TextAlive App API: 「リリックアプリ」の提案とプログラミング・コンテストでの実証実験

加藤 淳[1,a)]　後藤 真孝[1,b)]

概要：音楽に合わせてタイミングよく歌詞が動くリリックビデオは楽曲のプロモーション手段として一般化したが，いつ再生されても同じ内容を提示するため，視聴者は受動的に楽しむしかない．そこで我々は，ユーザとのインタラクションにより歌詞のテキストを再生のたびに異なる方法で提示でき，静的メディアの制約を取り払える歌詞駆動型のインタラクティブな視覚表現を「リリックアプリ」と定義する．そして，この表現形式をプログラマやミュージシャンに開放するため，リリックアプリを開発・配信できる Web ベースのフレームワーク「Lyric App Framework」を提案する．当該フレームワークは，我々が研究・開発・運営してきたリリックビデオ制作支援サービス「TextAlive」の Web インタフェースと，歌詞駆動の表現を開発できる機能をプログラマ向けに開放する「TextAlive App API」で構成される．当 API は，既存の，プログラマが使い慣れたクリエイティブコーディングのためのライブラリと相補的な役割を果たし，インタラクティブなリリックアプリをすぐに開発可能である．我々は，2020 年に当 API を一般公開し，新たな表現形式の可能性を探ってきた．とくに，創作文化に関するイベント「マジカルミライ」ではプログラミング・コンテストが毎年開催され，最初の 2 回で 52 作品が集まった．これらの作品を分析して得られたリリックアプリのカテゴリ 8 種類と，音楽とプログラミングの未来に関する示唆を報告する．

1. はじめに

歌詞には，楽曲単体よりも感情表現を伝えやすくする力があり [1]，メディア技術の進歩とともに多様な方法で聴衆に届けられてきた．例えば，レコードやコンパクトディスク (CD) には，グラフィックデザイナが歌詞を魅力的にデザインした歌詞カードがついてきた．また，音楽動画や動画共有サービスが一般化してからは，楽曲再生に合わせて歌詞が魅力的に動く「リリックビデオ (lyric video)」が人気を博すようになった．リリックビデオにおいて文字が時系列で変形する表現形式は「キネティックタイポグラフィ (kinetic typography)」と呼ばれ，多くの場合，モーショングラフィックデザイナが動きを手付けしているが，著者らは，音楽理解技術を活用してその制作を省力化し，支援するサービス「TextAlive」[2] を研究，開発，運営してきた．そして今や，ミュージシャンは多様なメディアを横断して聴衆に楽曲を届けるようになり (メディアコンバージェンス [3])，聴衆は楽曲を聴くだけでなく，ソーシャルネットワーキングサービス (SNS) への投稿などを通して自らの体験を共有するようになった (参加型文化 [4])．また，スマートフォンの普及により，多くの人々が楽曲との斬新なインタラクションを体験できるようになってきた [5]．

こうした社会・文化・技術的背景を踏まえ，我々は本稿において，歌詞カードやリリックビデオに続く歌詞駆動メディアとして，楽曲と歌詞を目と耳でインタラクティブに楽しめる「リリックアプリ (lyric app)」(図 1) を定義する．例えば，リリックアプリを起動したユーザは，楽曲を聴きながら，三次元コンピュータグラフィックス (3D CG) で動的に生成された情景を見ることができる．歌詞のフレーズを表す視覚的オブジェクトが発声タイミングに合わせて登場するが，静的なリリックビデオとは異なり，ユーザは情景のなかを自由に動き回り，フレーズに触れることができる．このように，ユーザは歌詞を読むだけでなく，歌詞とインタラクションすることで，楽曲を再生するたびに一度きりの情景を作り上げることが可能となる．リリックビデオという静的メディアと比較すれば，リリックアプリには，インタラクションデザイン上，工夫の余地がはるかに多く存在する．

一方で，リリックアプリの開発においては，リリックビデオの制作とは異なる課題に取り組む必要がある．まず，楽曲再生に合わせて歌詞や視覚的表現を表示するためには，事前準備として歌詞やビート構造などの音楽的要素の

[1] 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST)
[a)] jun.kato@aist.go.jp
[b)] m.goto@aist.go.jp

図 1: 「リリックアプリ」は，ユーザが歌詞駆動の視覚表現とインタラクションできるようにすることで，リリックビデオの限界を突破する新たな表現形式である．本稿で提案する「Lyric App Framework」はプログラマとミュージシャンによるリリックアプリ開発を支援する．

タイミング情報を手作業でラベル付け (アノテーション) しておかなければならない．また，音楽的要素に同期して見える時間制約が厳しいインタラクションデザインは容易ではない．さらに，開発したリリックアプリをユーザに届ける方法も自明ではない．

そこで我々は，リリックアプリの開発を支援し，この新たな表現形式の可能性を探るため，プロダクションレディ (production-ready) なリリックアプリを構築するための一貫した開発体験を提供する Web ベースのフレームワーク「Lyric App Framework」 [6] を提案する．なお，ここでプロダクションレディとは，プロトタイプ開発だけでなく一般ユーザによる利用にも耐える高い信頼性でのアプリの配信までサポートすることを指す．具体的には，このフレームワークは，筆者らによる Web サービス「TextAlive」[*1]の機能を活用することで，楽曲と対応する歌詞テキストを入力として受け取ると，音楽理解技術により，歌詞タイミングなどの音楽的要素の情報を自動的に推定する．そして，楽曲の再生を制御し，楽曲中の特定のタイミングの音楽的要素の情報へアクセスするために新たに設計した API「TextAlive App API」[*2]を提供する．こうして開発したリリックアプリは，スマートフォンやパソコン (PC) の Web ブラウザで実行できる Web 標準に則った Web サイトとして配信できる．筆者らは，このフレームワークを「TextAlive for Developers」[*3]という Web サイトで一般公開済みである．

本研究の貢献は以下の 3 点である．

(1) リリックビデオを拡張して，楽曲と歌詞に関するインタラクションを可能にする仕組みを加え，新たな表現形式「リリックアプリ」を定義したこと．
(2) Web ベースのワークフローと API (TextAlive App API) を提供する新たなフレームワークを実装し，プロダクションレディなリリックアプリを構築するための一貫した開発体験を提供したこと．
(3) フレームワークを一般公開し，創作文化に関するイベント「マジカルミライ」でプログラミング・コンテストを 2020 年から毎年開催し，最初の 2 年で集まった 52 個の応募作品をもとに，リリックアプリの 8 つのカテゴリを明らかにし，フレームワークの有効性を評価するとともに，今後の研究のための知見を得たこと．

2. リリックアプリ

本章では，我々が研究・開発・運営してきたリリックビデオの制作支援サービス「TextAlive」を紹介し，インタラクティブな表現形式「リリックアプリ」を着想した経緯を説明する．さらに，リリックアプリの典型的なインタラクションデザインを具体例とともに紹介し，リリックアプリ開発に際してプログラマが直面する課題を明らかにする．

2.1 リリックビデオからリリックアプリへ

2015 年に，我々は ACM CHI 2015 [2] (WISS 2014 [7]) で提案したインタラクションデザインをもとに，リリックビデオを Web ブラウザ上で制作できる支援サービス「TextAlive」を開発して一般公開した[*4]．TextAlive は，単なる動画制作支援サービスではなく，歌詞の文字列をアニメーションさせたり，グラフィックを描画したりするための視覚的アルゴリズムをプログラマが開発できる機能を備えた統合デザイン環境である．2023 年 2 月 6 日の時点で視覚的アルゴリズムが 1396 バージョン保存されており，18,867 件の動画データが保存されている．プロ・アマチュアを問わず，多くのミュージシャンが音楽作品のプロモーション用のリリックビデオ制作に活用する Web サービスとなっている．

ただし，ミュージシャンには有用であっても，プログラマの観点では，TextAlive は創造性を充分に活かせない課題が 2 つあることが分かってきた．まず，プログラマは

[*1] https://textalive.jp
[*2] https://github.com/TextAliveJp/textalive-app-api
[*3] https://developer.textalive.jp
[*4] https://www.aist.go.jp/aist_j/press_release/pr2015/pr20150908/pr20150908.html

TextAlive のライブプログラミングなどの先進的な機能を活用できる一方で，それ以外のプログラミング環境を使う自由がなかった．すなわち，自身が使い慣れた開発環境やクリエイティブコーディング用のライブラリを利用することができなかった．次に，プログラマは，ユーザがアルゴリズムをカスタマイズするためのパラメタ (色や数値などの決められた型の変数) を定義することはできても，それ以上のインタラクティブな機能を実装することはできなかった．リリックビデオという静的な表現形式が，プログラマの足枷になってしまっていた．

これらの経験から，我々は，リリックビデオにインタラクティブな機能を追加できる「リリックアプリ」という表現形式を着想した．我々は，プログラマに特定のプログラミング環境の利用を強いるのではなく，好みの開発ツールや環境を利用しながら「リリックアプリ」を開発できるフレームワークを提供する．また，ミュージシャンがリリックアプリをカスタマイズして自身の楽曲を宣伝できるようにする．聴衆は，リリックアプリを通して楽曲を聴き，歌詞を読み，歌詞とインタラクションできるようになる．

2.2 典型的なリリックアプリの例

前節では主にクリエータ (プログラマとミュージシャン両者) の視点でリリックアプリがメリットのある表現形式であることを紹介した．本節では，ユーザの視点で，リリックアプリによるインタラクティブなユーザ体験のイメージを，下記の「アリスが (架空の) リリックアプリを楽しんでいる」という筋書きで例示する．

> ある日の午後，アリスはお気に入りのミュージシャンの新しい SNS 投稿を見つける．投稿内のリンクを辿ると Web ブラウザがリリックアプリを起動し，新しくリリースされた楽曲が流れ始める．楽曲のジャケット画像とタイトルが 3D CG 空間内に浮かび上がる．
>
> 静的なリリックビデオとは異なり，アリスは画面をタッチして空間内を自由に動き回り，楽曲のテーマに合ったさまざまな視覚的オブジェクトを見つけることができる．そうしたオブジェクトはビートと同期してアニメーションしており，目を楽しませてくれる．歌詞が発声するタイミングでは，歌詞のフレーズが視界に飛び込んでくる．そして，それを読むだけでなく，タッチ操作でつかんで好きなところに置いておくことができ，一度きりの情景を作りあげることができる．サビではアニメーションの効果が派手になり，気持ちを盛り上げてくれる．
>
> 夕方，アリスがふたたびリリックアプリを開くと，同じ 3D CG 空間が夕闇のような暗い色で表示される．歌詞のフレーズも昼間とは違った配置になり，楽曲と歌詞を新鮮に楽しむことができ，楽曲をもっと好きになったように感じられる．さらに，リリックアプリの中で，そのミュージシャンの別の曲を開くこともできる．アリスは，こうしたインタラクションを通して楽曲の世界観により深く浸り，個人的かつ唯一無二な体験を得られる．

2.3 リリックアプリ開発の課題と支援の必要性

上記の筋書きを踏まえ，リリックビデオ制作とアプリ開発を比較することで，本節ではリリックアプリを開発する際にプログラマが直面する 3 つの課題と，それぞれに望まれる開発ツールによる支援について考察する．

まず，リリックビデオとリリックアプリを含む歌詞駆動型の視覚表現は，楽曲再生と同期している必要がある．そうした同期を実現するためには，通例，歌詞を含む音楽的要素のタイミング情報を手作業でラベル付け (アノテーション) しなければならない．我々はリリックビデオの制作支援に関する先行研究 [2, 7] において，音楽理解技術による自動解析と，解析結果を修正するためのユーザインタフェースを提案した．こうした半自動的な処理はリリックアプリ開発においても有用だと思われるが，アプリの開発ワークフローの中へどう組み込むかは自明ではない．

次に，ユーザと楽曲および歌詞との効果的なインタラクションデザインは容易ではない．リリックビデオでは対象楽曲が一意に決まっており，時間軸に沿った場面展開も最終的には一案に収束するため，モーショングラフィックデザイナはその完成形に向けて試行錯誤するだけでよい．一方，リリックアプリ開発では，プログラマは，何度実行されても飽きないような視覚効果を生成するアルゴリズムを思いつく必要がある．クリエイティブコーディングのためのライブラリ (3.3 節) の助けを借りることはできるが，リリックアプリ特有の課題，すなわち，さまざまな音楽的要素のタイミングや，歌詞の意味，そしてユーザ入力を考慮に入れる難しさは残る．したがって，クリエイティブコーディングの実践を邪魔しないようなツール支援が極めて重要である．

最後に，リリックアプリをユーザに届ける手段が明らかでない．創造性支援ツールの研究はラボスタディが中心で実用性の面で批判されることも多い [8]．しかし我々は，リリックアプリという新たな表現手段を提案するうえで，ラボ内に留まらず，プロダクションレディなアプリを開発できるような工学的視点を持つことが重要であると考えている．ミュージシャンが動画共有サービスを使ってリリックビデオを多数の視聴者へ配信できることと同様に，プログラマやミュージシャンがリリックアプリをユーザに届けられるようにすることまで考慮するべきである．

3. 関連研究

本章では，関連研究を紹介して本研究の貢献を明確にす

る．まず，リリックアプリに関連する先行事例を紹介する．そして，マルチメディアにおける複数トラックの自動同期技術と，マルチメディアコンテンツを生成したりインタラクション可能にしたりするためのプログラミング支援技術を紹介する．

3.1 リリックアプリに関連する先行事例

視覚と聴覚のメディアが同期することで，没入感のある体験が実現できる．カラオケは，歌詞のシンプルな字幕表示を見ながら，歌詞を正確に覚えていなくても歌を歌って，楽曲を楽しめる娯楽である．機器と楽曲によっては，リリックビデオを再生することで没入感を高める工夫がなされていることもある．リズムゲーム [9,10] や，タイピングや外国語を練習する音楽ゲーム [11] では，楽曲再生と同期して歌詞を表示できるものもある．

こうしたアプリケーションは，歌詞やビートなどの音楽的要素のタイミング情報を利用しており，リリックアプリに関連する先行事例と捉えることができる．しかし，これらは，インタラクションデザインの観点では比較的狭い応用範囲に留まっている．カラオケはインタラクティビティがほとんどなく，ゲームはゲームプレイ自体の楽しさやゲーミフィケーションによる学習効果の向上に主眼が置かれている．我々は，リリックアプリにはこれらの先行事例を超えたポテンシャルがあると考えており，デザイン空間を 6 章で示すように探索するため，リリックアプリ開発を支援するフレームワークを構築した．

3.2 自動同期技術

歌詞や音楽的要素と同期して感じられる体験は，リリックアプリに欠かせないものである．このように時系列に並ぶマルチメディアのコンテンツを自動的に同期したり生成したりといった例は，過去にもある．

音声付き動画を自動的に書き起こし，得られたタイミング情報付きテキスト，音声，そして動画を統一的に扱えるようにするインタラクション技術は，長く研究されてきた．PodCastle [12] は自動音声認識をポッドキャストに適用し，クラウドソーシングによるユーザ編集で精度を向上し，書き起こし内の全文検索機能を提供した．一方，本研究は，クラウドソーシングで精度を向上する同様の手法を採用し，さらにユーザによる編集結果のリビジョン管理を可能にしている．Video Digests [13] は，動画コンテンツを要約するためのデータ構造を定義し，音声の書き起こしに対するインタラクションによって要約結果を編集できるユーザインタフェースを提供した．本研究は同様にタイミング情報付きのデータ構造を扱うが，一般的な音声でなく歌声を対象としている．さらに，歌詞を含む音楽的要素を活用し，インタラクティブな表現形式を新たに提案するものである．

歌声を含む楽曲 (混合音) と歌詞の自動対応付けは，一般に，講演などの音声と書き起こしの対応付けと比較して技術的に困難である．これは，混合音から歌声を分離することが難しく，歌声が多様なためである．LyricSynchronizer [14] は，新たな信号処理技術を適用することで自動対応付けを行い，歌詞テキストをクリックすることで楽曲の再生位置を操作できるユーザインタフェースを提供した．TextAlive [2] は，さらにタイミングよく文字がアニメーションする Kinetic Typography という表現手法を活用したリリックビデオを制作できるようにした．また，アニメーションに用いられるアルゴリズムをライブプログラミングできるようにした．SyncPower [15]，Musixmatch [16]，そして LyricFind [17] は，歌詞テキストを取得するための API と，取得したテキストをカラオケのように表示するためのプチリリ機能，FloatingLyrics，そして Lyric Display と呼ばれる視覚的コンポーネントを提供している．本研究は，ベース手法としている TextAlive を除き，これらの先行事例と 2 つの点で異なる．まず，先行事例では既存の楽曲の情報を取得することしかできないが，本フレームワークでは，ユーザは新たな楽曲と対応する歌詞テキストを自由に登録できる．次に，既存の商用 API では，歌詞テキストを取得できても，タイミング情報に直接アクセスできない．一方，本研究が提供する TextAlive App API では，歌詞のテキストや，歌詞を含むさまざまな音楽的要素のタイミング情報へ直接かつ統一的な方法でアクセスできる．この 2 点の新規性は，プログラマが好きな楽曲のリリックアプリを開発できるようにする鍵となる．

音と映像が同期したメディアコンテンツを自動的に生成するための技術やフレームワークも，研究されてきた．MusicStory [18] は，楽曲を指定すると，その歌詞をインターネット上で取得し，関連する画像を指定されたデータセットから探し出し，楽曲のテンポにあった間隔で表示する動画を自動生成した．Crosscast [19] は，旅行に関するポッドキャストを与えると，その内容を自動的に書き起こし，関連する画像を自然言語処理とテキストマイニングを活用しながら探し出して動画を生成した．最近の例では，楽曲を与えると，その内容に合ったカメラワークのシーケンスを生成する研究 [20] があり，コンサートを多点撮影した動画からマッシュアップを自動生成するような目的で活用できる．本研究は，これらの手法と併用でき，静的なコンテンツでは不可能なユーザとコンテンツのインタラクションを切り拓くものである．

3.3 クリエイティブコーディング

マルチメディアコンテンツを生成したりインタラクション可能にしたりするためのプログラミングを支援するツールは数多く存在する．そうしたツールを用い，芸術的な目的で創造的にプログラミングすることは，しばしば「ク

リエイティブコーディング (creative coding)」と呼ばれる．同様の活動は，応用領域や文脈によって「ジェネラティブアート (generative art) [21]」「アルゴリズミックミュージック (algorithmic music) [22]」「ジェネラティブデザイン (generative design)」「プロシージャルモデリング (procedural modeling)」などの別名で呼ばれる．

Processing [23] と openFrameworks [24] は著名なクリエイティブコーディングツールであり，インタラクティブなアプリケーションをプロトタイピングするためによく使われる．また，美しい画像や動画，インタラクティブなグラフィックを描画するためにも使われる．クリエイティブコーディング用のライブラリには p5.js のように Web アプリケーションを構築できるものもある．こうしたライブラリは，Web ブラウザを備えた，スマートフォンをはじめとする端末で実行できるアプリケーションを容易に開発できるため，人気を博している．こうしたライブラリはリリックアプリの視覚表現を実装する助けになるが，プログラマによって好みが分かれる．そこで本研究では，プログラマが好みのクリエイティブコーディング用ライブラリを選べるように，フレームワークを設計している．

これまで研究者たちは，Web を使って新たな創造的表現形式を実現してきた．D3.js [25] はデータ可視化のために開発され，Idyll [26] などの後続研究はユーザがインタラクション可能な Web 上の記事を構築するワークフローを探求してきた．Webstrates [27] は，共有可能な動的メディアを Web 上に実現し，同時に協力してデータを編集したり，インタラクションをプログラミングしたりできるようにした．Videostrates [28] と Codestrates [29] は，Webstrates をベース手法としながら，それぞれ動画編集や文芸的プログラミングに特化したインタラクションデザインを提案した．本研究はこれらの先行研究と同様に，Web を使った創造的な表現の可能性を切り拓くものであり，とくに，リリックアプリという新たな表現形式のデザイン空間を明らかにするために，フレームワークを一般公開してきた．

楽曲を活用したインタラクティブなアプリケーションに関しては，これまでにも商業楽曲作品がスマートフォンアプリとしてリリースされたことがある．Björk の Biophilia [30] というアルバムは曲ごとにアプリ化され，ユーザがタッチ操作で音や映像を動的に編集できるものがあった．Brian Eno の Reflection [31] は，起動するたび異なるバージョンの曲が流れるため，誰もが自分自身の Reflection を楽しめる (他の事例は Levtov の「大量消費を可能にするアルゴリズミックミュージック」の記事 [5] を参照のこと)．しかしながら，こうした作品はプログラマが一般的なプログラミング用のツールを用いて開発している．対照的に，RjDj [32] は，スマートフォン上で Pure Data によって制作されたアルゴリズミックミュージックを演奏・編集できるようにしたもので，プログラマがインタラクティブな音楽を制作，開発してマス向けに配信する手段を実装したツールとして最初の試みの一つである．最近の例では variPlay [33] があり，これはアルゴリズミックミュージックではなく，事前に録音された複数トラックの楽曲をカスタマイズして再生できるものである．これらの先行事例と異なり，本研究では楽曲そのものの音を編集することを目的としていない．その代わり，既存の楽曲に対してインタラクションする機能を追加することで，新しい歌詞駆動の芸術的体験を創り出せるようにする．

なお，我々が研究開発して公開中の Web ベースの大規模音楽連動制御プラットフォーム「Songle Sync」 [34][*5] では，楽曲再生に合わせて多種多様な機器でのマルチメディア演出を同時制御できる．我々は，Songle Sync およびその先行研究である Songle Widget [35][*6] において，音楽的要素と連動する演出を手軽にプログラミングできるイベント駆動型の API を提供した．ただし，利用可能な音楽的要素はビート構造，楽曲構造 (サビ区間，繰り返し区間)，コード進行などであり，歌詞関連の情報は利用できない．また，後述 (4.2.2 節) のとおり，イベント駆動型の API には高度な演出を開発しづらく，時間精度が落ちやすい問題がある．本研究は，これらの技術的な制約を解消する API を新たに提案するものである．

4. Lyric App Framework

2.3 節で整理したとおり，プログラマはリリックアプリ開発において 3 つの課題に直面する．本章では，これらを解決する Lyric App Framework を提案する．本フレームワークは，各課題を解決する 3 つの主要機能で構成され，一貫した開発体験をプログラマに提供する (図 1)．

4.1 Web ベースの開発ワークフロー

まず，楽曲中の音楽的要素と同期した演出開発を容易にするため，Web ベースのワークフローを提案する．このワークフローでは，開発中いつでも新たな楽曲を扱えるようにしたり楽曲を差し替えたりでき，プログラマやミュージシャンのニーズに柔軟に対応できる特長がある．

4.1.1 自動解析とクラウドソーシング

本研究で提案するフレームワークは，我々が研究開発してきた能動的音楽鑑賞サービス「Songle」 [36, 37][*7] の機能を拡張することで実現しており，楽曲と歌詞テキストの自動対応付けを行い，楽曲の内容を自動解析してリリックアプリ開発に役立つ情報 (表 1) を提供する．さらに，必要に応じて情報の誤りを修正できる．

プログラマやミュージシャンのワークフローは以下のとおりである．まず，フレームワークの Web インタフェース

[*5] https://api.songle.jp/sync
[*6] https://widget.songle.jp
[*7] https://songle.jp

表 1: リリックアプリがアクセスできる解析結果の一覧

解析結果	説明
歌詞タイミング	歌詞テキストの各フレーズ，単語，文字が発声を開始し，終了するタイミング情報
品詞	歌詞テキストの各単語の品詞
声量	歌声の音量の時系列変化
音楽印象	音楽の印象の移り変わりを表す V/A(valence arousal) 平面上の座標値の時系列変化
ビート構造	四分音符に相当する各ビートや，各小節の先頭のタイミング情報
コード進行	コード名と各コード区間のタイミング情報
楽曲構造	サビ区間や，楽曲中の繰り返し区間のタイミング情報

を通してリリックアプリで利用したい楽曲と歌詞テキストの組を登録する．楽曲と歌詞テキストは新規にアップロードすることもできるし，すでに Web 上にあって二次利用に関する適切なライセンスが付与されているものなら URL を指定するだけでもよい．登録から約 10 分後には，自動解析の結果がフレームワークの Web サーバ上で利用可能になり，リリックアプリ上で読み込めるようになる．

その後，歌詞を含む音楽的要素の自動解析されたタイミング情報を Web インタフェースで確認し，必要に応じて修正できる．こうしたクラウドソーシングによる自動解析結果の修正はすでに Songle 上で可能であったが，我々は本研究や TextAlive の研究で，歌詞タイミングの修正も可能にしたほか，歌詞単語の品詞の自動検出や，タイミング情報のリビジョン管理まで可能な拡張を施した．

4.1.2 キーとしての楽曲 URL

本フレームワークでは，楽曲の URL が歌詞テキストと解析結果を取得するためのキーとなる．すなわち，リリックアプリで利用する楽曲はそれぞれ固有の URL を持っているという前提に立っている．プログラマは，楽曲 URL のリストを保持することでユーザに楽曲を選ばせることができ，単一のリリックアプリで複数の楽曲を開くことが可能となる．楽曲 URL だけを指定した場合，解析結果 (あるいはユーザによる修正結果) の最新版が読み込まれるが，さらに数値型のリビジョン番号を指定することで，特定の版のデータを取得できる．これにより，クラウドソーシングで仮に悪意ある修正が行われた場合にも影響を受けずに済み，リリックアプリの動作の再現性を向上できる．このように，コンテンツの URL をキーとして用い，さらにオプションで解析結果のリビジョン番号を指定できるようにする手法は，シンプルながら強力である．本フレームワークに限らず，静的メディアにインタラクションを追加する今後の研究でも有用であろう．

さらに，本フレームワークでは，楽曲と歌詞テキストの組を登録するタイミングはいつでもよく，柔軟な開発ワークフローが可能となっている．例えば，暫定的に既存の楽曲 URL を用いて開発を始めたリリックアプリにおいて，後に新しく楽曲が登録された場合，ソースコード中の楽曲 URL を書き換えるだけで，リリックアプリが新しい楽曲で動作するようになる．また，ミュージシャンは，未リリースの楽曲について，他の人がアクセスできないプライベート楽曲の URL を用いてリリックアプリの動作を確認できる．リリース時には，楽曲 URL を一般公開されたものに差し替えてリリックアプリを配信し，楽曲のプロモーションに役立てることができる．

4.2 リリックアプリのインタラクションデザイン用 API

次に，ユーザと楽曲・歌詞の間のインタラクションデザインを支援するため，プログラマ向けの TextAlive App API[*2]を提案する．本 API は楽曲再生の状態を管理できるイベント駆動型の API と，楽曲再生と精密に同期した演出を表示できる時刻駆動型の API に大別される．

4.2.1 状態管理用のイベント駆動型 API

リリックビデオは常に単一の課題曲と紐づいているが，リリックアプリは単一の楽曲向けのこともあれば，特定の楽曲セットで実行できたり（例: ユーザがプレイする曲目を選べるリズムゲーム），任意の楽曲で実行できたり（例: インタラクティブなビジュアライザのついた音楽プレイヤー）する．さらに，リリックアプリは，特定の操作をユーザに許可したり禁止したりできる．例えば，音楽プレイヤーは楽曲再生を一時停止したり，任意の再生位置にシークしたりできるが，ゲームはどちらの操作も禁止していることが多い．楽曲の物語性を強調するアプリケーションでは，一時停止は可能でも，シークは禁止されていたりする．

これらすべてのインタラクションシナリオの実装を支援するために，我々はリリックアプリが取りうる状態の一覧を図 2のように整理して定義した．TextAlive App API では，これらの状態間の遷移を通知するイベント駆動の API を実装している．具体的には，API の JavaScript クラス `Player` が，Lyric App Framework の機能を利用する際のエントリーポイントとなっている．

リリックアプリの起動後，フレームワークのサーバに接続できると，`onAppReady` イベントが発行される．もし単一の楽曲向けのリリックアプリを開発したけれ

図 2: 本フレームワークは，リリックアプリの複雑な状態遷移を管理するためのイベント駆動型 API を提供する．

ば，プログラマはこのイベントのコールバック関数内で `player.createFromSongUrl(songUrl)` という API を呼び出して楽曲を明示的に読み込めばよい．ここで読み込みが行われなかった場合，リリックアプリは，Lyric App Customizer と呼ばれる Web インタフェースで読み込むべき楽曲が別途指定されていないか確認する．本 Web インタフェースを使うことで，プログラマやミュージシャンは，開発とデプロイが終わったリリックアプリの振る舞いを事後的にカスタマイズできるようになっている (4.3.2 節で詳述)．Customizer でも読み込むべき楽曲が指定されていなかった場合，リリックアプリは待ち受け状態に入り，プログラマは任意のタイミングで先述の API を呼び出して楽曲を読み込める．こうして，ユーザに楽曲リストを提示して一曲を選んでもらうようなインタラクションが実現する．API を複数回呼んだ場合，最後に指定された楽曲が読み込まれる．これにより，複数の楽曲を切り替えて再生するリリックアプリが実装できる．

楽曲の読み込みが終わると，楽曲の再生状態を制御できるようになる．ユーザが再生を一時停止したり再生位置をシークしたりできるようにするか否かはプログラマに任されており，インタラクションシナリオに応じて自由に決められる．楽曲の再生状態が変化した場合には，さまざまなイベントが発行される．例えば，`onPlay`，`onPause`，`onSeek`，`onTimeUpdate` などがある．

4.2.2 時間精度を要するインタラクションデザイン用の時刻駆動型 (time-driven) API

グラフィカルユーザインタフェース (Graphical User Interface; GUI) 用のフレームワークや，我々が研究開発した音楽連動制御のためのプラットフォーム「Songle Sync」[34] では，イベント駆動型の API によって，時系列に発生するイベントに応じたインタラクション設計が行えるようになっている．しかし我々は，リリックアプリ用の API を設計するにあたり，次に述べる 3 つの理由から，音楽的要素のイベントをサポートしないことに決めた．

まず，イベント駆動型の API において，イベントは通常何かが起きた直後にのみ発行されるため，未来のイベントに向けた準備をコーディングする目的には向いていない．例えば，歌詞の単語の発声開始イベントを使って，発声よりも充分前に画面内に滑り込み始めるような予備動作を伴うアニメーションの設計には使えない．次に，イベントは個々の音楽的要素に紐づいて提供されるが，魅力的な視覚表現は往々にして複数の音楽的要素と連動しているものである．例えば，歌詞のフレーズが発声するタイミングでその周囲に現れる水紋のような視覚表現を考える．水紋はビートに合わせてフェードインとフェードアウトを繰り返すかもしれないし，サビの歌詞かどうかによって色を変えたくなるかもしれない．イベント駆動型 API では，フレーズの発声開始，ビート，サビ区間といったすべてのイベントについてリスナを定義し，情報をどこか別のロジックで集約してアニメーションさせる必要が生じて不便である．最後に，イベント駆動型 API では時間精度が低くなりがちである．こうした API を提供するためには，フレームワーク内にイベントループを設け，現在の再生位置と各種イベントの時刻情報を比較し続けるという実装が典型的である．この場合，各種イベントは最悪値でイベントループの実行間隔分，遅れて発行される可能性がある[*8]．このように時間精度を保証できない API は，求める表現を得るうえで不充分なことがあり，時間精度が意図せずに低くなった場合に初学者が混乱する可能性もある．

これらの課題を解決する (すなわち，未来に向けた準備のロジックを容易に設計でき，複数の音楽的要素の情報を考慮に入れやすく，高い時間精度を実現できるようにする) ために，我々は，引数にミリ秒単位の再生位置の数値を取り，その引数で指定された時点での音楽的要素を開始・終了などの時刻情報付きで返す「時刻駆動型 (time-driven) API」を提案する．例えば，5 秒後に発声が始まる歌詞フレーズの情報を取得するには `player.getPhrase(player.position + 5000)` という API を呼び出せばよい．さらに，同じ引数で `getBeat` という API を呼ぶなどすれば，フレーズ，ビート，サビ区間の情報をすべて考慮に入れた視覚表現を実装できる．イベント駆動型と時刻駆動型 API の図入りの詳しい説明は，別途，付録 A.1に示した．

時刻駆動型 API は，インタラクティブな視覚表現を生成する際に役に立つ，GUI やクリエイティブコーディング用のライブラリと相補的な役割を果たす．こうしたライブラリでは，通常，描画すべき内容を定義するレンダリング用の関数をプログラマが定義するようになっている．例えば，Web API における `requestAnimationFrame()`，React における `Components.render()`，p5.js における `loop()`，Three.js における `renderer.render()` などが挙げられる．プログラマは，これらの関数内で時刻駆動型 API を呼ぶことで，複合的な音楽的要素を考慮した視覚表現を設計できる．描画内容は，常に楽曲再生の最新の状態を反映していることが保証される．このように，本 API はプログラマが好みの使い慣れたライブラリと併用できる特長がある．

4.3 リリックアプリの大規模配信

最後に，ユーザにリリックアプリを届ける方法が自明でない課題に関して，本フレームワークは，プログラマやミュージシャンがリリックアプリを Web サイトとして安定的に大規模配信できる仕組みを提供する (図 3)．

[*8] 理論的な限界に加え，現在の Web ブラウザの実装では，CPU が過負荷になった場合，ブラウザのタブが非アクティブになった場合，低バッテリーで省電力モードになった場合など，さまざまな要因によってプログラムの実行タイミングが影響を受けるという実用上の限界もある．

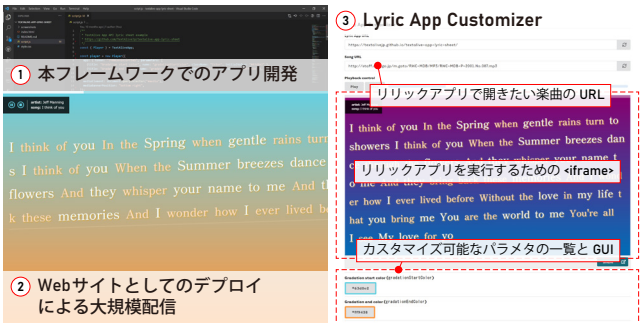


図 3: 本フレームワークを用いれば，リリックアプリを Web サイトとして大規模配信できるだけでなく，事後的にアプリをカスタマイズして新しい楽曲に対応させたりできる．

4.3.1 プロダクションレディなアプリ配信

一般に，インタラクティブなメディアコンテンツの配信は 2 つの観点で難しい．まず，ユーザを取り巻く情報環境は千差万別であり，多くのユーザがインタラクションを楽しめるような相互運用性を担保する必要がある．そして，コンテンツの利用者数は時と場合によって大きく上下するため，一度に大量のアクセスがあった場合などにも対応できるスケーラビリティを確保しなければならない．我々は，リリックアプリという新たな表現形式を提案するにあたり，プロトタイプ開発ができるだけでなく，これらの課題解決まで考慮に入れ，高い信頼性でのアプリの大規模配信までサポートしたいと考えた．

そこで我々は，インタラクティブミュージックの大量流通に関する議論 [5] を参考にして，現状もっとも現実的な解は，アプリを Web 標準に則った Web サイトとして配信できるようにすることだと考えた．本フレームワークを使えば，プログラマが Web サイトを構築する技術さえ持っていれば，上記のような大量流通が可能なプロダクションレディのリリックアプリを実装できる．

リリックアプリは，最小限のケースでは HTML/-JavaScript ファイルを配信できる静的な Web サイトだが，プログラマのスキルとインタラクションシナリオのニーズ次第では，スタイルシートやメディアコンテンツ，サーバサイドのコードなどを含んだサイトとして実装することも可能である．この際，プログラマは，楽曲，歌詞テキスト，音楽的要素に関する解析情報をスケーラブルに配信することに気を揉む必要はない．なぜなら，こうしたデータは，我々のフレームワークのサーバか，楽曲や動画の共有サービスからユーザの手元へ直接届けられるためである．アプリの可用性や読み込み速度をさらに向上する必要があれば，すべてのデータを単一の Web サイトにまとめ，コンテンツ配信ネットワーク (Content Distribution Network; CDN) 上に配置することも可能である．

4.3.2 Lyric App Customizer

本フレームワークは，プログラマやミュージシャンがリリックアプリの挙動をカスタマイズできるよう，TextAlive Lyric App Customizer[*9] という Web インタフェースを備えている (図 3)．Customizer のページでリリックアプリの Web サイトの URL を指定すると，ページ内の `<iframe>` コンポーネント内でアプリが起動し，フレーム間通信の仕組みを通してアプリが制御される．例えば，4.2.1 節で紹介した音楽プレイヤーのように，楽曲を自由に指定できるリリックアプリの場合，ミュージシャン側で楽曲の URL をカスタマイズできる．カスタマイズ結果は，最終的にはリリックアプリの URL に与えるクエリパラメータとして，`?ta_song_url=NewUrl` のように永続化できる．

また，楽曲 URL の他にも，プログラマは自由にカスタマイズ用パラメタのリストを定義することができる．例えば，現在の実装では，数値や色，文字列 (自由入力または選択式) といったパラメタがサポートされている．定義されたパラメタの一覧は，プログラマやミュージシャンがインタラクティブに調整できるように，Customizer 上でスライダーやカラーパレットなどの GUI として表示される．GUI が操作されてパラメタ値が調整された場合，リリックアプリ側では `onParameterUpdate` イベントが発行されるため，これに対応するリスナをプログラマが実装することで，アプリの振る舞いをインタラクティブにカスタマイズできるようになる．ミュージシャンは，Customizer でリリックアプリを自分好みにカスタマイズすることによって，より効果的な楽曲のプロモーションに繋げることができる．

5. 実装

本章では Lyric App Framework の実装について説明する．なお，本フレームワークは「TextAlive for Developers」[*3] という Web サイトで一般公開済みである．

5.1 フレームワークの Web サーバ

4.1 節で紹介した Web ベースのワークフローを実現するために，REST (REpresentational State Transfer) モデルの API を提供する Web サーバと，Web インタフェースを提供するためのクライアントアプリケーションが動作する Web サーバの 2 種類を実装した．これらのサーバは共にリリックアプリの情報（作者名，URL など）を保存するためのデータベースと接続されている．また，REST API を提供する Web サーバは，我々が研究開発してきた Songle [36, 37] のサーバや，その他の音楽的要素の情報を提供する外部サーバとも接続されている．本フレームワークと TextAlive のために，歌詞タイミングや歌詞単語の品詞の解析機能，タイミング情報のリビジョン管理，プライベート楽曲と歌詞のアップロードを実現するためのストレージ機能を，これらの外部サーバに新規実装した．歌詞

[*9] https://developer.textalive.jp/app/customize

タイミングの解析は当初，既存研究 [14] に基づいた技術で実現したが，後により精度の高い新規技術に差し替えた．また，単語の品詞解析は，日本語では Mecab [38] を，英語では Natural Language Toolkit (NLTK) [39] を用いた．

現在の実装では，歌詞は日本語，英語，日英混合を扱える．また，楽曲に関しては 10 分以内のものを扱える．解析結果は後述する Web インタフェースで修正できる．修正結果は数値のリビジョン番号とともにサーバに記録され，4.1.2 節で紹介したように，プログラマがその番号を指定することでリリックアプリの動作の再現性を向上できる．

我々は，本フレームワークの一般公開にあたり，当初は日本のコミュニティで活用されることを想定したため，日本語または日英混合で書かれた歌詞テキストをサポートした．日本語は文字ごとに発声する音が明確な場合が多く，文字ごとのタイミング情報を提供する合理性は高い．対照的に，英語は文字ごとのタイミング情報を定義できない場合が多い．そこで，現在の実装では，日本語は文字レベルでタイミング情報を解析する一方，英語は単語レベルまでタイミング情報を解析している．英語の文字ごとのタイミング情報は，便宜的に，単語レベルのタイミング情報を線形補完したものを提供している．

5.2 フレームワークの Web インタフェース

Web インタフェースには 3 つの役割がある．まず，フレームワークの Web ベースのワークフロー (4.1 節) のうち，楽曲情報を管理する機能を提供する．すなわち，Web 上の一般公開楽曲と歌詞の URL を登録して解析したり，新しく楽曲の MP3 ファイルと歌詞のプレーンテキストをアップロードして解析したり，これらの楽曲の解析結果を閲覧したり修正したりできるようにする．次に，6.1.2 節で後述するとおり，フレームワークのチュートリアルや API リファレンスなどの参考資料を提供する．最後に，Lyric App Customizer (4.3.2 節) を提供する．以下，Customizer の使い方とその実装手法について詳述する．

まず，プログラマは Customizer でリリックアプリの URL を指定する．この際，必要に応じてリリックアプリで開きたい楽曲の URL も指定できる．次に，Customizer はリリックアプリを `<iframe>` コンポーネント内で起動する．その際，楽曲 URL が指定されていた場合はリリックアプリに渡すクエリパラメタで指定する．そして，リリックアプリ側でフレームワークのクライアントライブラリ (5.3 節) が REST API サーバへの接続などを行ったうえで `onAppReady` イベント (図 2) を発行し，楽曲が再生可能な状態となる．Customizer はその後もフレーム間通信の Web 標準 API である Web Messaging API を用いてリリックアプリと情報のやり取りを続け，再生位置やカスタマイズ用のパラメタ値を更新し続ける．Customizer 側に表示されているカスタマイズ用 GUI が操作された場合は，リリックアプリ側で `onAppParameterUpdate` イベントが発行され，プログラマの処理によってカスタマイズ結果がインタラクティブに反映される．

5.3 リリックアプリ開発用クライアントライブラリ

リリックアプリ開発用の TextAlive App API[*2]は JavaScript 用クライアントライブラリとして実装してあり，“npm” パッケージをインストールするか `<script>` タグで読み込むことにより利用可能となる．

リリックアプリで楽曲を読み込んだあと利用できるようになるメディア情報には，フレームワークの Web サーバに登録したときに入力されるメタ情報 (楽曲 URL，歌詞 URL，アーティスト名，楽曲タイトルなど) と，Web ページ上に埋め込まれるメディアから取得できる実行時情報 (楽曲長など) がある．また，リリックアプリで読み込める楽曲は現状，Web 上の MP3 ファイル (フレームワークの Web サーバにアップロードされたプライベート楽曲か，ミュージシャンやプログラマが自身でホストしている楽曲) と，動画共有サービスにアップロードされた動画 (YouTube とニコニコ動画) である．TextAlive App API は，これらの楽曲の再生操作に必要な API (MP3 ファイルに関しては HTML DOM API，YouTube 動画に関しては YouTube Player API，ニコニコ動画に関してはニコニコ動画プレイヤー API) の差分を吸収するラッパーライブラリを提供する．一部の API は再生位置に関する情報を定期的にアップデートする仕組みになっているため，ラッパーライブラリの側では，アップデート処理が実行されるたびに再生位置と UNIX タイムスタンプの情報を保持しておき，プログラマが再生位置を取得しようとした際には，これらの情報をもとになるべく精確な再生位置情報を補完して提供できるようにしている．

フレームワークの音楽的要素の解析モジュールは，種類ごとに異なる時間精度を持っており (例えば歌詞タイミングは 10 ミリ秒，音楽印象は 15 秒など)，元データの精度によらずミリ秒単位で情報にアクセスできるような補完処理はクライアント側で行われる．具体的には，TextAlive App API で各音楽的要素を表すクラスを定義しており，指定された再生位置に対し近傍の解析結果を線形補完して返す処理をクラス内に実装した．

時刻駆動型 API は画面の更新頻度に合わせてリリックアプリ実行中に大量に呼ばれるため，充分高速に結果を返せるよう，事前に時刻順にソートした音楽的要素のインスタンスの配列からバイナリサーチを行うように実装した．

6. プログラミング・コンテストでの実証実験

リリックアプリという新たな表現形式のデザイン空間を明らかにするために，我々は 2020 年 9 月 18 日に TextAlive App API を含む Lyric App Framework を一般公開し，実

証実験として年次のプログラミング・コンテストを開催してきた．本章では，プログラミング・コンテストについて紹介するとともに，2 年分の応募作品 52 件を分析して判明したリリックアプリのカテゴリ 8 種類などの結果を報告する．また，利用統計やソースコード分析，質的ユーザ評価の結果を紹介し，本フレームワークの有効性を検証する．

6.1 プログラミング・コンテストの概要

プログラミング・コンテストは，年次の大規模イベント『初音ミク「マジカルミライ」』の一環として開催された．本イベントは著名な歌声合成ソフトウェアでありバーチャル・シンガーである「初音ミク」を生み出したクリプトン・フューチャー・メディア株式会社が主催し，「初音ミクたちバーチャル・シンガーの 3DCG ライブと，創作の楽しさを体感できる企画展を併催したイベント」 [40] である．

本研究において，本コンテストでの実証実験には 3 つの目的がある．まず，創作文化を楽しみ，創作文化に貢献する人々が集まる場となっているイベントでコンテストを開催することで，未来志向のプログラマとともにリリックアプリのデザイン空間を探索することである．次に，本フレームワークが幅広いプログラミング経験を持つプログラマを効果的に支援できるか検証すること，そして，リリックアプリのコミュニティを活性化することである．

6.1.1 コンテストのルールと進め方

コンテストは「初音ミク『マジカルミライ 2020』プログラミング・コンテスト」[41] および「初音ミク『マジカルミライ 2021』プログラミング・コンテスト」[42] と呼称され，「マジカルミライ」イベント会場での入選作品紹介[*10]と受賞作品発表[*11]を除けばオンラインで進行した．主催側でリリックアプリの開発に必要な権利処理が済んでいる楽曲と歌詞を課題曲として提供することで，参加者が開発に集中できるようにした．コンテストの応募期間中，本フレームワークのハンズオン (実習を伴う体験型イベント) は対面とオンラインの別を問わず開催しなかった．その代わり，6.1.2 節で後述する参考資料をオンラインで提供した．Twitter[*12]や Gitter[*13]でも参加者からの質問に適宜回答した．コンテストの応募に際して (参加は無料)，参加者は静的 Web アプリケーションのソースコードとビルド手順を提出した．サーバ側のプログラムを含まない静的アプリケーションに限ったのは，応募作品を主催側の Web サーバに設置して誰でも楽しめるようにするためである．

各年，応募を締め切ってからは，主催側の審査で入選作品を選定し，入選作品発表とともにオンラインでの一般投票を受け付けた．投票は 1 人 1 回 1 作品まで，スマートフォン用アプリ「ミクナビ」のユーザであれば誰でも行えた．プログラミング・コンテストの Web サイト上では，投票に際してユーザが実際に試せるよう，すべての入選作品をデプロイし，その動作を録画したデモ動画も掲載した．また，「マジカルミライ」イベント会場 (大阪) での「入選作品紹介」のステージ[*10]上で，著者ら (加藤，後藤) とコンテストを主催するクリプトン・フューチャー・メディア株式会社の代表取締役である伊藤 博之氏が，パネル形式で入選作品を紹介して投票を呼び掛けた．1 週間弱の一般投票の期間が終わってから，投票結果を踏まえて主催側で改めて審査を行い，受賞作品 (最優秀賞 1 件，優秀賞 3 件) を選定した．そして，後日の「マジカルミライ」イベント会場 (東京) での「受賞作品発表」のステージ[*11]の後，受賞したチームには賞状，グッズ詰め合わせなどが贈られた．受賞作品発表後に，事後アンケートを参加者にオンライン配布しており，その結果は 6.3.3 節で報告する．

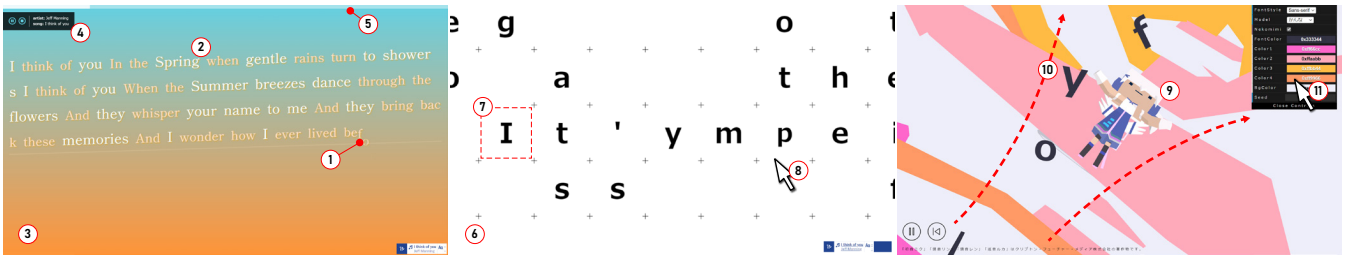
なお，主催側の審査の評価軸は，クリエイティビティ (Web アプリケーションの演出が楽曲と同期して魅力的に見えるか)，イノベーション (Web アプリケーションを支えるアイデアがユニークで，未来の創作文化を予感させるものであるか)，完成度 (Web アプリケーションが一般的な Web ブラウザで正常に動作するか，実装が技術的に優れているか) であった．3 つの評価軸はコンテストの公式 Web サイトに明記され，参加者はいつでも確認できた．

6.1.2 フレームワークの参考資料

我々は，Lyric App Framework を公開する際に，チュートリアルや API リファレンス，スライドの読み上げとデモを交えた説明動画などの参考資料を Web サイト[*3]に掲載した．チュートリアルはフレームワークの概要を紹介するとともに，リリックアプリの開発を始めるための手順を説明する内容であった．さらに我々は，フレームワークの使い方をデモするために 11 件のリリックアプリを開発し，GitHub 上で MIT License のサンプルコードとして公開した[*14]．ここでは，うち 3 件を紹介する (図 4)．

1 つ目の例であるインタラクティブ歌詞カードでは，楽曲再生の進行とともに歌詞テキストが CSS アニメーションで徐々に表示される ①．美しく読みやすい歌詞を表示するために，歌詞テキストは単語の品詞に応じた色と大きさで表示される ②．ユーザは，歌詞を読むだけでなく，単語をクリックしてそこから再生できる．Lyric App Customizer を使えば，背景の色味などを変更でき，カスタマイズした歌詞カードを他の人と共有できる ③．このリリックアプリには他にも，基本的な再生操作ができ，楽曲や歌詞の作者情報を表示するユーザインタフェースが備わっている ④．画面上部には半透明なシークバーもついている ⑤．

[*10] https://youtu.be/-t9AVVgZo5k (2020)
https://youtu.be/hvmDxCeyWU8 (2021)
[*11] https://youtu.be/Yz4Ucrw95bQ (2020)
https://youtu.be/MkILA_NC1T0 (2021)
[*12] https://twitter.com/TextAliveJp
[*13] https://gitter.im/textalive-app-api/community
[*14] https://github.com/TextAliveJp



A インタラクティブ歌詞シート　B 歌詞タイル　C ダンスするリリックアプリ

図 4: 本フレームワークで実装したサンプルコード 11 件中 3 件のスクリーンショット．(全件の概要は付録 A.2を参照．)

理解しやすいコードを書くことを心がけてコメントも多く挿入したが，本アプリは簡潔に 237 行 (コメントを除けば 160 行) で実装されている．

2 つ目の例である歌詞タイルは，HTML5 Canvas API で画面全体に正方形のタイルで構成される二次元グリッドを表示する ⑥．各タイルには歌詞テキストは 1 文字しか表示できない ⑦．そこでユーザがマウスかタッチ操作で二次元グリッド上を滑らかに移動すると，歌詞の文字が発声の都度，画面中央近傍の空いているタイルに表示されていく ⑧．ユーザは，二次元グリッド上をさまざまな移動経路で動き回ってタイルを歌詞で埋めていく．楽曲を再生する度に違う経路になるので，さながらピクセルアートを描くような創造的な体験ができる．

3 つ目の例であるダンスするリリックアプリは，「ちびキャラ」と呼ばれるスタイルのキャラクタがステージ上で楽曲に合わせて踊る様子を Three.js を使って実現したものである ⑨．画面下部から浮かび上がってくるパーティクル効果と歌詞テキストの文字オブジェクトがステージを盛り上げてくれる ⑩．これらの動きのパターンは，ビートや楽曲の繰り返し構造などを反映して動的に生成されている．Customizer を使えば，キャラクタを切り替えたり，ステージ上の配色を変更したりできる．Customizer が接続されていない場合 (すなわち，このリリックアプリが直接 Web サイトとして読み込まれた場合) でも，これらのパラメタを変更できるユーザインタフェースが表示される実装になっており，ユーザがアプリを常にインタラクティブに楽しめるよう配慮されている ⑪．

我々はこれらのリリックアプリがフレームワークの代表的な機能を活用したものになるよう注意深く設計したが，同時に，サンプルコードとして多くの人々の環境で動作するよう，スマートフォンや PC，タブレット端末など多様な Web ブラウザ上で動作する互換性にも気を配った．そのため，実験的な Web 標準 API (位置測位，加速度センサ，Bluetooth など) は利用していない．

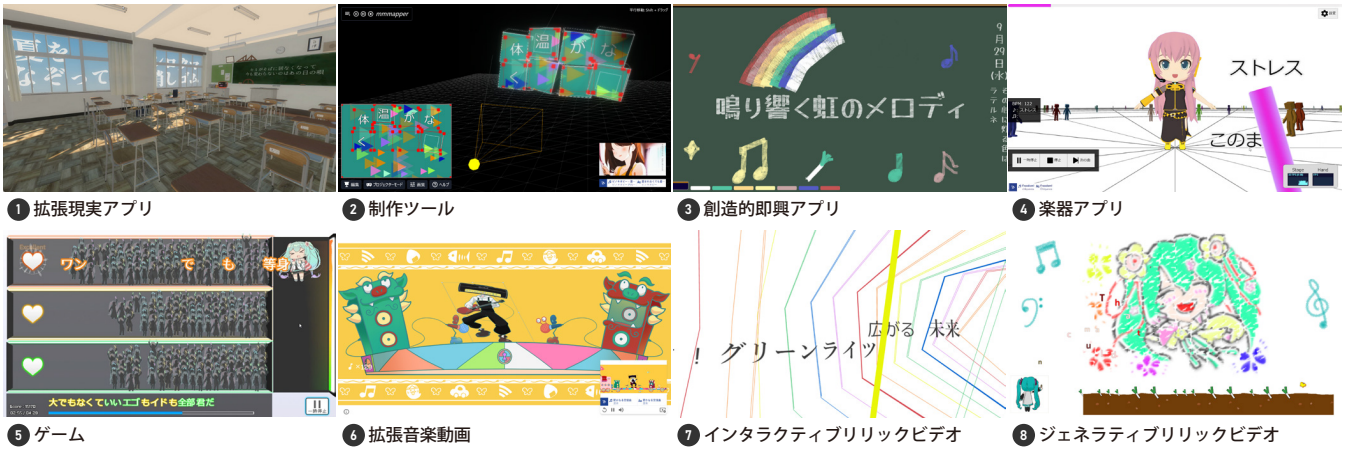
6.2 リリックアプリの 8 種類のカテゴリ

2 回のプログラミング・コンテストはそれぞれ 47 日間と 77 日間にわたって応募期間が設けられ，32 件と 20 件の応募があった．提出された全 52 件のリリックアプリはすべて完成して実行可能な状態であった．それらのリリックアプリの驚くほど豊かなデザイン空間を理解するため，我々は以下の手順で分類した．まず，各リリックアプリの内容を説明する短い文章を書いた．そして，これらの説明文に繰り返し現れるキーワードから，あまりに一般的な用語を除いた．例えば，歌詞テキストがアニメーションする説明は繰り返し出現するので除いた．この時点で，拡張現実 (extended reality)，ジェネラティブリリックビデオ (generative lyric video)，ゲーム (game) という 3 つのカテゴリが得られた．次に，この 3 カテゴリに収まらないリリックアプリの説明文を注意深く分析し，創造的アプリケーション (creative application)，楽器 (instrument)，拡張音楽動画 (augmented music video) という 3 つのカテゴリを得た．この時点ですべてのリリックアプリが 6 カテゴリに収まったが，いくつかのカテゴリに関してはユーザがインタラクションできる手段をもとにサブカテゴリに分けたほうが的確に分類できることが分かった．具体的には，ジェネラティブリリックビデオにはユーザの入力を何も受け付けないものもあれば，インタラクティブなものもあり，後者をインタラクティブリリックビデオ (interactive lyric video) というカテゴリに分けた．また，創造的アプリケーションにはシークや巻き戻しの操作を許さず即興的な体験を重視するものもあれば，楽曲を聴き込みながら何らかのコンテンツを作り込んでいく制作ツール (authoring tool) と呼ぶべきカテゴリのものもあった．

こうして我々は，リリックアプリのカテゴリとして 8 種類を発見した．52 件の応募作品は，9 件の拡張現実アプリ，6 件の制作ツール，6 件の創造的即興アプリ，3 件の楽器アプリ，6 件のゲーム，2 件の拡張音楽動画，5 件のインタラクティブリリックビデオ，16 件のジェネラティブリリックビデオに分類された．

6.2.1 拡張現実アプリ (extended reality)

このカテゴリは，仮想的な 2D・3D の情景における没入感のある体験に重点を置いている．カメラ入力を活用して，実世界の情景に歌詞駆動の視覚的演出レイヤーを追加するものもある．ユーザは楽曲の世界観の中にいるように感じられる．例えば，特定の楽曲のために作り込まれた，



① 拡張現実アプリ　② 制作ツール　③ 創造的即興アプリ　④ 楽器アプリ
⑤ ゲーム　⑥ 拡張音楽動画　⑦ インタラクティブリリックビデオ　⑧ ジェネラティブリリックビデオ

図 5: プログラミング・コンテスト応募作品の分析で判明したリリックアプリのカテゴリ 8 種類と，その代表例のスクリーンショット．付録 A.3に 12 件を追加掲載．(スクリーンショット提供: クリプトン・フューチャー・メディア株式会社)

学校の教室の中を自由に動き回れる作品 (図 5 ①) があった．この作品では，教室の中に歌詞のストーリーと関連のある隠し要素が配置されており，これを見つける楽しみ方もできるようになっていた．別の作品では，WebVR API を使って VR ヘッドセットで仮想空間に飛び込めるものや，AR.js を使ってカメラ画像の手前に歌詞がタイミングよく表示される立方体を映し出すもの，Tensorflow.js を使ってユーザの姿勢情報を取得し，カメラ画像の上に視覚的なエフェクトを表示するものなどがあった．

6.2.2 制作ツール (authoring tool)

このカテゴリは，楽曲と同期した派生コンテンツを制作し，作り込み，場合によっては他の人と共有することまで可能にする．例えば，リリックビデオを構成する視覚的要素を画面上に配置していくことで，音楽動画を作り上げることが可能な作品があった (図 A·3 ⑫)．視覚的要素は，歌詞テキストが表示されるプレースホルダやパーティクル効果，キャラクタがアニメーションする画像などの中からユーザが自由に選べるようになっていた．さらに複雑な例として，プロジェクションマッピング用のコンテンツを制作できる作品があった (図 5 ②)．ユーザは，曲ごとの内容を踏まえて定義された仮想的な 3D 空間で，特定の平面上に視覚的要素を配置していくことができた．こうして作り上げた 3D の情景は，実世界で同様の平面を再現すれば実際のプロジェクションマッピングに活用できるようになっていた．さらに別の作品では，3D 空間内でキャラクタが楽曲を歌唱しながらアニメーションしており，さまざまな設定項目でその様子をカスタマイズできるようになっていた (図 A·3 ⑪)．

6.2.3 創造的即興アプリ (creative application)

このカテゴリは，ユーザが再生中の楽曲と創造的にインタラクションできるようにしてくれる．制作ツールと異なり，このカテゴリでは基本的にシーク操作が許可されておらず，その代わり即興的な体験に重点が置かれている．例えば曲を聴きながら曲の終わりまで絵を描ける作品 (図 5 ③) や，タッチ操作で画面上に音符を置いていくとサビでその音符が踊りだす作品，キャラクタを操作して仮想空間内を移動させることができ，その軌跡に花を咲かせて絵や文字を構成できる作品 (図 A·3 ⑭) などがあった．

6.2.4 楽器アプリ (instrument)

このカテゴリは，ユーザが楽曲演奏に能動的に関与するための楽器や道具の役割を果たす．例えば，MediaPipe.js を用いたカメラ入力の画像処理によってユーザの手の動きを推定し，画面上の仮想的なペンライトに割り当てた作品があった (図 5 ④)．また，Web MIDI API を用いて接続された MIDI 機器の信号を読み取り，楽曲再生と同時に演奏ができる作品があった．ユーザのクリック操作のタイミングで既定の音を鳴らす作品もあった．

6.2.5 ゲーム (game)

このカテゴリは何らかのルールや得点制を採用しており，ユーザにゴールしたり高得点を取ったりしたいと思わせるような仕掛けが実装されている．典型的なゲームアプリでは，本フレームワークを活用することでユーザが課題曲を選べるようになっている．ゲームに分類された応募作品のすべてが，次々に出現する歌詞テキストとうまくインタラクションすることを目指す音楽リズムゲームであった．ある作品は，ユーザがライブステージ上のキャラクタを操作して，飛んでくる歌詞テキストにタイミングよく触れるゲームだった (図 5 ⑤)．また別の作品は，歌詞のフレーズごとに動的に生成された曲線上に文字が並んでおり，これらをタッチ操作でタイミングよくなぞっていくゲームだった (図 A·3 ⑯)．どの作品も，ユーザと歌詞の文字の間のユニークなインタラクションを追求していた．

6.2.6 拡張音楽動画 (augmented music video)

このカテゴリは対象楽曲の音楽動画を表示し，その再生と同期したインタラクティブな体験を提供する．例えば，特定の楽曲の音楽動画が画面右下に表示され，その情景を

参考にしたインタラクティブな視覚的要素が画面の他の領域に表示される作品があった (図 5 ⑥)．キャラクタが画面左から右に歩くシーンでは，ユーザがカーソル操作でキャラクタを左右に動かせるようになっていた．また，操り人形が操られて変形するシーンでは，ユーザがマウス操作で変形の仕方をコントロールできるようになっていた．他にも，音楽動画が額装されて画面中央に表示され，その周辺にステージのような情景が広がり，画面下部には客席が多数表示される作品もあった (図 A·3 ⑰)．楽曲再生と同期して歌詞などがステージに表示され，座席の観客もアニメーションするようになっていた．さながら映画館で作品鑑賞しているような演出であった．

6.2.7 インタラクティブリリックビデオ (interactive lyric video)

このカテゴリは，リリックビデオを動的に描画したうえで，ユーザがインタラクションできるようにしたものである．例えば，歌詞テキストが 3D の仮想空間内を右から左にスクロールする情景を描いた作品では，ユーザがテキストをマウスかタッチのドラッグ操作でつかんで左右に動かし，再生位置を変更できるようになっていた (図 5 ⑦)．別の作品では，発声された歌詞の文字が徐々に 3D 空間の中央に集まっていき，再生が進んでサビに到達すると文字が一気に散り散りになる演出がプログラムされており，ユーザはドラッグ操作で画面中央の文字の塊を自由な角度から見ることができるようになっていた．

6.2.8 ジェネラティブリリックビデオ (generative lyric video)

このカテゴリはリリックビデオをプログラムによって動的に描画するもので，単なる動画であればアスペクト比やフレームレートが固定であるのに対し，それらをユーザの閲覧環境に応じて柔軟に変更できるものが多い．シーク操作のような基本的な再生操作は可能だが，それ以外のインタラクションには対応していない．このカテゴリの典型的なリリックアプリはリリックビデオと似た体験をユーザにもたらすが，好みのスタイルにカスタマイズできる作品がある他，視覚表現として見たときに，歌詞のタイミングやその他の音楽的要素に合わせてアニメーションする同期的演出が多用される特徴がある．なお，リリックビデオでは見られない表現を追求した例外的な作品もあった．例えば，楽曲全体を特定の数のパートに分割し，イラストも同数のパネルに分割して，それぞれパズルのピースに見立てた作品があった (図 5 ⑧)．楽曲の再生が進むにつれ，ピースが画面上の適切な場所にはめこまれ，楽曲再生が完了するタイミングでイラストが完成する演出となっていた．また，歌詞テキストを同じ音素数の別の単語に置き換えて替え歌を生成し，画面上に表示する作品もあった．

6.2.9 プログラミング・コンテストの制約

プログラミング・コンテストの応募作品を分析し，フレームワークの評価を行うことに関しては，大きく分けて 2 点の制約がある．まず，コンテストの応募前につまづいた作品は含まれない．したがって，フレームワークが一定数のプログラマに利用されたことのエビデンスにはなるが，必ずしも初学者に使いやすいことを示すわけではない．ただし，後述するプログラミング・コンテスト後のアンケートの結果によれば，初学者が問題なく動作するリリックアプリを応募できていることが分かった．例えば，参加者には Web アプリケーションの開発経験がまったくない人々が含まれていた．

次に，コンテストの競争的なフォーマットは，参加者に対して，審査員映えするリリックアプリを開発するプレッシャーを与えてしまう．プレッシャーにより，(a) フレームワークの評価に影響が出る可能性と，(b) 応募作品の多様性が狭まる可能性がある．問題 (a) に関しては，フレームワークの開発者である我々が審査員を務めることで，フレームワークに関する技術的な問題が指摘しづらくなるリスクがあると考えており，以下のとおり対処した．(1) 審査員のうち半数をフレームワーク開発者以外とした．(2) 技術的な課題についてオープンに議論できるオンラインフォーラムを開設し，プログラミング・コンテストでは開発過程は評価対象にならないことを明示して，フレームワークに問題があった場合にためらわずに報告できるよう配慮した．(3) 応募作品のソースコードを読み，フレームワークの致命的な問題に対する対処がないことを確認した．

問題 (b) に関しては，近年の創造性支援研究の文脈 [8] で批判されてきた，統制が取れた環境で短期間ツールを利用してもらうような評価実験と比較すれば，むしろ条件は恵まれている．なぜなら，フレームワークが一般公開されており，多様な技術的背景のユーザに利用の機会が開かれてきたためである．また，一般公開から 2 年以上が経過し，長期間にわたる学習も可能な環境であった．さらに我々は，多様なリリックアプリが応募されるよう，プログラミング・コンテストの評価軸を工夫してきた．仮に評価軸がリリックアプリの美しさだけを重視するものであったなら，コンテストはデザイナやイラストレータの関与が求められるような性質のものになっていただろう．また，仮に評価軸がリリックアプリの実装の完成度だけを重視するものであったなら，コンテストは，競技プログラミングのように技術的課題を解くことが得意なプログラマが活躍するものになっていただろう．我々はコンテストがこのいずれにも偏ることがないよう 3 つの評価軸 (6.1.1 節) を定めた．このような多角的な評価軸を明示することで，参加者側に作品制作で重視するポイントを選択してもらえるようになり，結果として多様な応募作品が集まることを狙った．

6.3 フレームワークの有効性評価

本節では上記以外に追加で行った評価について報告する．

6.3.1 利用統計

本フレームワークの一般公開は 2020 年 9 月 18 日に始まった．リリックアプリ開発の利用統計を取るため，2021 年 9 月 22 日より，リリックアプリごとにアクセスキーを発行し，REST API サーバにアクセスする際にはアクセスキーをリクエストに含めるようにした．この記録によれば，2022 年 9 月 13 日の時点で 396 人のユーザが 448 個のリリックアプリを開発している．古いバージョンのリリックアプリが問題なく動作するよう互換性を保つため，アクセスキーがなくても REST API サーバへのアクセスを許すようにしており，実際にはこれより多くのアプリが開発されていることになる．

また，GitHub によればフレームワークのリポジトリは 78 個のスター (リポジトリへの興味と称賛を表す指標) を獲得しており，67 個以上のリポジトリでフレームワークが利用されている．さらに，2 年間のプログラミング・コンテスト参加者のうち 30 名が応募作品をオープンソース公開し，リリックアプリの開発コミュニティを盛り上げようとしてくれている．全体として，これらの客観的な統計値は，フレームワークが大きな問題なく利用されてきたことを示している．

6.3.2 ソースコード分析

我々は 52 個の応募作品のソースコードを読み，内容を分析した．16 個が我々の提供したサンプルコードの内容から部分的に派生したコードを含んでいたが，どの場合も差分があり，多くの場合で Web アプリケーションのビルド方法 (Parcel を使ってファイルをまとめ，一般公開用のコードを生成していた) が共通しているだけであった．他の，例えばインタラクションデザインや見た目に関しては，コンテスト参加者自身が工夫を凝らしていた．

フレームワークが提供している時刻駆動型 API は，さまざまなクリエイティブコーディングや GUI のライブラリと併用できていた．例えば p5.js，Three.js，Create.js，PixiJS，glslify，GSAP，PlayCanvas，そして Phaser などのクリエイティブコーディング用ライブラリが利用されていた．また，GUI ライブラリとして jQuery，React，Vue.js，そして Svelte などがリリックアプリをインタラクティブにする目的で活用されていた．Tweakpane や dat.gui などのユーザインタフェースのコンポーネントライブラリを活用して，ユーザがリリックアプリの振る舞いをカスタマイズできるようにしている例も見られた．さらに，多くの応募作品が静的画像，Web フォント，スタイルシートなどをリリックアプリに同梱しており，魅力的な見た目を実現していた．Lottie という，Adobe After Effects で制作したアニメーションを Web アプリケーションで使えるようにするツールを活用している例もあった．これらの多様な結果は，フレームワークが単に問題なく利用できていたというだけでなく，プログラミング用の道具として「低い敷居と高い天井 (low threshold and high ceiling)」 [43] の特長を兼ね備えていたこと，すなわち Web アプリケーション開発の初学者から熟練者まで幅広い技術的背景を持つプログラマに活用されていたことを示している．

さらにソースコードを読み込んだところ，多くの応募作品で独自の「シーンマネージャ」が実装されていることが分かった．典型的なケースでは，このマネージャはリリックアプリが楽曲を読み込んだ時点で初期化処理をし，楽曲再生中の映像演出を事前に準備する役割を果たしていた．その初期化内容としては，楽曲構造を分析してイントロ，繰り返し区間，アウトロなどに区間分割し，視覚的要素を表すオブジェクトをインスタンス化して再生時の演出に備えていた．そして楽曲の再生中は，マネージャは楽曲構造の遷移を監視し，視覚的要素の表示状態を管理していた．このリリックアプリの典型的なデザインパターンのために，フレームワーク側でできる拡張を 7.1.3 節で議論する．

6.3.3 質的ユーザ評価

我々はプログラミング・コンテスト参加者に自由記述でのアンケート回答を依頼し，各年に 21 件と 15 件の回答が寄せられ，コンテストに参加した動機やフレームワークに関するコメントなどの質的フィードバックを収集できた．例えば，プログラミング・コンテストが創造的な文化にプログラミングの力で貢献する貴重な機会となっていることを大きく評価する旨のコメントがあった．フレームワークに対しては肯定的な感想が大部分を占めたものの，致命的ではないが修正すべきバグに関する報告も寄せられた．

2 年目のアンケートでは参加者のプログラミング経験について詳細に尋ねた．回答によれば，まったくの未経験から 6 年の JavaScript 実務経験まで多様で，ソースコード分析 (6.3.2 節) で示されたフレームワークの「低い敷居と高い天井」を裏付ける内容だった．とくに，回答者の実に 3 分の 1 が JavaScript を用いたプログラミングの経験がほぼゼロであったことは驚きであり，それでもコンテストに参加した動機について 7.2 節で考察する．

このアンケートでは，参加者が時刻駆動型 API に満足しているか，そして歌詞などの音楽的要素に対するイベントリスナ (`player.addEventListener("phraseEnter", listener)` など) の必要性を感じているかについても質問した．回答は現状の API 設計に肯定的で，リリックアプリの状態管理用に 4.2.1 節のイベント駆動型 API は利用しているものの，我々の提案する時刻駆動型 API は合理的であり，リアルタイムなマルチメディア演出において有用であると考えていた．一人の回答者はイベントの種類が多いほどよいと回答していたが，他の回答者からはそういった要望はなかった．別の回答者は，複雑なシーン管理を実現したいが，そのためには現在の API 設計では不充分であるとコメントしていた．これはソースコード分析 (6.3.2 節) で述べたシーンマネージャの必要性を改めて示しているものである．

7. 議論

「リリックアプリ」という表現形式は，我々が提案したもので，現状研究対象としてはニッチと言えるかもしれない．しかし本研究は，3 つの幅広い観点で研究上の知見を提供するものである．

まず第一に，音楽情報処理の研究者，より広くは音楽業界の人々にとって，音楽コンテンツを取り巻くインタラクティブな体験の重要性は従前より指摘されており ([44] など)，リリックアプリはそうした体験を提供する有望な一手段となる．本フレームワークはリリックアプリの開発を支援する初めての取り組みであり，これは，リリックアプリをニッチから文化的メインストリームへ押し上げる重要な技術的マイルストーンである．

次に，本フレームワークで提案した時刻駆動型 API は，7.1 節で後述する通り，リリックアプリに限らず精密な時刻同期が必要なインタラクションデザイン全般で有用である．最後に，本フレームワークを利用した実証実験は，7.2 節で後述する通り，音楽とプログラミングという 2 つの創造的な活動の距離を近くする新技術によって，ミュージシャンとプログラマの新たな協業の機会が生まれたことを明らかにしたものである．

7.1 時刻駆動型 API に関する知見

6.3.3 節で紹介したとおり，時刻駆動型 API はユーザから肯定的に評価された．当該 API により，楽曲中の多様な音楽的要素を考慮に入れたインタラクションの実装が効果的に支援できた．本節では，さらにソースコードを分析することで得られた知見を報告する．すなわち，高度な演出をプログラミングするには再利用可能なコンポーネントが重要性であること (7.1.1 節)，そうしたデザインパターンを支援するためには，新たな API 設計が有用な可能性があること (7.1.2 節，7.1.3 節) である．

これらの知見は，精密な時刻同期が必要なインタラクションデザイン全般に応用できる．例えば，音と映像を同期させる演出の他，センサとアクチュエータを活用した実世界インタラクションの開発支援でも有用だと考えられる．

7.1.1 リリックアプリにおける再利用可能コンポーネント

プログラミング・コンテストの応募作品のほとんどが，クリエイティブコーディングや GUI のライブラリ (4.2.2 節) のためのレンダリング用関数を定義して画面演出をプログラミングしている．このようなジェネラティブなアルゴリズムは単一の巨大な関数内で多様な情報を組み合わせるようなコードを書くことでも実装できるが，こうした方法は単調な映像表現に繋がりやすく，我々はアンチパターンの一種だと考えている．

我々が分析したコードベースの中では，こうした関数を細分化して多くのサブルーチンに分け，サブルーチンを再利用可能なコンポーネントとして扱うデザインパターンに沿ったものが，コード全体のモジュール性を向上できており，複雑で魅力的な画面演出に成功している傾向にあった．

7.1.2 時区間駆動型 (time-range-driven) API

リリックアプリ開発においては，前述のような再利用可能コンポーネントのライフサイクルが，音楽的要素の可聴時区間 (ビートから次のビートまでの区間やフレーズの発声区間など) に紐づいていることは注目に値する．応募作品のソースコードの中で我々は，現在の再生位置がそうした可聴時区間の中に入ったか，あるいは外に出たことを検出する機能がたびたび実装されていることを発見した．例えば，ある作品では，画面上のパーティクルの出現を管理するパーティクルマネージャが，前回の `player.findPhrase` 関数の呼び出し結果を保存しており，前回と今回の結果を比較することで，パーティクルを表すオブジェクトを初期化するか破棄するか決定していた．

こうしたデザインパターンは，フレームワークに新たな API を実装することで支援できる．例えば，時区間を引数に与える次のような API が考えられる:

```
player.findPhraseChange(previousPosition,
    currentPosition);
```

返り値は，その時区間で発声を終えたフレーズの配列，発声を始めたフレーズの配列，そして発声中のフレーズとなる (詳しくは付録 A.1を参照のこと).

時刻駆動型 API を拡張した「時区間駆動型 (time-range-driven) API」によって，リリックアプリ開発において以下のような利点が得られる．まず，プログラマが自身のコードで能動的に情報を取りに行く API の特長が保たれるため，さまざまなクリエイティブコーディングや GUI のライブラリと併用できる．次に，時区間を引数で与える API 設計により，プログラマが任意の時区間を自ら定義して音楽的要素を探すことができる．これはアニメーション遷移をプログラミングするうえで有用である．例えば，サビ区間に突入する 5 秒前に遷移を始めたければ，以下のように時区間駆動型 API を呼び出せばよい:

```
player.findChorusChange(previousPosition,
    currentPosition + 5000);
```

7.1.3 ユーザ定義時区間

応募作品の多くが，楽曲全体を複数の区間に区切って，それぞれ固有の歌詞駆動演出を表示するように実装されていた．とくに，6.3.2 節でも紹介したとおり，区間ごとの演出の遷移を管理する「シーンマネージャ」を実装して，楽曲再生の進行と演出とのマクロな対応付けを行うという実装パターンが典型的なようであった．こうした時区間は，演出を組み立てるプログラマが独自に考えつく創造的な努力の成果であり，フレームワーク側であらかじめ定義して

おくことはできない．そこで，プログラマが自身で時区間を定義したうえで再生位置と比較する API を提供できれば，こうしたシーンマネージャの実装を支援できる．例えば次のような API が考えられる:

```
const timeRange = player.createTimeRange(
    startTime, endTime);
player.findTimeRangeChange(timeRange,
    previousPosition, currentPosition);
```

7.2 静的メディアへのインタラクティビティ付与

リリックアプリ開発は，コードエディタ上でプログラムをまっさらな状態から書く体験とは少し違っている．なぜなら，既存の楽曲に対して新しくインタラクションする機能を追加するプロセスだからである．本節では，こうした，既存のコンテンツを拡張するプログラミング体験が持つ利点と，今後の研究に向けた興味深いトピックを議論する．

7.2.1 創造性のキックスタート

我々は，プログラミング・コンテストを通して，本フレームワークが典型的なクリエイティブコーディングのコミュニティよりも広範なコミュニティに訴求できており，参加者が開発成果を公開することを促進できていると感じた．ある参加者は，通常であればこうしたコンテストには参加しようと考えないし，参加できなかっただろうとコメントしていた．コンテストで設定されていた課題曲への愛が，こうした参加者にリリックアプリ開発とコンテスト参加を促していた．クリエイティブコーディングは近年コンピュータ科学教育や芸術的意義の観点で注目を集めているが，すべてのプログラマが，自身が (クリエイティブコーディングを楽しめるほどに) 充分「創造的 (クリエイティブ)」だという自信を持っているわけではない．

リリックアプリには，そうしたプログラマが「創造的」な部分で既存の楽曲の力を借りて，結果として自身の創造性を発揮するキックスターターにできるポテンシャルがある．我々は，プログラミングという活動それ自体が充分「創造的」であると考えており，より多くの人々が創造性を発揮するきっかけにリリックアプリがなってきたことを嬉しく思っている．こうしたリリックアプリ開発は，コンピュータ科学教育のカリキュラムでも役立つはずである．さらに，音楽以外の静的メディアコンテンツに対してもインタラクティビティを付与するようなクリエイティブコーディングへの発展も，今後の研究として興味深い．

7.2.2 コミュニケーションとしてのプログラミング

リリックアプリの開発過程で，プログラマは何度も繰り返し楽曲を聴き，歌詞を読むことになる．こうした体験を通して，プログラマは魅力的な歌詞駆動型の演出を思いつくのかもしれない．リリックビデオがミュージシャンとモーショングラフィックデザイナの協力関係を可能にしたことと同様に，リリックアプリはミュージシャンがプログラマと協力する新たな機会を生み出している．「プログラミング環境」はコンパイラやエディタ，デバッガなど，単なるソフトウェアの集合として見なされがちだが，実際のプログラミングはもっとソーシャルな活動であり，プログラミング環境は人々とソフトウェアが織りなす共創的な環境なのである [45]．実際，拡張音楽動画のリリックアプリ (図 5 ⑥) を開発した参加者は，元の楽曲と音楽動画の作者に問い合わせを行い，派生作品の共創的な開発を楽しんでいた．

プログラミング・コンテストの受賞者を発表する「マジカルミライ」の企画展ステージにおいて，コンテストを主催するクリプトン・フューチャー・メディア株式会社の代表取締役であり，音楽業界に詳しい伊藤 博之氏は，音楽と技術の密接な関係に以下のように言及した．『ミュージシャンが聴衆に音楽を届ける方法は常に科学的発明の影響を受けてきた．ピアノを製造するための木工技術から，シンセサイザーや歌声合成ソフトウェアの技術まで，枚挙にいとまがない．これらの技術はミュージシャンの新たな創作の機会となってきたし，音楽コンテンツは常に技術の変化に適応してきた．さらに，近年では，ミュージシャンが歌声合成ソフトウェアで楽曲を作り，イラスト，歌唱，さまざまな動画をみんなが作って共有する創作文化がニコニコ動画や YouTube などの動画配信サービスで培われ，メインストリームの文化になってきた．リリックアプリは，プログラマが創作文化に貢献できる機会を増やす新しいプラットフォームであり，今後の展開に期待している．』（抜粋の上、要約して掲載）

7.2.3 創造的文化に関する今後の研究

創造性支援の研究は長らく，技術的新規性に偏重したものが多く，評価実験もラボスタディなど統制的な環境が主流で，「在野 (in the wild)」での研究が不足していると批判されてきた [8]．多様なメディアを横断してコンテンツがユーザの手元に届くメディアコンバージェンス [3] の時代には，コンテンツの価値はそれ自体で閉じない．派生作品や，聴衆の SNS での関与なども含めて総合的に考える必要がある．コンテンツを取り巻く生態系はますます広がりを見せており，その実態を踏まえた研究のためにも，在野での研究の重要性は増している．

本フレームワークは一般公開されており，まさに在野での取り組みと見なすことができるが，本稿で報告したプログラミング・コンテストでもまだ制約はあった (6.2.9 節)．例えば，サーバサイドのプログラムを含められなかったり，ソーシャルな機能の実装や実験的な API の採用が見送られがちだったりした．我々は，本フレームワークの研究開発を続け，在野での活用を支援し続けることで，新たなリリックアプリのカテゴリが切り拓かれ，創造的文化に関する構成論的研究をさらに進められると信じている．

8. おわりに

本稿では，リリックアプリという新たな表現形式を提案し，その特徴を明らかにして，プログラマが開発するうえでの技術的課題を整理した．そして，そうした課題を解決するフレームワークを開発し，一般公開して，プログラミング・コンテストでの実証実験を行った．これにより，リリックアプリのカテゴリ 8 種類や，今後の研究のための知見を得ることができた．

謝辞 プログラミング・コンテストはクリプトン・フューチャー・メディア株式会社が運営した．こうした取り組みは，新しい技術が人々の創造性に貢献するという信念を共有した信頼・協力関係のもとで可能となったものである．また，継続的なコンテスト開催は，情熱的な参加者による応募がなければ実現しなかった．参加者の尽力に感謝するとともに，リリックアプリの可能性を切り拓く学術的にも大きな意義を持つ取り組みの中で応募作品を紹介できたことを嬉しく思う．

本フレームワークの Web ベースのワークフローに貢献した産総研の関係者各位にも感謝したい: 中野 倫靖 (歌詞同期エンジン)，渡邉 研斗 (形態素解析 API)，川崎 裕太・井上 隆広 (Songle および Songle API)．本研究の一部は以下の支援を受けた: JST CREST (JPMJCR20D4)，JST ACCEL (JPMJAC1602)，JST ACT-X (JPMJAX22A3)．

参考文献

[1] Stratton, V. N. and Zalanowski, A. H.: Affective Impact of Music Vs. Lyrics, *Empirical Studies of the Arts*, Vol. 12, No. 2, pp. 173–184, DOI: 10.2190/35T0-U4DT-N09Q-LQHW (1994).

[2] Kato, J., Nakano, T. and Goto, M.: TextAlive: Integrated Design Environment for Kinetic Typography, *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pp. 3403–3412, DOI: 10.1145/2702123.2702140 (2015).

[3] Jenkins, H.: *Convergence Culture: Where Old and New Media Collide*, New York University Press (2006).

[4] Jenkins, H.: *Confronting the challenges of participatory culture: Media education for the 21st century*, The MIT Press (2009).

[5] Levtov, Y.: Algorithmic Music for Mass Consumption and Universal Production, *The Oxford Handbook of Algorithmic Music* (McLean, A. and Dean, R. T., eds.), Oxford University Press, chapter 34, pp. 627–644, DOI: 10.1093/oxfordhb/9780190226992.013.15 (2018).

[6] Kato, J. and Goto, M.: Lyric App Framework: A Web-Based Framework for Developing Interactive Lyric-Driven Musical Applications, *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, pp. 124:1–124:18, DOI: 10.1145/3544548.3580931 (2023).

[7] 加藤 淳，中野倫靖，後藤真孝：TextAlive: インタラクティブでプログラマブルな Kinetic Typography 制作環境，第 22 回インタラクティブシステムとソフトウェアに関するワークショップ，WISS '14, pp. 39–44 (2014).

[8] Frich, J., Mose Biskjaer, M. and Dalsgaard, P.: Twenty Years of Creativity Research in Human-Computer Interaction: Current State and Future Directions, *Proceedings of the 2018 Designing Interactive Systems Conference*, DIS '18, pp. 1235–1257, DOI: 10.1145/3196709.3196732 (2018).

[9] TuneWiki, Inc.: Lyric Legend, `https://web.archive.org/web/20110208002254/http://www.lyriclegend.com/` (2011). 2022 年 9 月 13 日にアクセス (2011 年 2 月 8 日のアーカイブ).

[10] 株式会社セガゲームス：Miku Flick-ミクフリック-｜初音ミクが、ついに iPhone に降臨！，`https://miku.sega.jp/flick` (2012). 2023 年 7 月 30 日にアクセス.

[11] LyricsTraining.com: Learn Languages with Music Videos, Lyrics and Karaoke!, `https://lyricstraining.com/` (2011). 2022 年 9 月 13 日にアクセス.

[12] Goto, M., Ogata, J. and Eto, K.: PodCastle: a web 2.0 approach to speech recognition research, *Proceedings of the 8th Annual Conference of the International Speech Communication Association*, INTERSPEECH '07, pp. 2397–2400 (2007).

[13] Pavel, A., Reed, C., Hartmann, B. and Agrawala, M.: Video Digests: A Browsable, Skimmable Format for Informational Lecture Videos, *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pp. 573–582, DOI: 10.1145/2642918.2647400 (2014).

[14] Fujihara, H., Goto, M., Ogata, J. and Okuno, H. G.: LyricSynchronizer: Automatic Synchronization System Between Musical Audio Signals and Lyrics, *IEEE Journal of Selected Topics in Signal Processing*, Vol. 5, No. 6, pp. 1252–1261, DOI: 10.1109/jstsp.2011.2159577 (2011).

[15] 株式会社シンクパワー：歌詞データ関連事業「01. 同期歌詞（プチリリ機能）」，`https://syncpower.jp` (2023). 2023 年 7 月 30 日にアクセス.

[16] Musixmatch s.p.a: Musixmatch Developer API, `https://developer.musixmatch.com/` (2022). 2022 年 9 月 13 日にアクセス.

[17] LyricFind Inc.: Products — Lyric Display — LyricFind，`https://www.lyricfind.com/products/lyric-display` (2023). 2023 年 7 月 30 日にアクセス.

[18] Shamma, D. A., Pardo, B. and Hammond, K. J.: MusicStory: A Personalized Music Video Creator, *Proceedings of the 13th Annual ACM International Conference on Multimedia*, MM '05, pp. 563–566, DOI: 10.1145/1101149.1101278 (2005).

[19] Xia, H., Jacobs, J. and Agrawala, M.: Crosscast: Adding Visuals to Audio Travel Podcasts, *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, pp. 735–746, DOI: 10.1145/3379337.3415882 (2020).

[20] Lin, J.-C., Wei, W.-L., Lin, Y.-Y., Liu, T.-L. and Liao, H.-Y. M.: Learning From Music to Visual Storytelling of Shots: A Deep Interactive Learning Mechanism, *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, pp. 102–110, DOI: 10.1145/3394171.3413985 (2020).

[21] Pearson, M.: *Generative Art: A Practical Guide Using Processing*, Manning Publications Co. (2011).

[22] McLean, A. and Dean, R. T.: *The Oxford Handbook of Algorithmic Music*, Oxford University Press (2018).

[23] Reas, C. and Fry, B.: *Processing: A Programming Handbook for Visual Designers and Artists, Second Edition*, The MIT Press (2014).

[24] Lieberman, Z., Watson, T. and Castro, A.: openFrameworks, https://openframeworks.cc/ (2022). 2022 年 9 月 13 日にアクセス.
[25] Bostock, M., Ogievetsky, V. and Heer, J.: D3: Data-Driven Documents, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 17, No. 12, pp. 2301––2309, DOI: 10.1109/TVCG.2011.185 (2011).
[26] Conlen, M. and Heer, J.: Idyll: A Markup Language for Authoring and Publishing Interactive Articles on the Web, *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, pp. 977–989, DOI: 10.1145/3242587.3242600 (2018).
[27] Klokmose, C. N., Eagan, J. R., Baader, S., Mackay, W. and Beaudouin-Lafon, M.: Webstrates: Shareable Dynamic Media, *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pp. 280–290, DOI: 10.1145/2807442.2807446 (2015).
[28] Klokmose, C. N., Remy, C., Kristensen, J. B., Bagge, R., Beaudouin-Lafon, M. and Mackay, W.: Videostrates: Collaborative, Distributed and Programmable Video Manipulation, *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19, pp. 233–247, DOI: 10.1145/3332165.3347912 (2019).
[29] Rädle, R., Nouwens, M., Antonsen, K., Eagan, J. R. and Klokmose, C. N.: Codestrates: Literate Computing with Webstrates, *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pp. 715–725, DOI: 10.1145/3126594.3126642 (2017).
[30] björk: björk: full biophilia app suite, https://www.youtube.com/watch?v=dikvJM__zA4 (2011). 2022 年 9 月 13 日にアクセス.
[31] Opal Limited: Brian Eno: Reflection, https://apps.apple.com/us/app/brian-eno-reflection/id1180524479 (2016). 2022 年 9 月 13 日にアクセス.
[32] Reality Jockey Ltd.: RjDj, https://web.archive.org/web/20120730133514/http://rjdj.me/ (2012). 2022 年 9 月 13 日にアクセス (2012 年 7 月 30 日のアーカイブ).
[33] Paterson, J. and Toulson, R.: Interactive digital music: enhancing listener engagement with commercial music, *Innovation in Music II* (Paterson, J., toulson, R., Hodgson, J. and Hepworth-Sawyer, R., eds.), KES Transactions on Innovation in Music, Vol. 2, Future Technology Press, pp. 193–209 (2015).
[34] Kato, J., Ogata, M., Inoue, T. and Goto, M.: Songle Sync: A Large-Scale Web-Based Platform for Controlling Various Devices in Synchronization with Music, *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, pp. 1697–1705, DOI: 10.1145/3240508.3240619 (2018).
[35] Goto, M., Yoshii, K. and Nakano, T.: Songle Widget: Making Animation and Physical Devices Synchronized with Music Videos on the Web, *2015 IEEE International Symposium on Multimedia (ISM)*, pp. 85–88, DOI: 10.1109/ISM.2015.64 (2015).
[36] Goto, M., Yoshii, K., Fujihara, H., Mauch, M. and Nakano, T.: Songle: A Web Service for Active Music Listening Improved by User Contributions, *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR '11, pp. 311–316 (2011).
[37] 後藤真孝，吉井和佳，藤原弘将，Mauch, M.，中野倫靖：Songle: 音楽音響信号理解技術とユーザによる誤り訂正に基づく能動的音楽鑑賞サービス，情処学論， Vol. 54, No. 4, pp. 1363–1372 (2013).
[38] 工藤　拓：MeCab: Yet Another Part-of-Speech and Morphological Analyzer, https://taku910.github.io/mecab/ (2013). 2022 年 11 月 11 日にアクセス.
[39] Loper, E. and Bird, S.: NLTK: The Natural Language Toolkit, *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pp. 63–70, DOI: 10.3115/1118108.1118117 (2002).
[40] クリプトン・フューチャー・メディア株式会社：初音ミク「マジカルミライ」ポータルサイト，https://magicalmirai.com/ (2023). 2023 年 7 月 30 日にアクセス.
[41] クリプトン・フューチャー・メディア株式会社：初音ミク「マジカルミライ 2020」プログラミング・コンテスト，https://magicalmirai.com/2020/procon (2020). 2023 年 7 月 30 日にアクセス.
[42] クリプトン・フューチャー・メディア株式会社：初音ミク「マジカルミライ 2021」プログラミング・コンテスト，https://magicalmirai.com/2021/procon (2021). 2023 年 7 月 30 日にアクセス.
[43] Myers, B., Hudson, S. E. and Pausch, R.: Past, Present, and Future of User Interface Software Tools, *ACM Trans. Comput.-Hum. Interact.*, Vol. 7, No. 1, p. 3–28, DOI: 10.1145/344949.344959 (2000).
[44] Goto, M. and Dannenberg, R. B.: Music Interfaces Based on Automatic Music Signal Analysis: New Ways to Create and Listen to Music, *IEEE Signal Processing Magazine*, Vol. 36, No. 1, pp. 74–81, DOI: 10.1109/MSP.2018.2874360 (2019).
[45] Kato, J. and Shimakage, K.: Rethinking Programming "Environment": Technical and Social Environment Design toward Convivial Computing, *Conference Companion of the 4th International Conference on Art, Science, and Engineering of Programming*, Programming '20, pp. 149–157, DOI: 10.1145/3397537.3397544 (2020).

付　　録

A.1 イベント・時刻・時区間駆動型 API

4.2.2 節と 7.1.2 節への補足として，精密な時刻同期が必要なインタラクションデザインを支援する API の 3 類型について，図 A·1とともに説明する．A.1.1 節で紹介するイベント駆動型は既存手法，A.1.2 節の時刻駆動型は本研究で提案した設計，A.1.3 節の時区間駆動型は実証実験を踏まえて新たに考案した設計である．

A.1.1 イベント駆動型 (event-driven) API

イベント駆動型 (event-driven) API は，何かが起きたときに「イベント」を発行し，リスナに通知する．この「push」型の API 設計は，アニメーションが楽曲再生中の特定のタイミングで開始するシンプルなユースケースでは便利である．例えば，ビートのイベントに対応するリスナを実装すれば，ビートごとに波紋のアニメーションを再生できる．

しかし，4.2.2 節で議論したとおり，これは未来に向けた準備のロジックの設計や，複数の音楽的要素を考慮に

入れた演出のプログラミングがしづらく，時間精度にも課題がある．さらに，音楽的要素のなかには，再生時刻に紐づいたデータであるにもかかわらず，特定のタイミングで発行されるイベントとして表現しづらいものがある．こうした不適合によって，プログラマがリスナを大量に書かざるを得なくなるなどのデメリットがある．例えば，表 1に示した音楽的要素の中には，歌詞の単語の発声区間やコード進行のように特定の時区間を表すものがあり，これらは `onWordEnter` と `onWordLeave`，`onChordEnter` と `onChordLeave` のように 2 種類のイベントに分割して表現する必要がある．さらに，時系列で値が変化する声量などの音楽的要素に関しては，`onVocalAmplitudeUpdate` のようなイベントを頻繁に (例えば 10 ミリ秒ごとなどに) 発行しなければ，なめらかな演出をプログラミングできない．

A.1.2 時刻駆動型 (time-driven) API

4.2.2 節において, 我々は「時刻駆動型 (time-driven) API」を提案した．これは，特定の時刻情報を引数に与えて API を呼び出せば，そのタイミングに関する時系列データが返される「pull」型の設計である．フレームワーク側で，結果を高速に返せるよう工夫する (例えば，我々の現在の実装では，情報を時刻でソートされたバイナリツリーとして保持しておく) 必要はあるが，プログラマが好きなときに情報を問い合わせられるメリットは大きい．

イベント駆動型 API ではデータが特定の時刻と紐づいた個別のイベントとして表現されるのに対し，時刻駆動型 API では特定の時区間として表現される．これはデータの自然な表現になっており，A.1.1 節で述べたようなデメリットが生じない．例えば，歌詞の単語は発声が始まる時刻から終わる時刻までの時区間として表現される．ビートはその開始時刻から次のビートの開始時刻直前までの時区間となる．その他，連続的に値が変化する声量などの音楽的要素は，楽曲の再生開始から終了まで続く時区間となる．

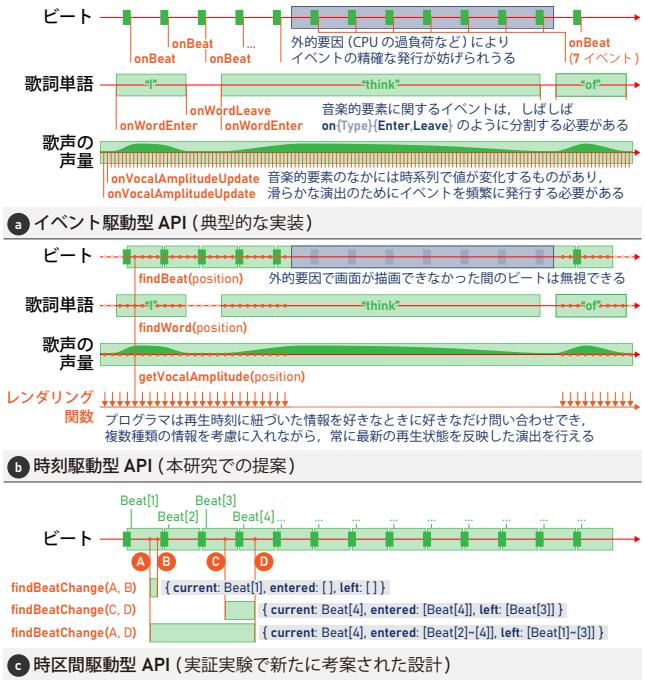


図 A·1: 精密な時刻同期が必要なインタラクションデザインを支援する API の 3 類型.

A.1.3 時区間駆動型 (time-range-driven) API

7.1.2 節で，我々はユーザフィードバックとソースコード分析の結果を受けて新たに「時区間駆動型 (time-range-driven) API」を提案した．時刻駆動型 API が指定された再生時刻と音楽的要素の時区間との重複を探索し，見つかった最初の音楽的要素を返すのに対して，時区間駆動型 API は指定された時区間との重複があるすべての音楽的要素を探索して返す．典型的なユースケースでは，プログラマは画面を描画するレンダリング関数の中で現在の再生時刻を保存し，次に関数が呼ばれた際にそれまでに起きたことを知るために時区間駆動型 API を呼ぶことができる．

時区間駆動型 API の返り値は `current`，`entered`，そして `left` のタプルである．プログラマはこの結果から，`entered` に含まれる音楽的要素に対応する視覚的要素を初期化し，`current` に対応する音楽的要素の状態情報を用いて視覚的要素の見た目を更新し，最後に `left` に対応する視覚的要素を破棄する．なお，`current` は `null` または単一の音楽的要素であり，`entered` および `left` は可変長配列となる．例えば `(0, duration)` を引数として渡すと，それぞれに `null`，全要素，全要素が格納された結果が返ってくる．

A.2 サンプルコードの詳細

6.1.2 節への補足として，本フレームワークで開発したサンプルコード全 11 件の内容を以下にまとめる．図 4の 3 件に加え，図 A·2に他の 3 件のスクリーンショットも示す．

なお，すべてのサンプルコードが GitHub Pages 上にデプロイされ，一般公開されており，プログラマはその Web サイトにアクセスするだけでリリックアプリの振る舞いをすぐ確認できるようになっている．一部のサンプル (インタラクティブ歌詞カードなど) は内容が Web ベースの統合開発環境である CodePen 上にコピーされており，プログラマは Web 上でソースコードを編集し，TextAlive App API の動作を確認できる．

A) **インタラクティブ歌詞カード**．ビルドツール不使用，DOM 操作と CSS アニメーションにより視覚効果を実装．6.1.2 節 Ⓐで紹介したもの．図 4 Ⓐ, GitHub:TextAliveJp/textalive-app-lyric-sheet.

B) **歌詞タイル**．Parcel でビルド，HTML5 Canvas API で画面描画．6.1.2 節 Ⓑで紹介したもの．図 4 Ⓑ, GitHub:TextAliveJp/textalive-app-lyric-tiles.

C) **ダンスするリリックアプリ**．Parcel でビルド，Three.js で画面描画．6.1.2 節 Ⓒで紹介したもの．図 4 Ⓒ，GitHub:TextAliveJp/textalive-app-dance.

D) **歌詞文字キューブ**．Parcel でビルド，Three.js で画面描画．シンプルなワイヤフレームで描画された立方体が画面中央に表れて回転する．各面は発声中の歌詞テキスト一文字をアニメーションさせながら表示している．フレームワークと 3D CG 用クリエイティブコーディングライブラリの組み合わせでもっともシンプルな例として実装された．図 A·2 Ⓓ，GitHub:TextAliveJp/textalive-app-char-cube.

E) **歌詞モザイク**．Parcel でビルド，HTML5 Canvas API で画面描画．発声中の歌詞テキスト一文字がモザイク状のパターンで表示される．内部処理としては，まず，歌詞中のすべての文字が既定サイズのオフスクリーンバッファにレンダリングされ，塗りつぶされたピクセルの数の順に並び替えられる．次に，ブラウザの画面が適切なサイズのタイルを敷き詰めたグリッドに分割される．楽曲再生中は，発声中の歌詞テキスト一文字を画面全体に最大化表示する代わりに，グリッドの各マスの濃さに近い文字がランダムに選択されマス内に表示される．図 A·2 Ⓔ，GitHub:TextAliveJp/textalive-app-mosaic.

F) **p5.js サンプル**．Parcel でビルド，p5.js で画面描画．画面中央に p5.js によるキャンバスが表示され，歌詞テキストのシンプルなアニメーションが表示される．著名なクリエイティブコーディングライブラリである p5.js のシンプルな活用事例として実装された．図 A·2 Ⓕ，GitHub:TextAliveJp/textalive-app-p5js.

G) **基本サンプル**．Parcel でビルド，DOM 操作で視覚効果を実装．画面中央に発声中の歌詞単語が表示される．基本的な再生コントロールの他，歌詞の最初の一文字の発声が始まる箇所の頭出しができるボタンや，サビの頭出しができるボタンなどのユーザインタフェースが表示される．初学者向けのサンプルとして実装された．GitHub:TextAliveJp/textalive-app-basic.

H) **Lottie サンプル**．Parcel でビルド，Lottie でアニメーションを表示．基本サンプルをベースに，Lottie ファイルを読み込むことで歌詞テキストの背景に魅力的なアニメーションが表示される．Lottie は Adobe After Effects で制作可能なアニメーションのフォーマットであり，Web プログラマとモーショングラフィックデザイナの協業でしばしば使われる．リリックアプリ開発においても従来のモーショングラフィック用制作ツールを活用可能であることを示すサンプルとして実装された．GitHub:TextAliveJp/textalive-app-lottie.

I) **パラメタ調整サンプル**．Parcel でビルド，React で DOM 操作することで視覚効果を実装．基本サンプルとほぼ同じ見た目だが，Lyric App Customizer (4.3.2 節) で配色などを調整可能．GitHub:TextAliveJp/textalive-app-params.

J) **フレーズとビートの同期サンプル**．Parcel でビルド，DOM 操作で視覚効果を実装．基本サンプルをベースに，発声中のフレーズを表示するように改変し，ビートと同期した視覚効果が表示される．GitHub:TextAliveJp/textalive-app-phrase.

K) **フレーズとビートの同期サンプル (script タグ使用)**．ビルドツール不使用，DOM 操作で視覚効果を実装．ひとつ前のサンプルコードと同じ機能と見た目だが，ビルドツールを使わず，HTML の script タグで TextAlive App API を読み込んでいる．ビルドツール不要で，素の HTML/JavaScript/CSS ファイルをテキストエディタなどで編集するだけでリリックアプリが開発できることを示すために実装された．GitHub:TextAliveJp/textalive-app-script-tag.

A.3 プログラミング・コンテストの応募作品

6.2 節への補足として，プログラミング・コンテストの応募作品の多様性を示すため，図 5の 8 作品に加え，さらに 12 作品のスクリーンショットを図 A·3に示した．

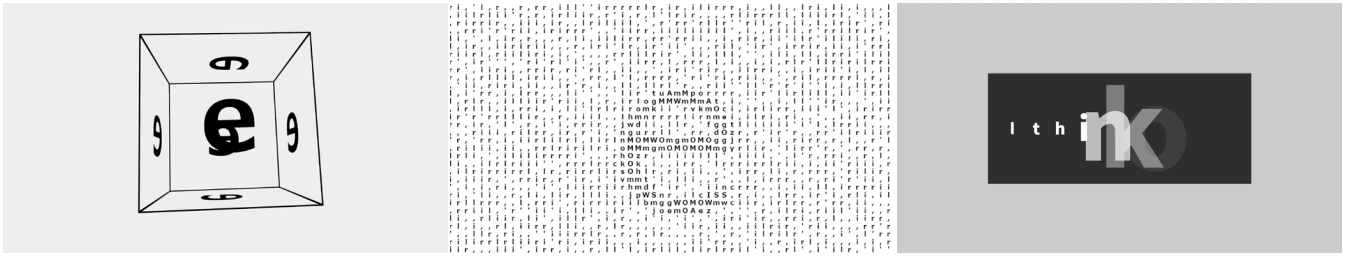
なお，これらの 20 作品は，以下の URL から，デモを閲覧したり実際に試したりできる．

A.3.1 図 5

1) https://youtu.be/NqszHM1g9xE
2) https://youtu.be/z_M1uBCV0tc
3) https://youtu.be/XzYZr_urn3I
4) https://youtu.be/hvmDxCeyWU8?t=3911
5) https://youtu.be/r26c6Og7r9k
6) https://youtu.be/5BlFr6Ma70w
7) https://youtu.be/_yAlLiIGByI
8) https://youtu.be/bvRqBNwZBEM

A.3.2 図 A·3

9) https://youtu.be/sYyGA_4YbwM
10) https://youtu.be/vnginTMqg0Y
11) https://youtu.be/LiHmw7m5bCs
12) https://youtu.be/2dopnxQrWGc
13) https://youtu.be/KQc3FCelKNo
14) https://youtu.be/hvmDxCeyWU8?t=3859
15) https://youtu.be/10qkhB_NYEw
16) https://youtu.be/VaCmJWSiNBg
17) https://youtu.be/-t9AVVgZo5k?t=3061
18) https://youtu.be/W-Sb5FbnvZI
19) https://youtu.be/cs9qxULFARg
20) https://youtu.be/hvmDxCeyWU8?t=3802



D 歌詞文字キューブ　E 歌詞モザイク　F p5.js サンプル

図 A·2: オープンソース公開しているサンプルコード 11 件中 3 件のスクリーンショット.

9 拡張現実アプリ　10 拡張現実アプリ　11 制作ツール
12 制作ツール　13 創造的即興アプリ　14 創造的即興アプリ
15 ゲーム　16 ゲーム　17 拡張音楽動画
18 インタラクティブリリックビデオ　19 インタラクティブリリックビデオ　20 ジェネラティブリリックビデオ

図 A·3: 多様な視覚的スタイルとインタラクションデザインを実装したプログラミング・コンテスト応募作品 12 件のスクリーンショット. (スクリーンショット提供: クリプトン・フューチャー・メディア株式会社)