patch com o código binário sendo mostrado no console do Pure data.

Figura 47 - *Patch* alterado, resultado no console



Fonte: Arquivo pessoal (2018)

Possuímos duas informações bem importantes com os nossos experimentos, que pareciam nos dar um resultado favorável para automatizar o processo de glitch com Pure Data. Com o *patch* versão 5, conseguimos chegar ao processo inicial que o patch original do Antonio Roberts possuía, uma mensagem contendo a informação relativa ao arquivo JPEG em código binário. Utilizando essa mensagem original, o *patch* (fig. 21) faz o *glitch* na imagem utilizando o objeto [random], porém, não estava embutido no *patch* como ele havia transcrito essa informação nem a forma de transcrever novas imagens.

Juntamos então o processo inicial do nosso *patch* versão 5 com a parte de *glitch* e criação do arquivo no *patch* versão 6. Depois de investigar o blog de Antonio Roberts, ele descreveu como foi obtida a informação contida na mensagem de seu patch, que na verdade é o header de um arquivo JPEG para ser preenchido com informação aleatória a fim de criar uma imagem JPEG randômica. Abaixo ele descreve o processo:

> A parte mais importante deste processo é saber como construir e aplicar um cabeçalho jpg para dados. A Wikipedia informou-me que todas as imagens de jpg começam com FF D8. Eu pensei que tudo o que eu precisaria fazer é usar um editor hexadecimal, como Ghex ou Bless Hex Editor, para adicionar esses valores de bytes a um arquivo.Infelizmente, este não é o caso. Há muito mais em um cabeçalho de jpg, como tabelas Huffman, tabelas de quantificação, bytes para definir a largura e a altura de uma imagem, e muito mais que ainda não entendo bem.Tentei pegar dados desde o início de um arquivo jpg aleatório, mas isso incluiu muitos dados estranhos, como a marca de câmera, programas usados para modificar a foto, dados de gps e data de criação. Esses dados totalizaram vários kilobytes, o que é muito para um cabeçalho. O que eu precisava era uma versão "baunilha" ou um cabeçalho simples que eu poderia aplicar a qualquer arquivo. mesmeon me mostrou o projeto HEADer REMIX

por Ted Davis. Os valores do cabeçalho no lado esquerdo da tela são usados para glitch as imagens, seja a imagem padrão ou uma selecionada por um usuário. Eu salvei a imagem padrão, extrai manualmente da imagem o cabeçalho, e executei-a através do exiftool e acabei com um cabeçalho para uma imagem de 640 × 480 que é apenas 588 bytes! (ROBERTS, 2012)[100]

[100]Tradução Nossa.

APÊNDICE B - Sugestão apresentada no fórum de *Pure Data*

A postagem e a solução apresentada por ele seguem abaixo:

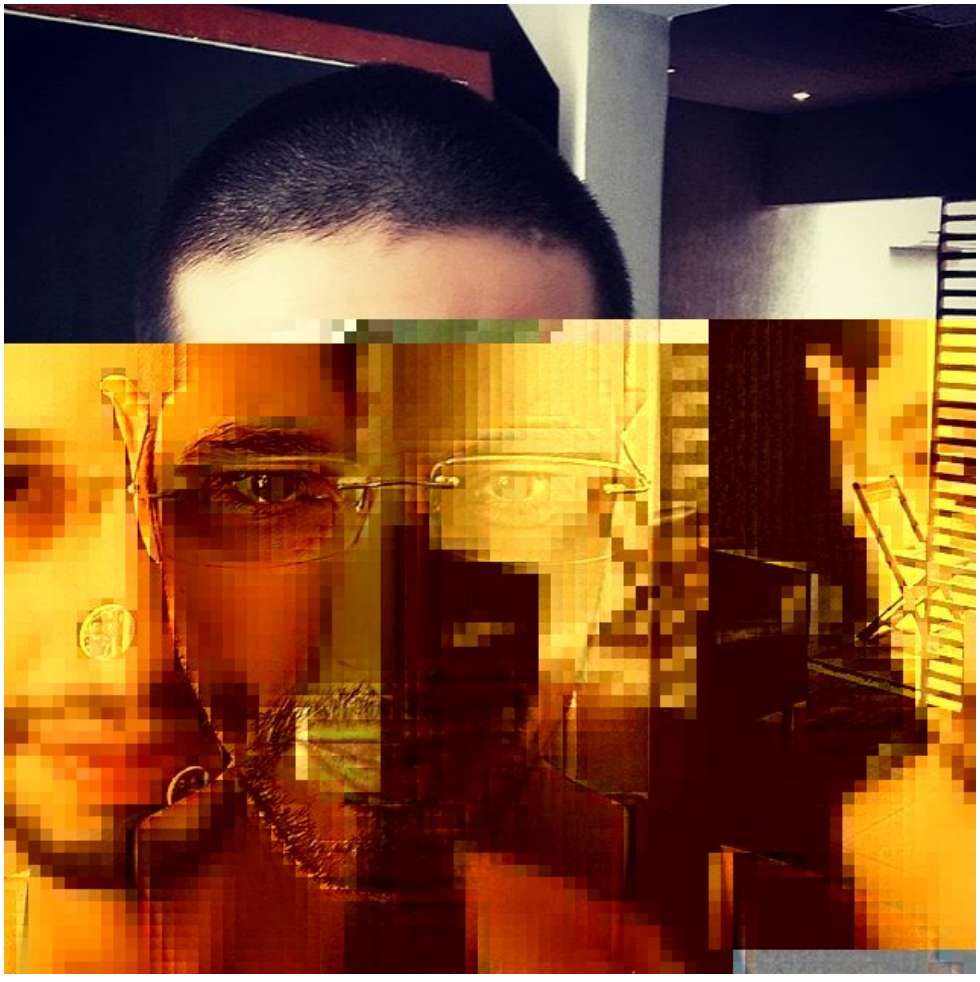
> "Se você procurar nas páginas da Wikipedia por todos os diferentes formatos de vídeo e vídeo, você verá que todos os arquivos terão um cabeçalho, que diz ao sistema operacional como interpretá-lo (mesmo os formatos não compactados como .bmp). Se você falha (corrompe) o cabeçalho, então o arquivo não pode ser lido. Então você não deve corromper os dados até que você esteja alguns (ou alguns milhares) bytes no arquivo (quantos depende do formato do arquivo ..... jpg, tiff, bmp etc.) Eu encontrei o arquivo de Antonio Roberts no meu computador e o modifiquei para enviar os códigos ascii ele pode fornecer um começo para o seu projeto JPG - binary - killer_mod. pd Examine (clique com o botão direito do mouse) uma das caixas de abstração verdes para ver como você pode escrever seu novo arquivo e salve-o e, mais importante, como você pode evitar a modificação do cabeçalho Você não pode usar [enviar glitch_ascii_code_list] como os códigos de cabeçalho já estão quebrados !! é apenas para visualização no terminal. Se você tentar escrever os "símbolos" ascii reais em um arquivo de mensagens, você receberá as mensagens "chave de fechamento" que você está vendo ou pior! O Pd tentará interpretar alguns caracteres como mensagens de "controle", e isso travará o Pd. Você terá que trabalhar com os números que representam os "caracteres" (ascii ou binário) e não os caracteres "texto" .... a, b, c, etc ....."(WHALE-AV[101], https://goo.gl/YRvnbS, 2016)[102]

[101]Pseudônimo do autor do texto no fórum.
[102]Tradução nossa.

APÊNDICE C - Resultados dos Experimentos

Abaixo, uma coleção dos resultados obtidos nos experimentos em todas as etapas, desde as primeiras experiências utilizando editores de texto para manipulação do código da imagem até os resultados finais conseguidos com o *patch* de *Pure Data*. Os resultados apresentados estão de acordo com a ordem listada no sumário.

Figura 1 -Resultado de *glitch* com notepad++



Fonte: Arquivo pessoal (2018)

Figura 2 - Exemplo de *glitch* utilizando o Frhed

Fonte: Arquivo pessoal (2017)

Resultado obtido utilizando o Audacity para se gerar *glitch* em uma imagem.

Figura 3 -*Glitch* Audacity

Fonte: Arquivo pessoal (2016)

Resultados musicais

Apesar das dificuldades técnicas de mixar áudio de frequências diferentes e da notória falta de dinâmica no áudio 8 bits, a experiência de compor por expressões matemáticas obteve um resultado de campo sonoro diferenciado. Apesar do pouco domínio matemático, a composição resultou em um experimento que pode ser fruto de novas pesquisas.

A mistura de notação musical tradicional para uma linguagem matemática musical pode facilitar a aproximação de pessoas com interesse de compor, mas que não o fazem pela falta de domínio da notação musical tradicional.

O processo de produção se modifica muito no aspecto de fidelidade de áudio, pois, além de se trabalhar com frequências e bits mais baixos que o habitual, o próprio processo de *glitch* usado no áudio funciona melhor com mídias comprimidas. Muitos trechos da música que originalmente eram em formato WAVE foram codificados[103] para MP3, inclusive no resultado final. A última versão da música, pelo seu processo de produção, não resulta em um arquivo WAVE máster, e sim em um MP3. Esse processo não convencional trabalha o formato do arquivo como parte do processo de composição, e não somente como uma ferramenta de trabalho, um invólucro para dados. A composição se vale de uma razoável gama de ruídos digitais, recurso da *glitch art* que, juntamente com outras similaridades, traz a hipótese de sua ligação com o grotesco.

A música The End desenvolvida nesta dissertação pode ser escutada em:
https://soundcloud.com/sarabade/the-end

[103]Codificado ou transcodificado.

Resultados obtidos utilizando o *patch*, *glitch like.*

Figura 4 - Performance de Luna dias com *patch glitch like*

Fonte: Arquivo pessoal (2018)

Figura 5 - Performance de Luna Dias com *patch glitch like #2*

Fonte: Arquivo pessoal (2018)

Comparando as duas imagens, a da direita utilizando o *patch* em sua versão final e a direita o imagem original.

Figura 6 - Comparativo

Fonte: Arquivo pessoal (2018)

Figura 7 - Obra de Thiago Trapo com *Pure data Glitch art*

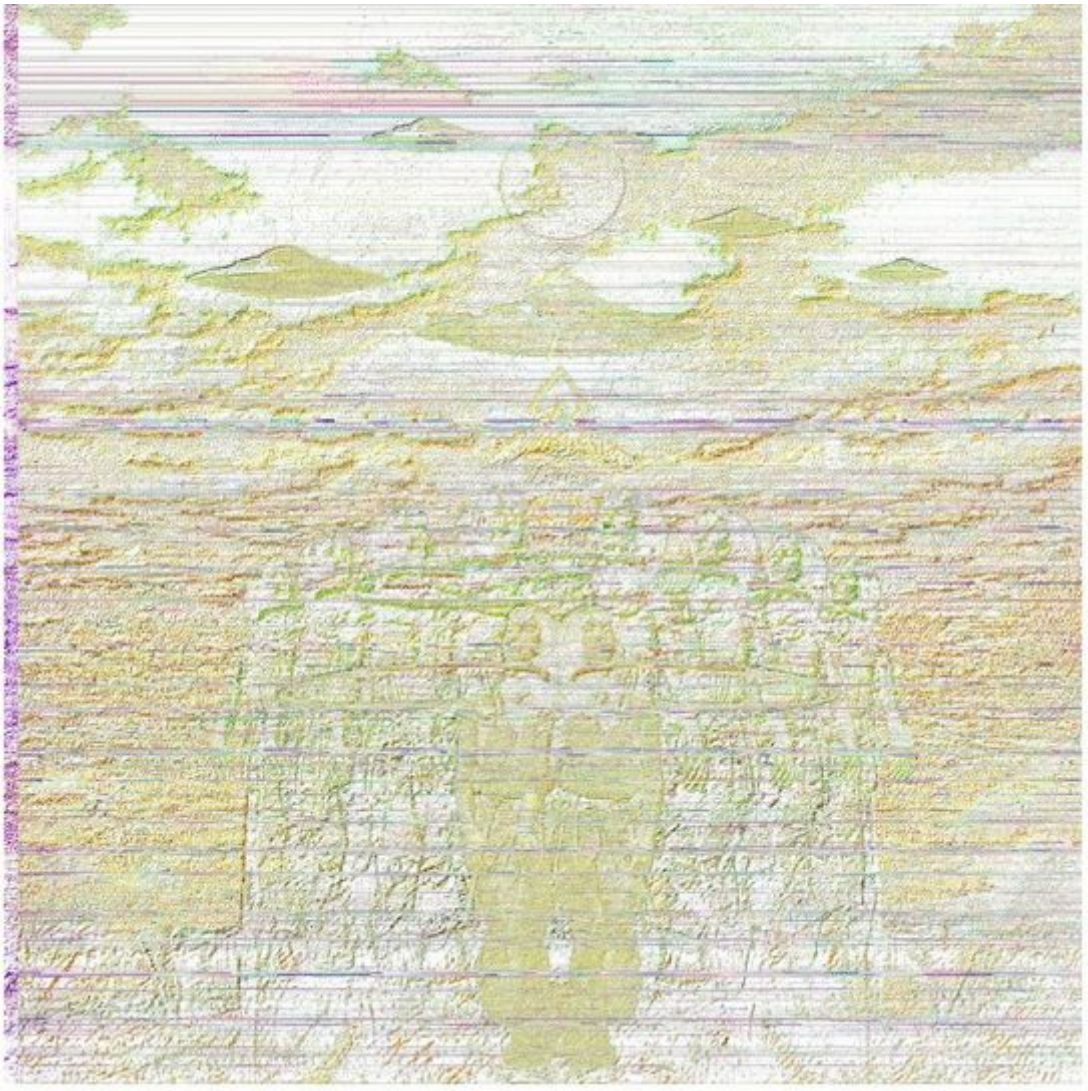
Fonte: Arquivo pessoal (2018)

Comparando as duas imagens, a da direita utilizando o *patch* em sua versão final e a direita o imagem original.

Figura 8 - Comparativo II

Fonte: Arquivo pessoal (2018)

Figura 9 - Obra de Thiago Trapo com *Pure data Glitch art*



Fonte: Arquivo pessoal (2018)

Resultados desenvolvidos com artista da região utilizando o *patch* finalizado.

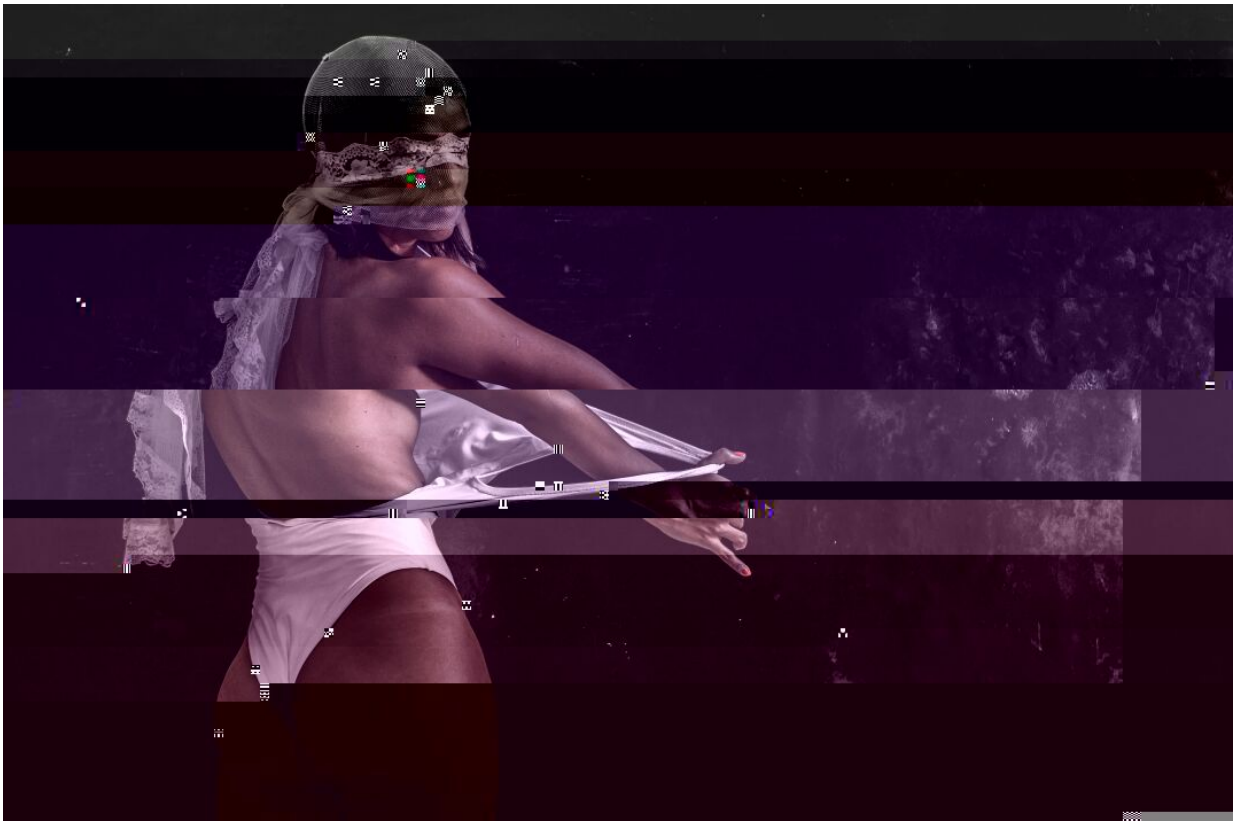
Figura 10 - Fotografia de Raquel Medeiros com *Pure data glitch art*

Fonte: Arquivo pessoal (2018)

Figura 11 - Fotografia de Raquel Medeiros com *Pure data glitch* #2

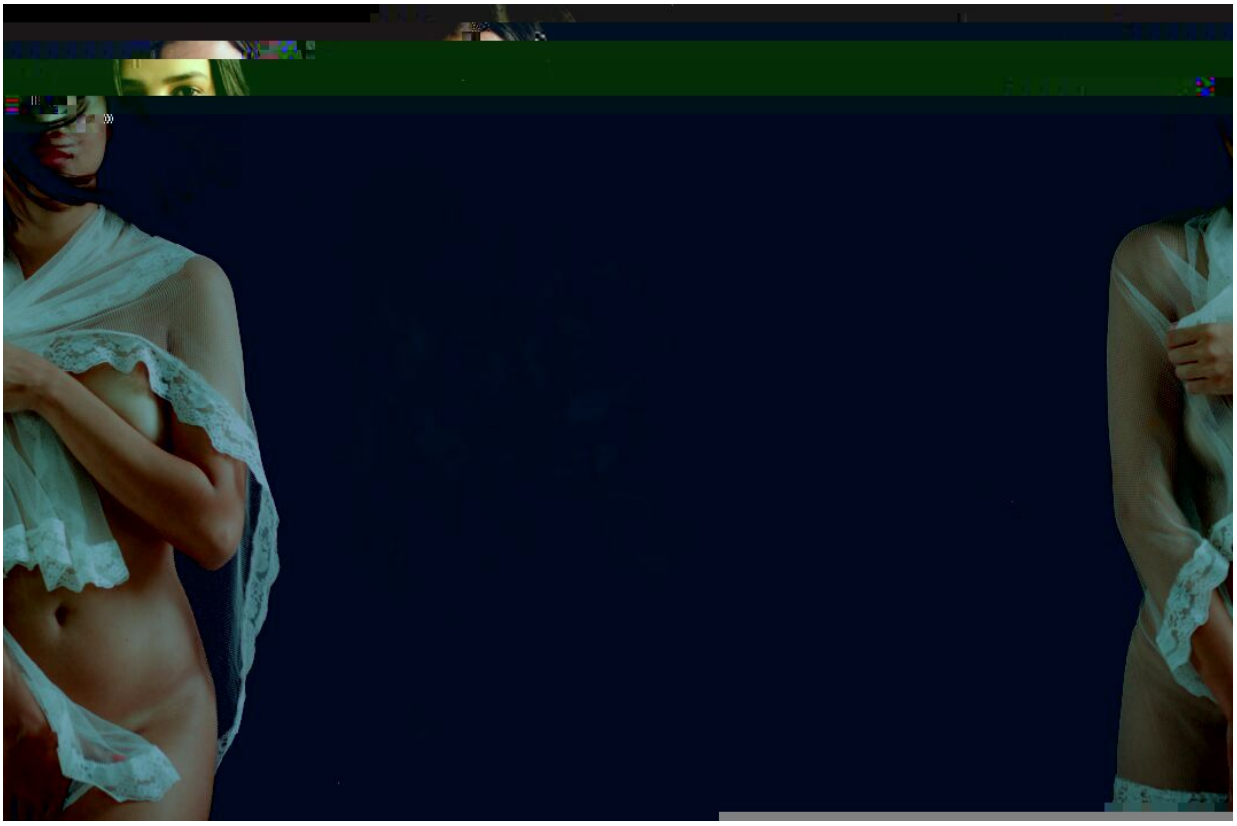
Fonte: Arquivo pessoal (2018)

Figura 12 - Fotografia de Isis Andrade com *Pure data Glitch art*



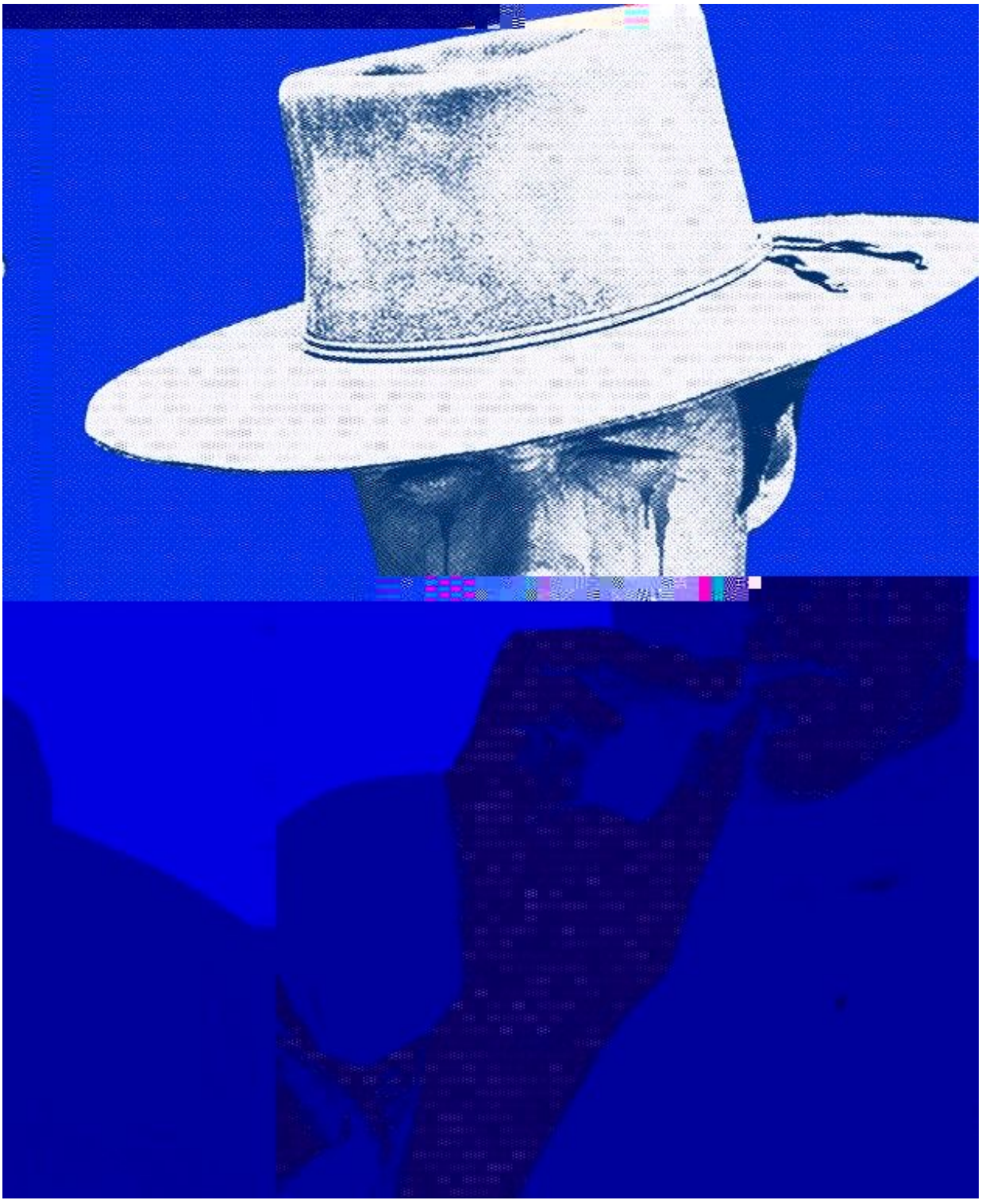
Fonte: Arquivo pessoal (2018)

Figura 13 - Fotografia de Isis Andrade com *Pure data Glitch art* #2



Fonte: Arquivo pessoal (2018)

Figura 14 - Obra de Thiago Trapo com *Pure data Glitch art* #3



Fonte: Arquivo pessoal (2018)

Figura 15 - Obra de Thiago Trapo com *Pure data Glitch art* #4

Fonte: Arquivo pessoal (2018)

Figura 16 - Obra de Thiago Trapo com QR code que leva a imagem criado com *Pure data Glitch art*

Fonte: Arquivo pessoal (2018)

Figura 17 - Obra de Thiago Trapo com QR code que leva a imagem criado com *Pure data Glitch art* #2

Fonte: Arquivo pessoal (2018)

Figura 18 - Obra de Thiago Trapo com *Pure data Glitch art* imagem exibida no QR CODE

Fonte: Arquivo pessoal (2018)

Figura 19 - Obra de Thiago Trapo com *Pure data Glitch art* #5

Fonte: Arquivo pessoal (2018)

Figura 20 - Obra de Thiago Trapo com *Pure data Glitch art* #6

Fonte: Arquivo pessoal (2018)

A videodança Dead Pixel desenvolvida nesta dissertação pode ser assistida em:

https://vimeo.com/230278137

Figura 21 - Videodança de Luna dias com edição *glitch* e *glitch like*

Fonte: Arquivo pessoal (2018)

A videodança Aquarelas desenvolvida nesta dissertação pode ser assistida em:

https://vimeo.com/205762559

APÊNDICE D - Código do patch em sua versão final

```
#N canvas 1813 935 820 830 10;
#X obj 388 596 mrpeach/binfile;
#X obj 250 544 random 255;
#X floatatom 250 569 5 0 0 0 - - -, f 5;
#X obj 315 650 print;
#X msg 321 601 add \$1;
#X msg 192 570 rewind;
#X msg 361 664 writeat \$1;
#X obj 297 536 random 5000;
#X floatatom 294 571 5 0 0 0 - - -, f 5;
#X floatatom 160 581 5 0 0 0 - - -, f 5;
#X obj 394 632 +;
#X obj 197 600 openpanel;
#X msg 389 671 write glitched.jpg;
#X msg 301 633 read \$1;
#X obj 239 668 delay 500;
#X obj 351 540 bng 31 250 50 0 empty empty empty 17 7 0 10 -262144
-1 -1;
#X obj 1 -90 cnv 15 832 847 empty empty empty 20 12 0 20 -133184 -66577
0;
#X text 461 604 Based on work by Little-Scale http://little-scale.blogspot.com
;
#X text 459 572 2013 Antonio Roberts http://www.hellocatfood.com;
#X text 462 646 GNU LESSER GENERAL PUBLIC LICENSE Version 3;
#X text 459 543 Daniel Marques \, Carlos Batista \, José Tonezzi \,
2018 UFPB;
#X obj 213 11 vsl 54 539 0 5000 0 0 empty empty tweak\ Glitch 0 -9
1 30 -260097 -1 -1 0 1;
#X obj 13 -2 bng 113 250 50 0 empty empty Select\ File 17 7 1 30 -258049
-258049 -1;
#X obj 416 0 bng 142 250 50 0 empty empty Glitch 17 7 1 30 -260113
-1 -1;
#X obj 619 -63 bng 167 250 50 0 empty empty Write\ File 17 7 1 30 -4033
-1 -1;
#X connect 0 0 3 0;
#X connect 1 0 2 0;
#X connect 2 0 4 0;
#X connect 4 0 0 0;
#X connect 5 0 0 0;
#X connect 6 0 0 0;
#X connect 7 0 8 0;
#X connect 8 0 10 0;
#X connect 9 0 10 0;
#X connect 10 0 6 0;
#X connect 11 0 13 0;
#X connect 11 0 15 0;
#X connect 12 0 0 0;
#X connect 13 0 0 0;
#X connect 14 0 5 0;
#X connect 15 0 14 0;
#X connect 21 0 9 0;
#X connect 22 0 11 0;
#X connect 23 0 1 0;
```

```
#X connect 23 0 7 0;
#X connect 24 0 12 0;
```