情報の可視化とアクセス容易化のための C++ フレームワークの新たな設計方法に関する研究

Research on a new design method of C++ framework for information visualization and ease of access

2022年3月

鈴木　遼
Ryo SUZUKI

情報の可視化とアクセス容易化のための C++ フレームワークの新たな設計方法に関する研究

Research on a new design method of C++ framework for information visualization and ease of access

2022年3月

早稲田大学大学院　基幹理工学研究科
表現工学専攻　デジタルメディア表現研究

鈴木　遼
Ryo SUZUKI

目次

1 序論

1.1 本研究の意義

誰にでも開かれた高度なデジタル社会の到来を加速するには，一人でも多くの人々がそのような仕組みの構築に携わり，それらの作業が効率的に実行される必要がある．計算機の活用においては，とくに視覚情報の提示と，人と計算機のインタラクションが大きな役割を果たしているが，それらの設計に使われるプログラミングは，表現技術の高度化・複雑化といった要因もあり依然として難しいものである．このような分野におけるプログラミングの難易度を下げるための取り組みは長年にわたり世界中で行われており，例えば OpenGL のようなグラフィックスライブラリ，Processing のような新たなプログラミング言語，Unity のようなゲームエンジンが代表例である．新しい技術や設計の工夫によるプログラミングのしやすさの改善は，数万〜数百万人単位のソフトウェア開発者の生産性を底上げし，プログラミングの裾野の拡大に貢献してきた．

本論文の筆者は，利用者の多いプログラミング言語 C++ に着目し，C++ が持つ豊富なライブラリやコードベースなどの蓄積を生かしつつ，プログラミングに要求されるスキルを下げる工夫を行うことによって，C++ を用いた高度な情報処理へのアプローチを容易にするための新たな手法を考案し，それを実現するために「Siv3D」と呼ぶプログラミングフレームワークの提案を行った．本論文では，Siv3D の設計や運用に関する一連の取り組みをまとめ，重要な要素や技術的ポイントについて説明する．

1.2 本研究の課題と目的

プログラミング言語 C++ は，ゼロコスト抽象化と，決定論的なリソース寿命管理という特長を有し，実行時の速度を重視するソフトウェアや，大規模で複雑な並行処理を行うソフトウェアの開発に広く用いられる．一方で，小さなアプリケーションを開発するだけでも専門的な知識が要求される学習難度と，標準ライブラリのカバーする領域の小ささから，情報の可視化やユーザとのインタラクションを扱うアプリケーションや，プロトタイピングといった小規模かつ高レイヤの開発において用いられる機会は少なく，そのような用途では他のプログラミング言語が採用されることが一般的である．

こうした状況は，C++ がこれまで蓄積してきた様々な分野にわたるライブラリ資産の活用機会を狭め，400 万人以上 [82] とされる C++ 利用者が生み出せるソフトウェアプロダクトの可能性を制限する，もったいない状態を引き起こしている．そこで筆者は，C++ が抱えるこのような弱点を克服するために，2011 年以降急速にアップデートされている C++ の言語仕様を活用する API（Application Programming Interface）設計と，機能設計の工夫を盛り込んだ C++ フレームワーク Siv3D を開発し，9 年間にわたって普及と更新に取り組んできた．

本論文では，Siv3D を題材に，フレームワークを構築するアイデア，コミュニティ，プロセスを文書化し，重要な要素や技術的ポイントについて説明する．主に議論される Siv3D の方法論は次の通りである．

- C++ API 設計におけるコードの表現力の拡張
  - より高度な抽象化と，冗長・曖昧さの排除
  - コンパイル時におけるエラー検出の強化
- プログラミングに直接関連しない要素
  - フレームワークへのアセット同梱による利便性向上

– 開発参加を促進するユーザコミュニティ育成

現在の Siv3D の機能性や先進性，コミュニティは，過去 9 年間の試行錯誤によって実現したものであり，誰もが容易に達成できる仕事ではない．Siv3D の開発経験から得られる具体的かつ再利用可能な知見を，世界中のソフトウェア開発者と共有することで，将来のフレームワークの品質向上と，アプリケーション開発の発展に寄与できると考える．

1.3 本研究の成果

本研究の特筆すべき成果は次の 4 つである．

研究開発を支援するツールセットを提供した

HCI (Human-Computer Interaction) 研究やデジタルメディア表現研究のためのソフトウェア開発に活用できる C++ プログラミングフレームワーク Siv3D を開発し，オープンソースライセンスで公開した．これまでに，実験や可視化用のソフトウェア開発に Siv3D を活用した学術論文が少なくとも 12 報発表されている．また，Siv3D により提供される機能を組み合わせることで，新しいメディア表現の開発が容易に達成されることを，実際に稼働したアプリケーションの事例を通して示した．

情報可視化やインタラクションを扱うオープンソースソフトウェアの発展に役立つ知見を共有した

本研究では将来のフレームワーク開発者がより少ないコストで Siv3D と同等以上のフレームワークを開発できるよう，Siv3D の設計決定と，開発省力化に関する情報を報告した．また，個人のオープンソースソフトウェア開発者が参考にできる，Siv3D のユーザコミュニティに関する施策について説明し，オープンソースソフトウェアのコミュニティ運営を改善するための知見を共有した．

大規模フレームワーク開発における C++ の使い勝手を改善する技法を提案，実証した

情報可視化や人と計算機のインタラクションに関する大規模フレームワークを，C++ を用いて開発する過程で顕在化した，C++ の使い勝手に関する課題を複数指摘し，言語仕様の制約の中でそれらの課題の解決もしくは緩和を図る独自の工夫に基づく API 設計を提案した．これらのアイデアについて，長年にわたるフレームワーク運用を通して不整合などの問題が生じないことを実証した．ソースコードは開発者が容易に再利用できるよう，オープンソースライセンスで公開した．

新しい C++ 開発環境へのフィードバックを提示した

直近に策定された C++ 規格や，新しい C++ ツールなど，開発環境の進化を先取的にプロジェクトで採用・実践し，C++ 言語仕様策定プロセスや，将来の C++ アプリケーション開発者が参照できるよう，Siv3D におけるそれらの活用事例と評価を報告した．

1.4 本論文の構成

本論文は全部で 7 章から構成されている．以下に各章の概要について述べる．

第 1 章は，本研究の意義や目的，本論文の構成について述べている．

第 2 章では，フレームワークの役割，使いやすさを実現するための要素，情報可視化やインタラクションの

ためのプログラミングフレームワークの関連研究，プログラミング言語 C++ の位置づけについて述べたあと，Siv3D の概要と利用事例について説明している．

第 3 章では，フレームワークの機能設計全般に関して，Siv3D の特徴的な事例を取り上げて議論を行っている．具体的には，フレームワークのアーキテクチャと基本クラスの設計，コンテンツ制作用のアセットを充実させることを目的とした，オープンソースの絵文字・アイコンフォントの活用，ロードに失敗したアセットをフォールバックさせる手段のデザイン，開発イテレーション高速化のためのスクリプティングシステム，そしてフレームワークの更新について述べている．

第 4 章では，情報可視化や人と計算機のインタラクションに関する大規模フレームワークを，C++ を用いて開発する過程で明らかになった，C++ 言語の使い勝手に関する課題を 7 つ指摘し，言語仕様の制約の中でそれらの課題の解決もしくは緩和を図った，Siv3D の API 設計の工夫を説明している．具体的には，次の項目である．

- 名前付き引数エミュレーションによるコンストラクタオーバーロードを用いた柔軟な図形定義
- ユーザ定義リテラルと，bool 型の strong typedef を用いたコードの表現力向上
- サブシステムの初期化と終了処理の効率的な実装
- C++ の RAII（Resource Acquisition Is Initialization）イディオムを利用したレンダーステートの管理
- ヘッダファイルのドキュメント性の向上
- C++ の難しさを低減させるための標準ライブラリ拡張
- 演算子オーバーロードの柔軟な活用

第 5 章では，オープンソースソフトウェアの普及・発展の観点から Siv3D のユーザコミュニティ運営における事例の分析と評価を行っている．とくに Siv3D 独自の取り組みである，利用者の協働開発参加を促すための「実装会」や「チャレンジ」といった施策と，フィードバックの収集方法について説明している．

第 6 章では，Siv3D の開発における C++ の最新仕様や開発ツールの利用状況を報告し，合わせて，Siv3D のようなフレームワークの開発が発展していくために，どのような C++ 言語の進化や，ツールの整備が望ましいか，運用経験を踏まえたフィードバックと提言を行っている．

第 7 章では，本研究の成果と貢献について，表現工学分野およびプログラミング分野の観点からまとめ，今後の課題と展望について述べている．

1.5 ソフトウェア開発の論文化

現代社会におけるソフトウェアの貢献については改めて述べるまでもないが，産業界でのソフトウェア活用の広がりに比して，品質の高いソフトウェア開発の技法や知見を共有し，継承，進化させていくプロセスへの学術界の貢献は，まだ発展の余地がある．関連する取り組みとして，日本ソフトウェア科学会は「ソフトウェア論文」というカテゴリを設け，ソフトウェア開発の論文化を奨励してきた [111][119].

> 日本ソフトウェア科学会編集委員会は，ソフトウェア研究の深化を目指すのみならず，先進的なアイデアを実現したソフトウェアの開発と普及を一層推進することを目的として，学会誌『コンピュータソフトウェア』の論文カテゴリとして「ソフトウェア論文」を新設いたしました．
> 発想，構成法，実装法などの点で優れ，実際に我々が使うことのできる先進的ソフトウェアの開発成果

> や，ソフトウェア設計・作成上の有益な知見を与える開発成果を学術論文としてまとめることを奨励し，その掲載を通じてソフトウェア文化の発展に寄与してゆきたいと考えています．
> しかし，ソフトウェア論文は従来の研究論文と異なり，どのようなソフトウェアについてどのような観点から論文を執筆すれば学術論文として認められるかに関して十分な社会的合意があるわけではなく，このことがソフトウェア開発の論文化の妨げとなってきました．
> 本「ソフトウェア論文」特集はこの問題を打破すべく企画したものです．優れたソフトウェア成果を積極的に論文化してご投稿いただき，編集委員会での議論と査読・改訂のプロセスを経ることによって，「良いソフトウェア論文」に関する合意を形成し，その具体例を実際に示すことを目標としています．
>
> 日本ソフトウェア科学会，「ソフトウェア論文」について [119] より引用

本論文は，Siv3D というプログラミングフレームワークを対象として，その設計や実装，普及といった側面に注目して論述したソフトウェア論文でもあり，とくに次のような点で特色がある．

- ソフトウェアが実際に利用されており，第三者による利用事例や制作物を多く見つけられる
- 筆者がソフトウェアの誕生から現在まで関わり続け，9 年間にわたる設計選択や環境の変化，運営の試行錯誤を経験している
- 「ソフトウェア論文」として日本ソフトウェア科学会から表彰を受けた 2 つの既発表論文の内容を含んでいる

本論文の執筆にあたっては，現役，そして将来のソフトウェア論文執筆者にとって，参考や比較，着想の材料となる議論を提供すること，ソフトウェア論文を書きやすくすることを心がけた．もし読者がソフトウェア開発者であれば，本論文を読みながら，自身の成果物の知見を論文として共有するとしたときに，どのように説明するかイメージし，実際にソフトウェア論文を書いてみようと少しでも思ってもらえると幸いである．

2 フレームワークの役割

2.1 フレームワークの目的

インタラクティブなアプリケーションを理想的に動作させるには，2D/3D グラフィックスのレンダラー，オーディオ再生，マウスやキーボードなど HID(ヒューマン・インタフェース・デバイス) の状態管理，画像処理，音声波形処理，通信，物理演算など，複雑で膨大なサブシステムを並行して緻密に制御する必要がある．それぞれのサブシステム単位では，小さな目的に特化したライブラリや SDK (Software Development Kit) が仕事を助けてくれるが，サブシステム間のデータ表現の差異を解消し，データの流れを統合し，全てのサブシステムが滞りなく協調して動作するシステムの構築には多大な労力がかかる．これらを自前で実装しようとすると，アプリケーションの本質的な部分の開発に使える時間が減ってしまう．

このような問題を解決するため，一般的なアプリケーション開発では，「フレームワーク」と呼ばれる開発者向けのソフトウェアが利用される．フレームワークは低レイヤのサブシステムを複数統合し，一連の機能への効率的なアクセスを一貫性のあるインタフェースで提供することで，アプリケーション開発の生産性を向上させる大規模なプログラミングライブラリである．フレームワークは特定の 1 つのアプリケーションのために開発されることもあれば，様々なアプリケーションの開発で利用できる汎用性を重視して開発される場合もある．

2.1.1 フレームワークに求められる条件

万人を納得させる「最も使いやすいプログラミング言語」が存在しないように，フレームワークの使いやすさも，その利用者の知識や経験，開発の目的，そして周辺ツールや新しい技術の登場など，様々な要因によって変わるものである．したがって，新しいフレームワークの開発，および運用を成功させる確率を高めるには，過去の議論や既存のフレームワークから共通して抽出されるノウハウや，避けるべき罠について知ったうえで，自らが開発するフレームワークの使用言語，対象とする分野と利用者，開発に割ける時間などの要素を考慮して，落としどころとなる設計や戦略を導き出す必要がある．

まずは「使いやすい」という言葉について考えてみよう．Nielsen が「Usability Engineering」[55] で述べたところによると，ユーザインタフェースの使いやすさを表す “usability” は，次の 5 つの要素から構成される．

- Learnability（学習のしやすさ）
  - ユーザがデザインに遭遇したときに，基本的なタスクを実行するのがどれだけ簡単であるか
- Efficiency（効率性）
  - ユーザがデザインを習得したら，どれだけ迅速にタスクを実行できるようになるか
- Memorability（覚えやすさ）
  - ユーザーが一定期間を空けて再びデザインに戻ったとき，どれだけ簡単に習熟度を回復できるか
- Errors（間違いにくさと間違いからの復帰のしやすさ）
  - ユーザがどれだけ間違いを起こしにくく，また，エラーからの回復がどれだけ容易にできるか
- Satisfaction（満足度）
  - デザインを使用することでどれだけ満足を得られるか

この 5 つの分類を参考に，プログラミングフレームワークが備えていると望ましい具体的な性質について，

引用や事例，筆者の経験をもとに列挙する．

導入コストが低いこと（Learnability, Efficiency）

フレームワークの機能を利用できるようになるまでの手順を自動化等によって少なくし，利用者の時間を浪費させないようにすることが望ましい．導入前の段階においても，複数のライブラリを検討している開発者のために，機能や制約，ベンチマーク，実用例に近いサンプルコードなどの検討材料をドキュメントに示すべきである．ソースコードのライセンスとして，一般的に普及しているオープンソースライセンスを採用することは，利用者がフレームワークの利用条件を的確に把握し，目的に適合しているかを確認するための手間を減らす．

オーバーヘッドが小さいこと（Efficiency）

とくに C/C++ ライブラリは，計算資源の制約のために選択されるシナリオが多く，各種処理でオーバーヘッドが生じることは好ましくない．例えば画像や音声波形に対するデータ処理を効率的に実装するには，データを格納したメモリ領域に直接アクセスする手段が用意されている必要がある．マルチコアや SIMD, GPU などの計算資源を有効に活用する並列・並行プログラムは，標準的な C++ のみで記述することが難しいため，フレームワークによって何らかのサポートがあることが望ましい．

信頼できること（Efficiency）

アプリケーション開発時に，フレームワーク由来のトラブルを避けるために，フレームワークに存在する不具合が少ないこと，フレームワークの挙動に疑問がある場合に内部コードにアクセスできること，フレームワークの開発体制が安定していて長期的にメンテナンスされる見通しがあること，利用者と開発側のコミュニケ―ション手段が用意されていることが望ましい．ただし，不具合の少なさを客観的に示すことは難しく，一般にはテストの充実度や，利用実績から判断される．また，API の破壊的変更が少ない，もしくは変更がある場合には適切な移行手段が示されていて，過去のコード資産を長期的に保守できるといったことも信頼性を高める要素である．

間違った使い方をしにくいこと（Errors）

論理エラーや，実行時性能の予期せぬ低下を避けるため，利用者が間違ったコードを書きにくい API 設計が求められる．Meyers[50] はこれを “Make interfaces easy to use correctly and hard to use incorrectly.” と表現している．C++ ではプログラムのエラーを表明する方法として，コンパイル時エラー，リンク時エラー，実行時エラーの 3 つが存在するが，その中でも最も早い段階である，コンパイル時にエラーが検出できることが望ましい．具体例として，C++ 標準ライブラリに採用された `std::format`のコンパイル時フォーマット文字列チェック [107] は，誤ったフォーマット指定子の記述の検出を，実行時のエラーからコンパイル時のエラーへ前倒しすることができる改良である．

学習しやすいこと（Learnability, Memorability）

学習しやすいフレームワークを作るには，API 設計の一貫性が重要である．具体例な項目として Reddy[67] は命名法，パラメータの順序，標準パターンの活用，メモリモデリングセマンティックス，例外の使用法，エラー処理を挙げている．1 つのプロジェクトで複数のライブラリを取り扱う場合，異なるスタイルが混在することで，コーディング時の認知負荷が増加し，誤った使い方をする原因になる．フレームワークには，こうし

たライブラリ間のギャップを吸収し，一貫したスタイルの API で機能を再提供することで，利用者が覚える必要のあるルールを減らすことが求められる．

教育的であること（Learnability, Satisfaction）

フレームワークの機能が最大限活用されるよう，利用者が適切な API を選択し，利用パターンや実装に役立つドメイン知識の習得を助ける仕組みを設けることが望ましい．ドキュメンテーション以外にも，利用者どうしの情報交換や，利用者からの質問と回答のストックを保存することも有意義であり，GitHub のサービスはそうしたコミュニティ活動を支援している．大規模なフレームワークでは，より柔軟で円滑なコミュニケーションのために，グループウェアや電子掲示板など，複数の手段でコミュニティを運用することもある [42]．適切に設計されたフレームワークやそのコミュニティは，プログラマの能力を引き出し，成長させ，優れたソフトウェアとソフトウェア開発者を生み出す機会を増加させることができる，

機能の拡張が容易であること（Efficiency）

あらゆる領域の課題を解決できる網羅的な API を作ることは不可能であり，そのため利用者は必要に応じてフレームワーク上に，よりドメインに特化したライブラリを実装することがある．再利用できる形で共有できるものはプラグインと呼ばれ，フレームワークの利用者間で共有される．Reddy[67] はアドオンの利点として，フレームワークの用途の拡張，ユーザ貢献によるコミュニティの発展，フレームワーク本体からの独立した配信による更新の小容量化，将来性の確保，コアシステムに手を入れずに機能を更新することによるリスクの分散を挙げている．

多くのプラットフォームに対応すること（Efficiency）

様々な OS やアーキテクチャの計算機が日常で使われるようになり，フレームワークのマルチプラットフォーム対応は不可欠になっている．情報可視化やインタラクションに注目すると，OS に応じてグラフィックス API やオーディオ API が異なることに注意すべきである．ただし，それらの差を吸収するオープンソースのライブラリが近年は多く公開されているため，特別な理由が無ければ自力で再実装をするよりも，既存のライブラリを活用することが望ましい．また，Web ブラウザさえあればあらゆる環境で動作する，Web アプリケーション形式での出力も有効な手段である．

管理者やメンテナのモチベーションが維持されること（Satisfaction）

オープンソースで開発されるソフトウェアの複雑さや専門性が高くなる一方で，その管理者やメンテナがプロジェクトから直接収益を得られるケースはいまだ少ない．ユーザとのやり取りで疲弊し，燃え尽き症候群となる事例も報告されている [66]．フレームワークの開発に関わるメンバーのモチベーション維持，負荷の削減のための施策も，フレームワーク運用における重要な要素である．

2.2 情報可視化とアクセス容易化のためのフレームワーク

本論文では，フレームワークの用途として，情報可視化と人と計算機のインタラクションに注目し，それらを開発するための汎用的な C++ フレームワークの設計と開発について議論を行う．

計算機と人間がインタラクション（相互作用）を通して仕事を達成するには，双方の情報伝達がスムーズに行われることが欠かせない．計算機から人間へ情報を伝達するには，まず計算機上のデータを人間が解釈しや

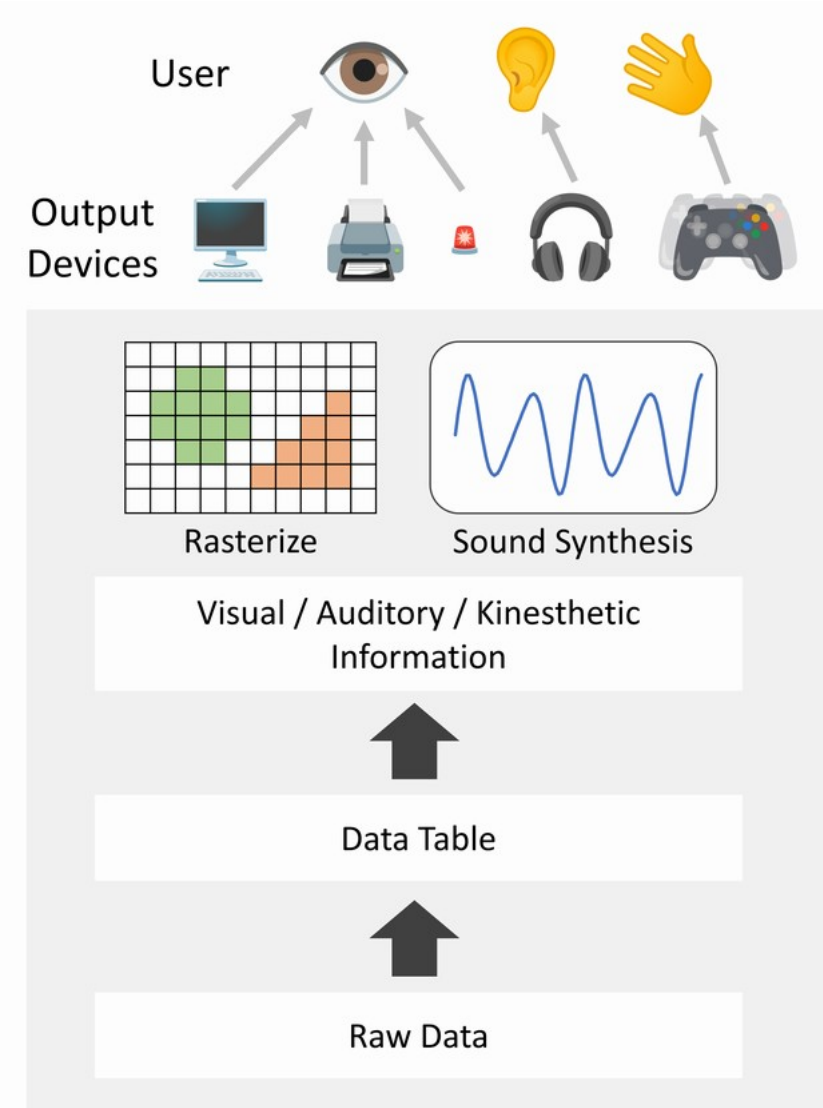

図 1　計算機から人間への情報伝達

すい抽象度の高いデータ形式に変換し，それをモニタやプリンタなどの物理的なデバイスのための入力データ形式に再変換する．最終的に，デバイスの出力装置を通して人間が感じることのできる光や振動に変換される（図 1）．

Card ら [9] は情報可視化のパイプラインをより詳細に分類し，計算機上に記録されている生データを抽出等の分析的な操作によって整理されたデータテーブルに変換する Data Transformation, データテーブルを視覚要素とその構造に変換する Visual Mappings, 視覚要素とその構造のパラメータを操作することによって人間に理解しやすいビューを作るための View Transformations という 3 つの変換操作を定義した．本研究で議論する情報可視化は，その中の Data Transformation と Visual Mappings が中心になる．

一方，人間から計算機への情報伝達は，圧力や位置，振動などの物理的な情報を，計算機に接続された入力装置のセンサが読み取り，センサの値を計算機が扱いやすい，時系列や分析済みデータに変換して計算機上に記憶することで達成される（図 2）．本研究で議論する処理は，センサの生データを時系列データや分析的デー

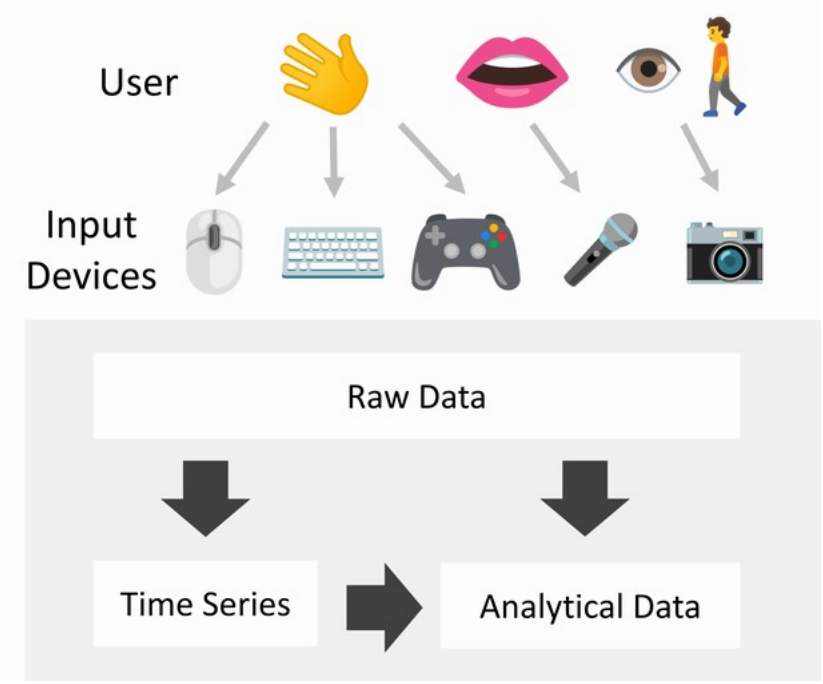

図 2　人間から計算機への情報伝達

タに変換し，プログラミングで扱いやすく抽象化する操作が中心となる．

コンピュータにおける情報可視化とアクセス容易化（インタラクション）のためにプログラムが行うべき仕事は，大きく分類して次のとおりである．

- テキストや図形，画像など，グラフィックスの出力
- オーディオなど，信号や波形の出力
- キーボードやマウスなど，入力デバイスの処理

これらを部分的に実現する機構として，プログラミング言語は標準入出力を備えているが，標準入出力はテキストデータという文字集合を入出力するのみで，テキストの色や書体，図形や画像の表示，マウス，オーディオなどを制御することはできない．プログラミング言語によっては，プリミティブなユーザインタフェースライブラリを備えている場合もあるが，複雑なアプリケーションを構築するには不十分な場合が多く，ほとんどのアプリケーション開発では，プログラミング言語規格とは独立した非標準のライブラリやフレームワークが利用される．情報可視化とアクセス容易化のための代表的なフレームワークを表 1 に示す．

C++ においては，過去に基本的な 2D グラフィックスの標準ライブラリを追加する提案が議論されたが [49]，次のような理由から C++ 標準化ワーキンググループのコンセンサスが得られず進展していない [2]．

- 標準ライブラリによって提供される，限られた性能と機能の API だけでは，多くの学習者が目標としているようなコンピュータゲームや複雑なアプリケーションの開発には適さない
- 言語の安全性を低下させる
- ライブラリ設計提案者の専門知識が不足している
- 実用されている既存のフレームワークと設計が異なり，学習した知識を再利用できない

表 1　情報可視化とアクセス容易化のための代表的なフレームワーク

フレームワーク	メイン言語	ライセンス
Processing[16]	Java	GPL および LGPL
libGDX[43]	Java	Apache-2.0
three.js[102]	JavaScript	MIT
pygame[64]	Python	LGPL-2.1
Ebiten[20]	Go	Apache-2.0
Godot[25]	C++	MIT
SFML[72]	C++	Zlib
SDL[71]	C++	Zlib
openFrameworks[60]	C++	MIT
Cocos2d-x[14]	C++	MIT
OGRE[57]	C++	MIT
Cinder[12]	C++	Apache-2.0

このような状況から，C++ におけるユーザインタフェースライブラリ開発については，将来にわたっても非標準ライブラリが主流であることは確実な状況である．

これに加えて，C++ はコーディングの難易度が高い問題と，コンパイル時間が長い問題があり，2000 年代以降，アプリケーションのすべてを直接構築するような汎用かつ高レイヤのフレームワークには，C++ 以外のプログラミング言語やゲームエンジンが採用されるようになった．その一方で，複雑なシステム API を小さいオーバーヘッドでラップする低レベルライブラリや，より高レイヤのフレームワークやエンジンで利用されることを想定した領域特化型のライブラリのような，実行時性能と移植性という C++ の利点を生かす形での活用は継続されていた．代表例として，様々なグラフィックス API を共通のインタフェースでラップした bgfx[3] や，アプリケーションに軽量な GUI を導入できる Dear ImGui[34] が挙げられる．

ところが，近年の C++ における注目すべき環境の変化として，C++11 以降，言語や標準ライブラリに大規模な改善が立て続けに導入され，コードの静的解析やホットリロードなど，C++ プログラミングの開発生産性向上につながるような周辺ツールの発展も急速に進んだ．こうした最新の状況を踏まえると，C++ で汎用かつ高レイヤなフレームワークを改めて開発することで，従来の C++ のイメージを払拭するような高性能なフレームワークを構築できる可能性が高くなっている．現在普及している C++ フレームワークの多くは C++03/C++11 時代からの API 設計に基づいているため，C++17/C++20 や現在のエコシステムを前提としたフレームワーク設計は新規性のあるチャレンジングな領域である．

2.3　C++ の特徴

Siv3D の開発において採用したプログラミング言語 C++ の特徴について述べる．

表 2　C++ の標準規格の変遷

規格の通称	発行年	備考
C++98	1998 年	初の国際標準
C++03	2003 年	マイナーな修正
C++11	2011 年	大幅な更新（これ以降が一般に「モダン C++」と呼ばれる）
C++14	2014 年	一部の機能の改善
C++17	2017 年	言語機能と標準ライブラリの拡張
C++20	2020 年	従来のコードを置き換えるような新しい機能の導入，ライブラリの拡張

2.3.1　C++ の歴史と利用事例

C++ は，ハードウェアの直接制御を効率的かつ高レベルな抽象化によって行うことを目的に開発されたプログラミング言語である．1979 年に Bjarne Stroustrup によって，C++ の原型となる「C with Classes」の開発が始まり，1985 年に C++ の初めての商業リリースが行われた．1989 年から標準化に向けた作業が始まり，1998 年に初めて国際標準が承認された．その後は現在まで，表 2 のように標準規格が更新されている．C++ の標準化委員会は新しい規格案（既存の仕様の修正や新しい機能）の提案をオープンで受け付けているが，実際に採択に至るまでにはペーパーの執筆とリビジョン，委員会の国際会議への出席が必要であり，また近年の標準規格の更新サイクルは 3 年ごとで，議論の結果によっては 3 年以上にわたる対応が求められることから，個人が気軽に言語仕様の進化に貢献できるようなシステムにはなっていない [80].

2011 年以降の C++ で強化された機能として，次の項目が挙げられる [82].

- メモリモデル
- 型安全な並行性サポート
- 型推論
- コードの簡易化
- ムーブセマンティクス
- コンパイル時プログラミング
- ジェネリックプログラミング
- メタプログラミング

2015 年時点で C++ のプログラマは 400 万人を超えると推計されている [82]．C++ が使われている代表的なソフトウェアやシステムには次のような例がある．より具体的なアプリケーションは Bjarne Stroustrup によるリスト [81] が詳しい.

- OS (Windows)
- Web ブラウザ (Google Chrome, Firefox)
- オフィススイート (Microsoft Office)
- ゲームエンジン (Unreal Engine)
- グラフィックソフトウェア (Adobe Photoshop)

- データ解析ソフトウェア (ROOT)
- コンソール機向けゲーム
- Web サービスのバックエンド
- 組込みシステム

2.3.2 C++ の構成要素

利用者という立場に立ったとき，C++ は単なる言語規格ではなく，以下のような要素によって構成されるツールセットであると Bjarne Stroustrup は述べている [82].

- コア言語
  - C++ の基本的な文法
- 標準ライブラリ
  - C++ の実装に標準で備えられているライブラリ群．C++ の標準ライブラリは約 100 種類のヘッダから構成され，標準入出力やデータ構造，アルゴリズム，マルチスレッドサポートなどの機能が提供される
- その他の多くのライブラリ
  - 利用者によって開発された非標準の様々なライブラリ
- 大量の（しばしば古い）コードベース
  - 書籍やコードリポジトリに蓄積された過去のコード
- ツール（他の言語を含む）
  - 統合開発環境 (IDE) やコードエディタ，デバッガ，コード解析ツール，パフォーマンス測定ツールなど，C++ プログラミングを支援するソフトウェア
- 教育とトレーニング
  - 書籍，オンライン記事，スライド，大学講義，オンラインコースなど，C++ の教育のための資料とシステム
- コミュニティによるサポート
  - 各地のユーザグループやインターネット上のコミュニティ，様々なカンファレンスイベントなど，C++ 利用者の情報交換を促進するための，多人数による活動全般

ソフトウェア開発やフレームワークの開発に C++ を採用する判断においては，C++ の仕様上の利点だけではなく，こうしたエコシステムを含めた総合的な評価が行われる点に注意したい．C++ を採用したソフトウェア開発は，これらのエコシステムの恩恵にあずかることで高い生産性を発揮できるし，C++ よりも優れた言語仕様を持つ言語が提案されたとしても，このようなエコシステムの構築には時間がかかるため，すぐに移行が進むことはないだろう．

2.3.3 他の手続き型言語との比較

C++ を他の手続き型言語と比較したときの特徴を次に示す．

C 言語に対して

- より強い型システム

- より強力なコンパイル時処理 (template や constexpr)
- より高い抽象化レベル
- よりカスタマイズできる型 (C++ のクラス)

Java に対して

- 複数のパラダイムへの対応 (手続き型，オブジェクト指向，ジェネリック，関数型)
- メモリアロケーションの厳密な制御
- 決定論的なリソースの寿命管理
- より少ないメモリ消費

Python に対して

- 静的な型付け
- より高速な実行速度
- より難解な文法，言語知識の要求
- より多くのエラーをコンパイル時に捕捉できる
- よりコンパクトな標準ライブラリ

このような C++ の特徴は Siv3D の設計に強く影響を与えており，本論文でも，C++ の型システムを生かした機能に関して 4.1, 4.2 で，決定論的なリソースの寿命管理を生かした機能に関して 4.4 で，言語知識の難解さやコンパクトな標準ライブラリへの対応に関して 4.6 で，型のカスタマイズに関して 4.7 において，それぞれ議論を行っている．

2.4 Siv3D の概要

本論文では，筆者が開発したオープンソースのフレームワーク「Siv3D」を題材に，情報可視化とインタラクションのための C++ フレームワーク設計について論じる．

2.4.1 Siv3D の歴史

Siv3D は，2011 年に Windows 向けの C++ ゲーム開発ライブラリとして，最初のバージョンがリリースされた．2016 年にプロジェクト名を OpenSiv3D と改め，MIT ライセンスのもとでオープンソース化し，同時に macOS への対応を始めた．2017 年には Linux に対応し，2019 年には Web ブラウザへの対応も開始された．これまで 5,000 回以上のコードの追加や変更（コミット）が公開リポジトリ上で行われ，32 人がコミッタとして関与するなど，活発な開発とユーザコミュニティ運営が続いている．2021 年時点で公開されている最新のバージョンは 0.6.3 であり，本論文の説明はこのバージョンに準拠したものである．Siv3D という名称の由来は，筆者が Siv3D の開発を始めたときに在学していた，渋谷教育学園幕張高等学校という学校名である．

2021 年現在，Siv3D は以下の Web ページとロゴ（図 3）を公開している．

- Web サイト: `https://siv3d.github.io/`
- コードリポジトリ: `https://github.com/Siv3D/OpenSiv3D`
- 日本語リファレンス: `https://zenn.dev/reputeless/books/siv3d-documentation`

図 3　Siv3D のロゴ

- 英語リファレンス: `https://zenn.dev/reputeless/books/siv3d-documentation-en`

2.4.2　Siv3D の機能

Siv3D の機能をコードリポジトリ [75] より引用して次に示す.

- グラフィックス
  - 発展的な 2D グラフィックス
  - 基本的な 3D グラフィックス (Wavefront OBJ, 基本形状)
  - カスタム頂点・ピクセルシェーダ (HLSL, GLSL)
  - テキストレンダリング (Bitmap, SDF, MSDF)
  - 画像形式 (PNG, JPEG, BMP, SVG, GIF, Animated GIF, TGA, PPM, WebP, TIFF)
  - Unicode 14.0 emojis と 7,000 種類以上のアイコン
  - 画像処理
  - ビデオレンダリング
- オーディオ
  - 音声形式 (WAVE, MP3, AAC, OggVorbis, Opus, MIDI, WMA, FLAC, AIFF)
  - 音量やパン，スピード，ピッチの調整
  - ストリーミング再生
  - フェードイン，フェードアウト
  - ループ
  - ミキシングバス
  - フィルタ処理 (ローパスフィルタ，ハイパスフィルタ, エコー, リバーブ)
  - FFT
  - サウンドフォントレンダリング
  - テキスト読み上げ
- 入力
  - マウス
  - キーボード
  - ゲームパッド
  - ウェブカメラ
  - マイク
  - Joy-Con / Pro Controller
  - XInput ゲームパッド

  - ペンタブレット
  - Leap Motion
- ウィンドウ
  - フルスクリーンモード
  - 高 DPI サポート
  - ウィンドウのスタイル
  - ファイルダイアログ
  - ドラッグ&ドロップ
  - メッセージボックス
  - トースト通知
- ネットワーク・通信
  - HTTP クライアント
  - TCP 通信
  - Serial 通信
  - プロセス間通信 (pipe)
- 数学
  - ベクトルと行列クラス (`Point`, `Float2`, `Vec2`, `Float3`, `Vec3`, `Float4`, `Vec4`, `Mat3x2`, `Mat3x3`, `Mat4x4`, `SIMD_Float4`, `Quaternion`)
  - 2D 形状クラス (`Line`, `Circle`, `Ellipse`, `Rect`, `RectF`, `Triangle`, `Quad`, `RoundRect`, `Polygon`, `MultiPolygon`, `LineString`, `Spline2D`, `Bezier2`, `Bezier3`)
  - 3D 形状クラス (`Plane`, `InfinitePlane`, `Sphere`, `Box`, `OrientedBox`, `Ray`, `Line3D`, `Triangle3D`, `ViewFrustum`, `Disc`, `Cylinder`, `Cone`)
  - 色クラス (`Color`, `ColorF`, `HSV`)
  - 曲座標系クラス
  - 2D / 3D 幾何計算
  - Rectangle packing
  - 平面細分割
  - 色空間
  - 疑似乱数生成器
  - 補間，イージング，スムージング
  - パーリンノイズ
  - 数式パーサ
  - ナビメッシュ
  - 拡張数値型 (`HalfFloat`, `int128`, `uint128`, `BigInt`, `BigFloat`)
- 文字列
  - 文字列クラス (`String`, `StringView`)
  - Unicode 変換
  - 正規表現
  - 文字列フォーマット
  - テキストファイル読み書き

  - CSV / INI / JSON / XML / TOML パーサ
  - CSV / INI / JSON 出力
- その他
  - 基本的な GUI (ボタン，スライダー，ラジオボタン，チェックボックス，テキストボックス，リストボックス，カラーピッカー)
  - 2D 物理エンジン (Box2D)
  - 配列クラス (`Array`, `Grid`)
  - Kd-tree
  - 非同期ファイルロード
  - データ圧縮 (zlib, Zstandard)
  - シーン遷移
  - ファイルシステム
  - ディレクトリ監視
  - QR コード
  - GeoJSON
  - 日付と時刻
  - 時間計測
  - ロギング
  - シリアライズ
  - UUID
  - 子プロセス管理
  - クリップボード
  - 電源管理
  - スクリプティング (AngelScript)

プログラマが Siv3D を利用してアプリケーションを開発するとき，そのコードの大部分は，Siv3D により提供される型や関数を使って記述されるため，一般的な C++ フレームワークよりも DSL (Domain-Specific Language) としての性格が強い．テキストや画像，図形をインタラクティブに表示する Siv3D のサンプルプログラムを次に示す．

```
1  # include <Siv3D.hpp>
2
3  void Main()
4  {
5    // 背景を水色に
6    Scene::SetBackground(ColorF{0.8,0.9,1.0});
7
8    // フォントを作成
9    const Font font{ 60 };
10
11   // 画像ファイルからテクスチャを作成
12   const Texture texture{ U"image.png" };
13
```

```
14  // テクスチャを描画する座標
15  Vec2 pos{ 200,200 };
16
17  while (System::Update())
18  {
19    // テクスチャを描く
20    texture.draw(pos);
21
22    // 画面の中心にテキストを表示
23    font(U"Hello, Siv3D!).drawAt(Scene::Center(), Palette::Black);
24
25    // マウスカーソルを中心に半透明の赤い円を描く
26    Circle{ Cursor::Pos(), 40 }.draw(ColorF{ 1, 0, 0, 0.5 });
27
28    // キーボードの[A] が押されたら
29    if (KeyA.down())
30    {
31      // メッセージを出力
32      Print << U"Hello!";
33    }
34
35    // 画面上に"Move" と書かれたボタンを表示し，それが押されたら
36    if (SimpleGUI::Button(U"Move", {640,40}))
37    {
38      // 画面上のランダムな位置に座標を移動
39      pos = RandomVec2(Scene::Rect());
40    }
41  }
42 }
```

2.5 Siv3D の利用実績と活用事例

Siv3D のソフトウェア開発キット（SDK）のダウンロード数は，年間 1 万回に及ぶ（2020 年 8 月〜2021 年 7 月）．Siv3D の活用状況を調べるために，2021 年 10 月時点で，最大手のソースコード共有サイト GitHub において，「Siv3D」というキーワードを含む C++ プログラムファイルを検索したところ，1 万件以上がヒットした（図 4）．この結果の中には GitHub 上で非公開設定になっているリポジトリは含まれず，そもそもコードを GitHub で公開せずに開発する事例のほうが多いと考えられることから，少なくとも数万〜数十万件のプログラム開発で Siv3D が活用されたと見積もられる．

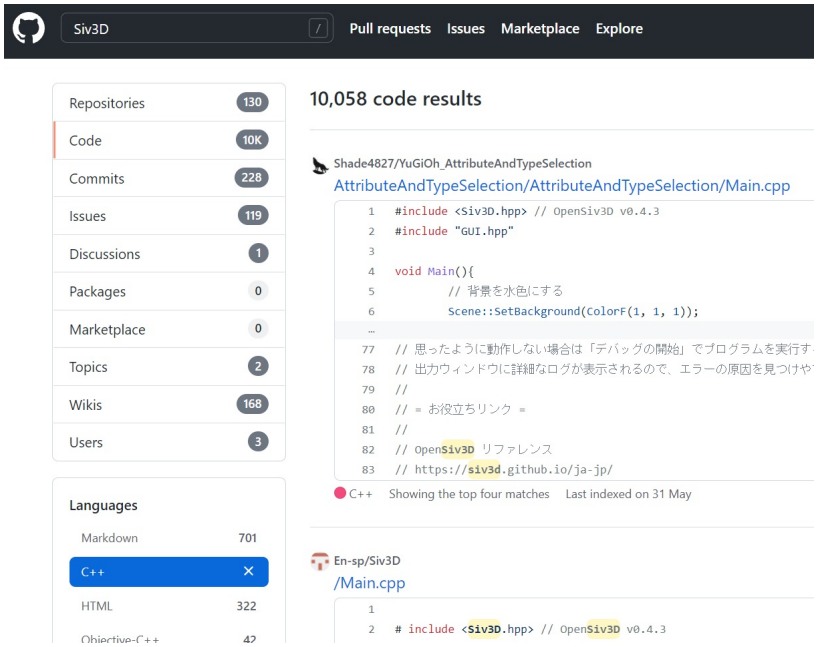

図 4　ソースコード共有サイト GitHub で，キーワード「Siv3D」を含む C++ ファイルを検索すると，1 万件以上がヒットした

Siv3D は，コンピュータゲームやメディアアート展示，研究，教育プロジェクトでの活用事例が多数報告されている．例えば，国内で開催される全国高等専門学校プログラミングコンテストでは，例年複数の学校が Siv3D を開発フレームワークとして採用している [116][117] ことから，教育機関でプログラミングを学ぶ若い世代にも浸透していることがわかる．

学術研究分野での活用状況を調査するために，論文検索システム Google Scholar におけるキーワード検索でヒットした，Siv3D を研究に活用している論文を列挙すると，実験ソフトウェアのビジュアライザ [36][37][38][101][112][110][114][118][115]（図 5，図 6，図 7），特殊な入力デバイスの制御 [108][113]（図 8），ロボット制御 [100] などの利用事例が見つかった．

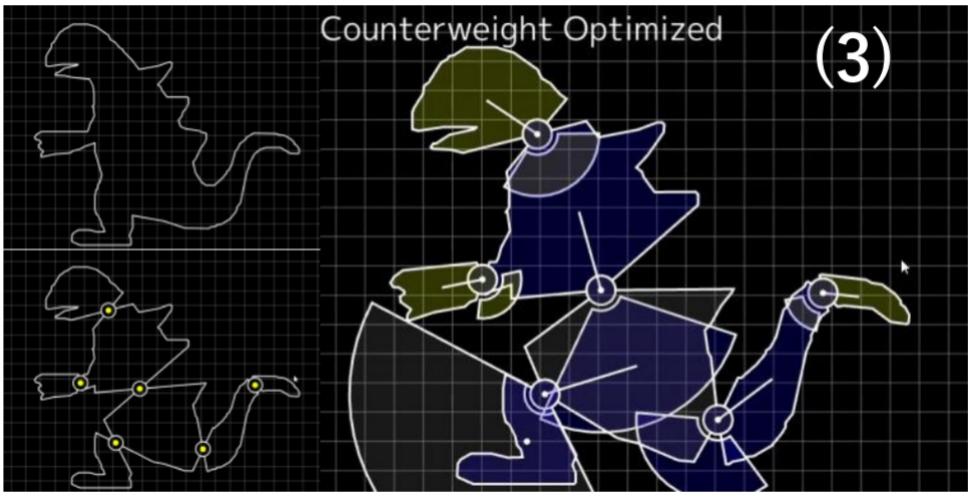

図 5　2D 図形描画機能が使われている事例（図版は [101] より引用）

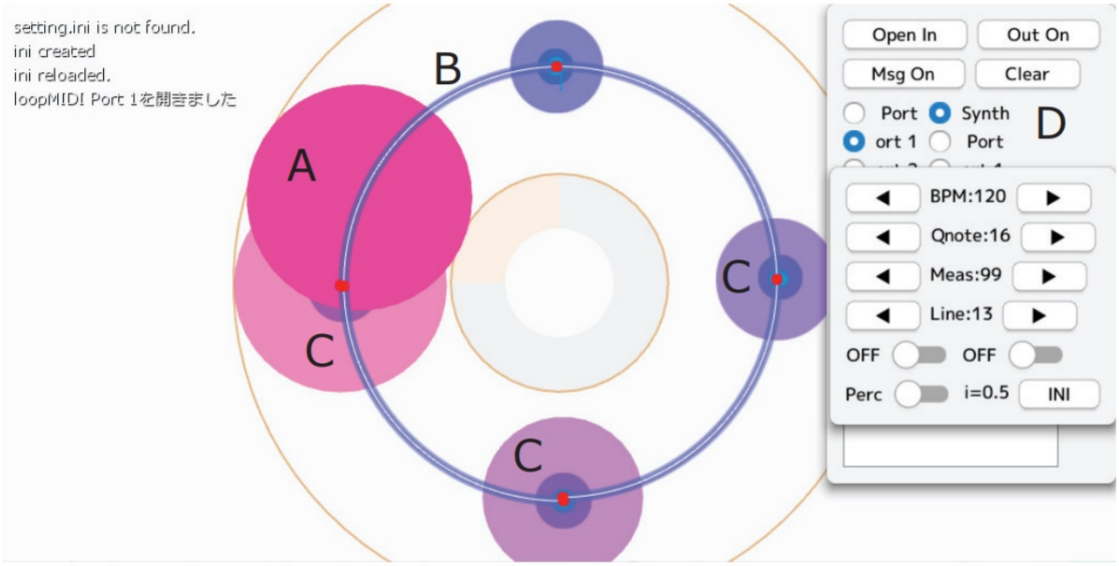

図 6　GUI 機能が使われている事例（図版は [114] より引用）

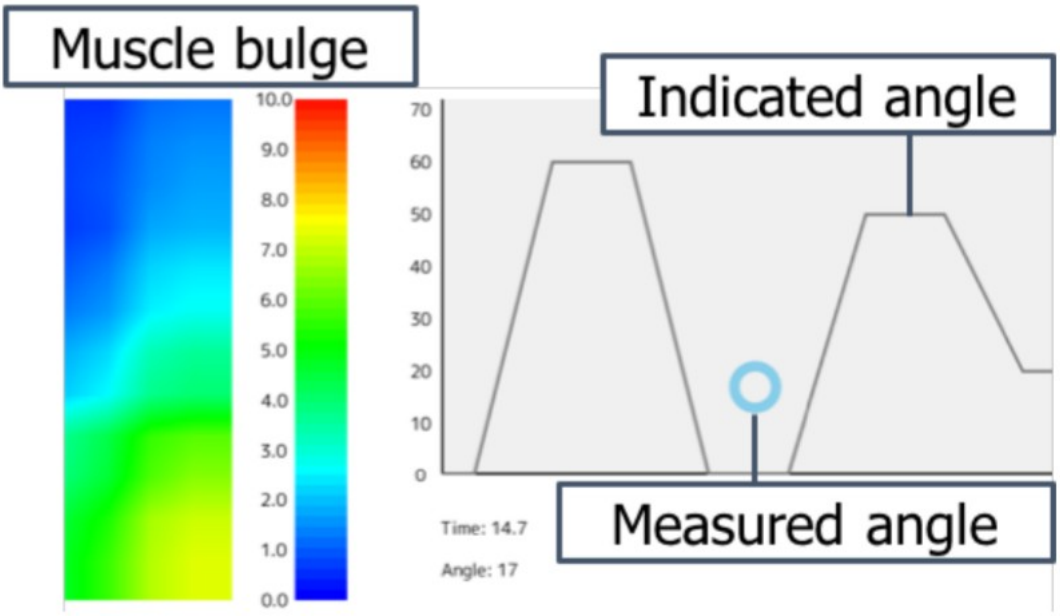

図 7　センサの状態をリアルタイムに可視化する事例（図版は [37] より引用）

図 8　入力デバイス処理機能が使われている事例（図版は [113] より引用）

Siv3D を利用して開発されたソフトウェアで，インターネットで入手可能なものをいくつか選んで紹介する．

Siv3D for Kids

公開 URL: `https://siv3d-for-kids.github.io/`

筆者によって発表された教育用のビジュアルプログラミング言語である．ゲームやアプリケーションの Siv3D プログラム (C++) の一部を，キーボードを使わずにタッチだけで操作できるグラフィカルユーザインタフェース (GUI) を通してカスタマイズし，オリジナルのプログラムを作るツールである．Siv3D に標準搭載される絵文字をゲームのアセットとして用いることで，2000 種類以上のアイテムやキャラクター，動物などの絵を登場させられる（図 9，図 10）．絵文字の情報を物理演算や幾何計算機能と連係させることで，絵文字を積み重ねる物理演算や，絵文字を変形，加工するなどの多様な表現技法を実現している [97].

図 9　「Siv3D for Kids」のプログラミング画面

図 10　「Siv3D for Kids」のプログラム実行画面

Enrect

公開 URL: `https://enrect.org/`

筆者によって発表された，アクセシビリティを考慮して開発されたビジュアルプログラミング言語である [99]．Siv3D for Kids 同様，Siv3D に標準搭載される絵文字をアセットとして用いる（図 11，図 12）．ビジュアルプログラミングエディタでは，ブロックとノードを使って C++ 風のコードを記述でき，そのコードは内部で Siv3D 標準機能のスクリプト言語のプログラムに変換され，Siv3D によって実際に実行される．

図 11　「Enrect」のプログラミング画面

図 12　「Enrect」のプログラム実行画面

One week, My room

公開 URL: `https://www.freem.ne.jp/win/game/14961`

2017 年にサークル常夜灯（`http://de.2-d.jp/joyato/`）によって公開された無料プレイのアドベンチャーゲームである（図 13）．日本語，英語，中国語でリリースされ，制作者の発表で 2 万回ダウンロードされた．ゲームレビューや，動画サイト上でのゲームプレイ実況配信が，英語も含め数十件以上投稿されるなど，その反響を確認できる．

図 13　「One week, My room」のプレイ画面

CORGI JUMP

公開 URL: `https://voidproc.itch.io/corgi-jump`

2017 年に voidproc（`https://itch.io/profile/voidproc`）によって公開された無料プレイのアクションゲームである（図 14）．機敏に動くキャラクターアニメーションと，サウンドエフェクト，音楽の連係は，Siv3D の機能を効果的に活用している．

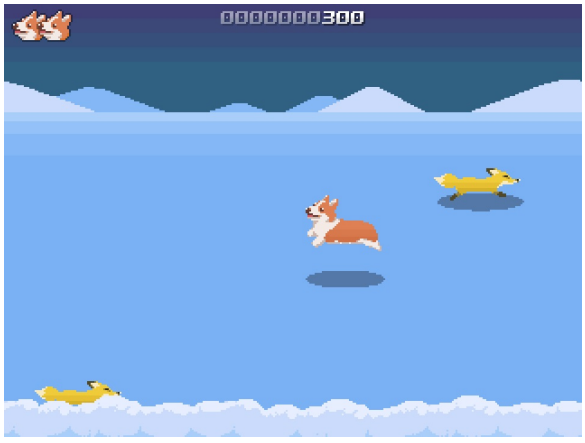

図 14　「CORGI JUMP」のプレイ画面

破損した JPEG を修復するゲーム

公開 URL: `https://siv3d.jp/web/sample/jpeg_game/jpeg_game.html`

2021 年に筆者によって発表された Web ブラウザ向けゲームである（図 15）．ゲーム開始時にパスワードが無作為に決められ，プレイヤーは画面下部のダイアルを回してパスワードの解除を目指す．画面中央には，メッセージを構成する個々の文字の画像が，JPEG グリッチエフェクトを適用された状態で表示されており，そのノイズの強度はプレイヤーが選択したダイアル番号と正解のダイアル番号との近さによって変化する（図 16）．プレイヤーは画像の復元度を見極めながらパスワードを探り当て，なるべく短時間でメッセージを復元することを目指す．発表直後に Twitter 上で話題を集め，公開から 3 日間で 50,000 件のアクセスと，約 1,000 件のプレイ結果の投稿（Twitter ハッシュタグ`#JPEG_Game`）があった．Siv3D が提供する JPEG エンコード API を活用して正解画像を JPEG バイナリに変換し，その一部のビットを異なる値に書き換えてから再度デコード API を用いて読み込むことで，このような表現を実現している．

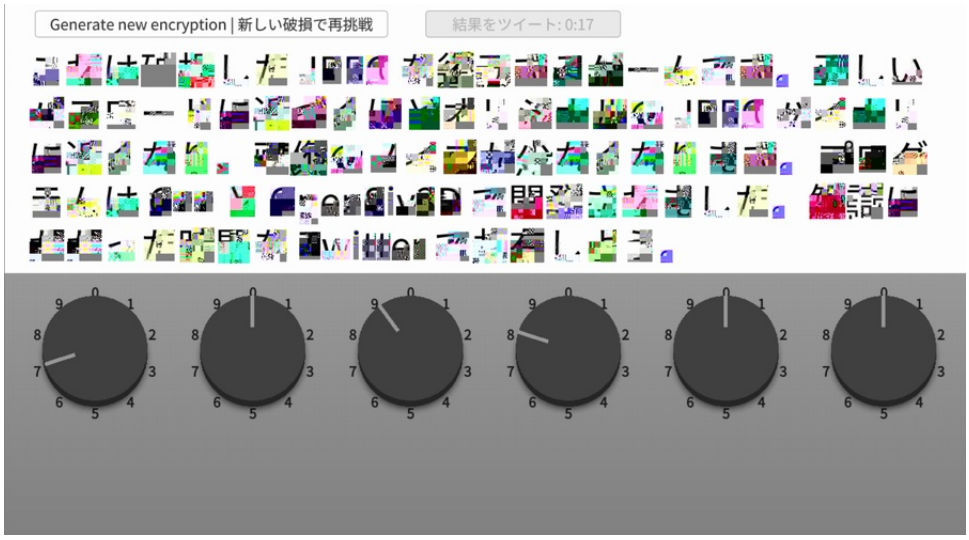

図 15　「破損した JPEG を修復するゲーム」のプレイ画面

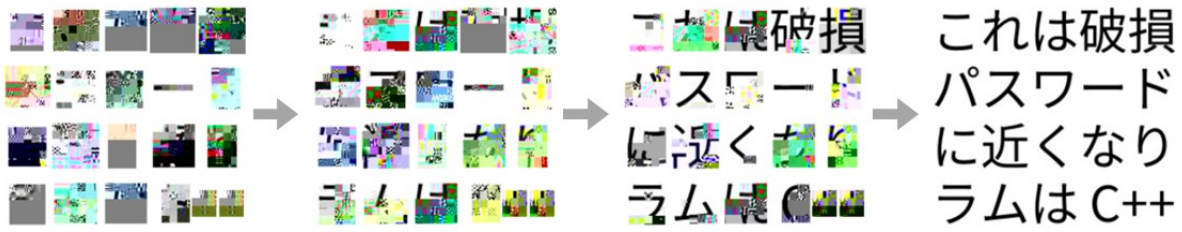

図 16　正解のパスワードに近づくと JPEG の破損レベルが小さくなる

忙しいブロックくずし

公開 URL: `https://twitter.com/reputeless/status/1112674711245193217`

2019 年に筆者によって発表されたゲームコンセプト映像である．ブロックくずしゲームをプレイできる小さなウィンドウのタイトルバーが，そのウィンドウの下層にある大きなブロックくずしのパドルになるという二層構造のブロックくずしのゲームアイデアが実演されている（図 17）．この映像は Twitter 上で 16 万回再生された．国外の Web 開発者によるプレイアブルなリメイク「Brickception」(`https://brickception.xyz/`) が発表され，複数の Web メディア記事に取り上げられた [22][44].

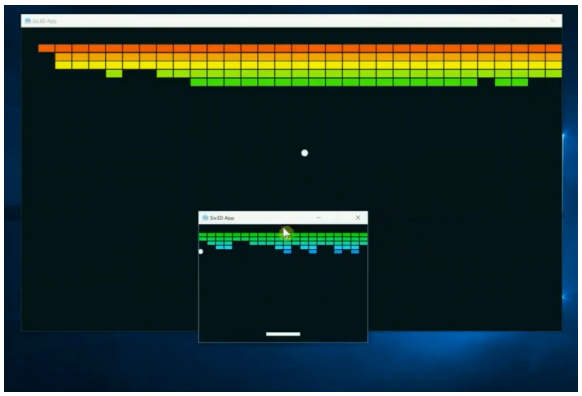

図 17　「忙しいブロックくずし」の映像のワンシーン

100px 以内にある点集合で凸包を作りながら移動する生き物

公開 URL: `https://twitter.com/reputeless/status/1082471025449680896`

2019 年に筆者によって Twitter 上に投稿された映像作品である（図 18）．画面上に無作為に散らばる点の上を，3 匹の生き物の核が一定の軌道に沿って移動し，その核の周囲 100px 以内にある点の集合の凸包を生き物の輪郭として可視化したものである．プログラムにもかかわらず，実際の生き物のような不思議な動きを見せることが大きな反響を呼び，Twitter で拡散された映像は 13 万回再生された．

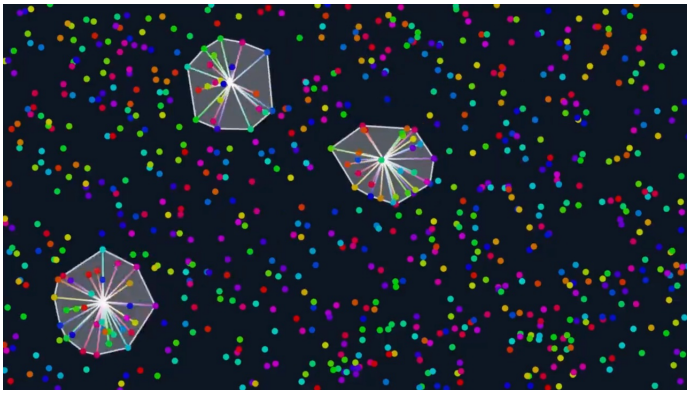

図 18　「100px 以内にある点集合で凸包を作りながら移動する生き物」の映像のワンシーン

漢字 Wordle

公開 URL: `https://twitter.com/Reputeless/status/1485186067120926721`

2022 年に筆者によって発表されたゲームコンセプト映像である．同年にインターネット上でブームとなった文字遊びゲーム「Wordle」を日本語にアレンジしたもので，オリジナルゲームで用いられているアルファベットの代わりに，漢字を構成する部品が問題解決のヒントになるというアイデアが実演されている（図 19）．この映像は Twitter 上で 6 万回再生され，Wordle ブームで生まれた日本語版亜種の一例として新聞記事に取り上げられた [15]．

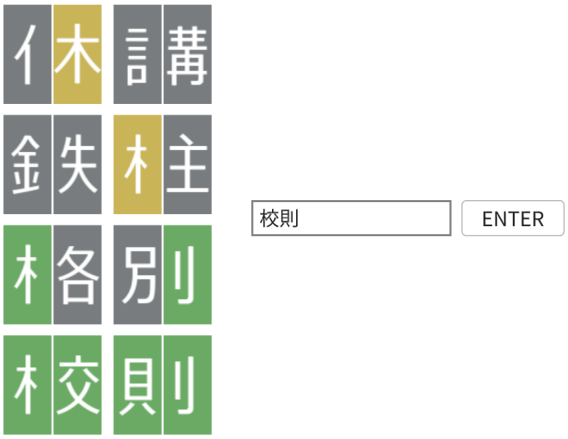

図 19　「漢字 Wordle」の映像のワンシーン

3　機能設計の課題と Siv3D での解決

本章では，フレームワークの構成を俯瞰するために，Siv3D の主要なサブシステムとそれに関連するクラスの実装方法について説明する．

3.1　フレームワークのアーキテクチャと基本クラス

図 20 は Siv3D を構成する主要なサブシステムの構成図である．各サブシステムは必要に応じた粒度でさらに細分化された機能を提供する．ユーザから見た Siv3D フレームワークは，約 600 種類のクラスと，それらが持つメンバ，さらに多くの定数・関数の集合である．詳細な機能は Siv3D の公式リファレンス [90] で参照できるため，本節では大きな分類と設計決定について説明する．

3.1.1　コアシステム

動的配列やハッシュテーブル，数学関数や文字列処理，ファイルシステムなど，ユーザのコードに頻繁に登場するデータ構造や低レイヤ，C++ の言語の柔軟性を拡張するための機能群である．汎用性が高く，多くのサブシステムで使われるため，なるべく高速に動作させる必要があり，標準ライブラリやサードパーティライブラリのラッパーであったり，OS の API を直接使用するような低レイヤの実装である場合が多い．

数値型

計算で使われる整数や浮動小数点数などの数値型は，消費するメモリのサイズや精度，計算速度のトレードオフのため，C++ を含むほとんどのプログラミング言語で複数の種類が提供されている．C++ フレームワークにおいては，数値型について次の 2 つの課題を解消する必要がある．

- コア言語の予約語で使われている整数型はサイズを保証していない
- コア言語が提供しない数値型を必要とするアプリケーションがある

C++ の `long`型は，データ型モデルによってサイズが異なり，Windows で採用されている LLP64 モデルでは 32-bit であるが，macOS や多くの Linux で使われている LP64 モデルでは 64-bit になる．したがって，`long`型を使用するアプリケーションコードは移植性の問題が生じる可能性がある．C++ では特定の型の使用を禁止することはできないため，より合理的で使いやすい API を提供することで，利用者に移植性に優れた数値型を使用させるべきである．

また，コンピュータグラフィックス分野で，float 型よりも低い精度で十分である画像データの表現に使われる半精度浮動小数点数型 binary16 や，深層学習用途で使われる bfloat16 型，また，64-bit で表現できるよりも大きな数を扱うための 128-bit 整数型，多倍長整数・浮動小数点数型は，いずれも近年のアプリケーション開発でのニーズが高まっており，C++ 標準でのサポートも提案されているが [59][106]，C++20 時点では未採択である．フレームワークのアプリケーションで，こうした拡張数値型の使用が想定される場合はサポートできると便利である．

Siv3D ではこうした背景を踏まえ，表 3 に挙げた数値型を独自に定義し提供する．8-bit から 64-bit 整数型は C++ 標準の`<cstdint>`ヘッダに定義される `std::int8_t`や `std::uint64_t`などのエイリアスである．128-bit 整数型は abseil-cpp[27] が提供する `absl::int128`および `absl::uint128`のエイリアスである．`size_t`型は

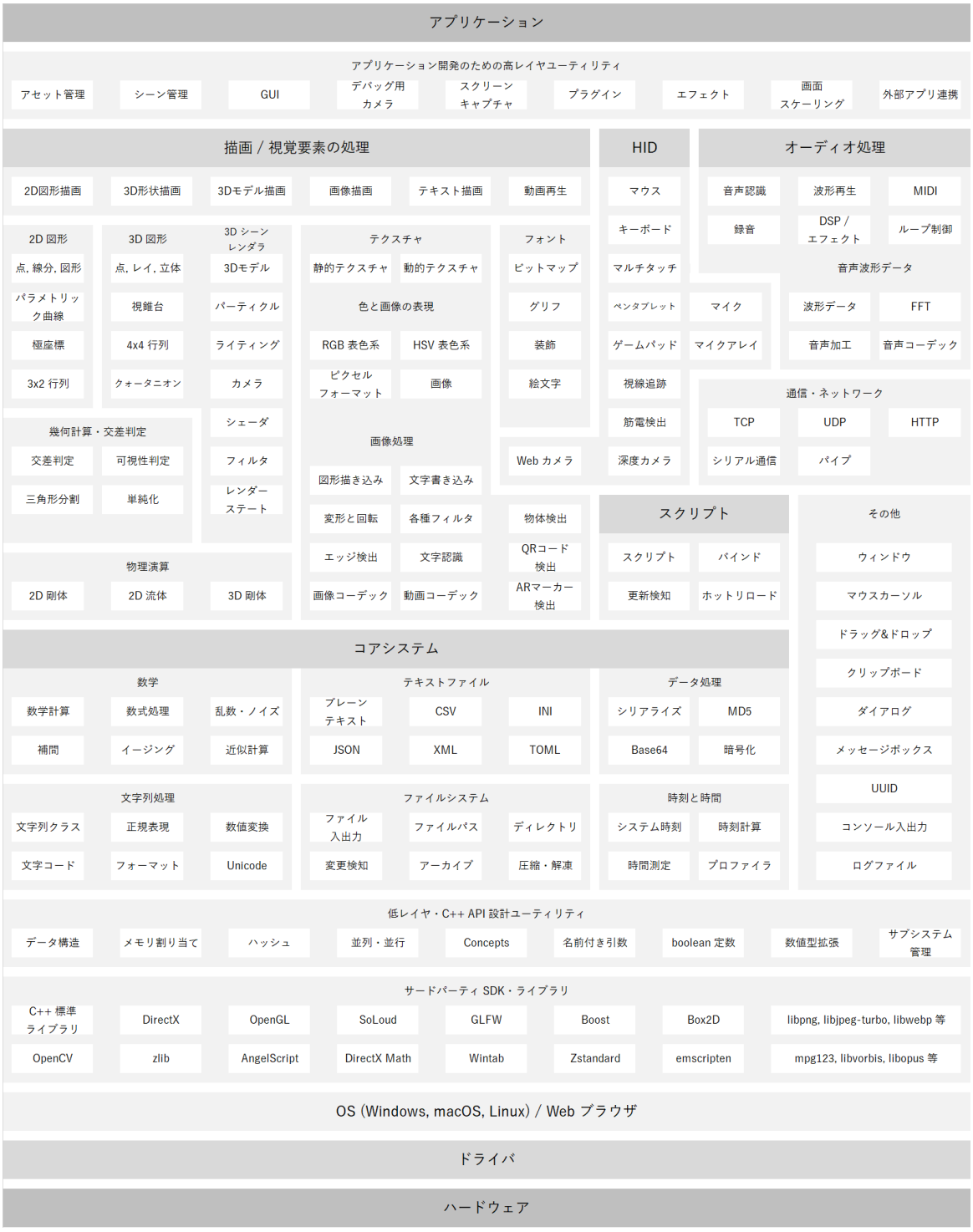

図 20　Siv3D のアーキテクチャ

表 3　Siv3D で使われる数値型

型	説明	C++ のコア言語機能に含まれるか
`bool`	boolean 型	○
`int8`	符号付き 8-bit 整数型	○ (`std::int8_t`)
`uint8`	符号無し 8-bit 整数型	○ (`std::uint8_t`)
`int16`	符号付き 16-bit 整数型	○ (`std::int16_t`)
`uint16`	符号無し 16-bit 整数型	○ (`std::uint16_t`)
`int32`	符号付き 32-bit 整数型	○ (`std::int32_t`)
`uint32`	符号無し 32-bit 整数型	○ (`std::uint32_t`)
`int64`	符号付き 64-bit 整数型	○ (`std::int64_t`)
`uint64`	符号無し 64-bit 整数型	○ (`std::uint64_t`)
`int128`	符号付き 128-bit 整数型	
`uint128`	符号無し 128-bit 整数型	
`size_t`	アーキテクチャ依存のサイズ整数型	○
`HalfFloat`	半精度浮動小数点数型 (binary16)	
`float`	単精度浮動小数点数型	○
`double`	倍精度浮動小数点数型	○
`BigInt`	多倍長整数型	
`BigFloat`	有効桁数 100 桁の浮動小数点数型	

32-bit OS 向けのビルドでは 32-bit, 64-bit OS 向けのビルドでは 64-bit になる，サイズがアーキテクチャ依存の型である．`HalfFloat`は `uint16`へのビット演算により binary16 をエミュレートするクラスで, float 型への変換と float 型からの変換をサポートする．`BigInt`はメモリが許す限り大きな整数を表現できるクラスで，Boost.MultiPrecision[48] の `boost::multiprecision::cpp_int`をラップしたクラスである．`BigFloat`は有効桁数 100 桁で浮動小数点数を表現するクラスで，Boost.MultiPrecision の `boost::multiprecision::cpp_dec_float_100`をラップしたクラスである．Boost.MultiPrecision の採用理由は，任意精度型に対する計算操作だけでなく数学関数の実装も充実していたこと，寛容なライセンス (Boost Software License)，利用実績である．

文字と文字列

ひらがなや漢字を含む文字列について，各文字へのランダムアクセスをシンプルなコードで記述できるよう，現在の Siv3D では UTF-32 を基本の文字コードとしている．Siv3D では当初 Windows 版から開発が始まった歴史的経緯で，Windows のネイティブ API と互換性のある `wchar_t`型 (Windows では UTF-16) を標準の文字型として採用していたが，`wchar_t`型は処理系により文字コードとサイズが異なるため移植性に難があった．また，UTF-8 は空間効率に優れるが，ひらがなや漢字，複数のコードポイントからなる絵文字を含む場合の取り扱いが難しかった．そこで，マルチプラットフォーム対応を機に，文字コードを UTF-32 に共通化することにした．Siv3D では C++ 標準の `char32_t`のエイリアスである `char32`型，`std::u32string`をラップして 4.6 で述べるように便利なメンバ関数を追加した `String`型，`std::u32string_view`をラップした

表 4　Siv3D で使われる文字・文字列型

型	説明
`char32`	UTF-32 のコードポイントを表現する型
`String`	char32 の集合である文字列型
`StringView`	所有権を持たない文字列型

`StringView`型を提供する．

コードポイントとグリフクラスタ

テキストレンダリングにおいては，`char32`型の 1 要素で表現される Unicode のコードポイントと，グリフの視覚的表現が必ずしも 1 対 1 で対応しないことに留意する．例えば「OK のジェスチャーをする男性」の絵文字は 4 つのコードポイント (U+1F646, U+200D, U+2642, U+FE0F) で構成される．構成要素を見ると，U+1F646 は「OK のジェスチャーをする人物」の絵文字であり，U+200D は ZWJ(Zero Width Joiner), U+2642 は「男性のサイン」の絵文字，U+FE0F は異体字セレクタである．フォントによっては「OK のジェスチャーをする男性」を収録せず，それらの構成絵文字のみを収録している場合があり，そのときは「OK のジェスチャーをする人物」と「男性のサイン」の絵文字の 2 つの表示にフォールバックすることが望ましい．つまり，グリフの個数はフォントに依存する．

Siv3D では，フォントを表現するクラス `Font`に定義された関数呼び出し演算子にテキストを渡すことで，次のようなメンバ変数で構成される `DrawableText`オブジェクトを得る．

```
// 視覚的な1 文字に対応するグリフ
struct GlyphCluster
{
  // フォント内で特定のグリフを参照するためのインデックス
  GlyphIndex glyphIndex = 0;

  // フォントのインデックス（フォールバックフォントを指す場合，以上になる）1
  uint32 fontIndex = 0;

  // テキストを構成する一連のコードポイント配列におけるインデックス
  size_t pos = 0;
};

struct DrawableText
{
  // フォント．追加のフォールバックフォントを持つ場合もある
  Font font;

  // テキスト
  String text;

  // グリフの配列
```

```
  Array<GlyphCluster> clusters;
};
```

そして，DrawableTextの持つメンバ関数.draw()により，一連のテキストを画面に描画する．

```
// フォントを作成
const Font font{ 20 };

// テキストを描画
font(U"Hello, world!").draw(20, 20);

// 一般的なフレームワークで使われるAPI
// Siv3D のclusters に相当するデータにプログラマがアクセスできない
//font.draw(U"Hello, world!", 20, 20);
```

Siv3D ではこのような設計により，文字単位で描画色や位置をカスタマイズする場合，DrawableTextクラスのメンバ変数 clustersを通して個別のグリフ情報を取得できる．一般的な描画フレームワークでは隠蔽される中間データを，他の図形やテクスチャと一貫性のある自然な記述で取得できるところが利点である．

3.1.2 スクリプト

Siv3D では，C++ に構文の似ている AngelScript をカスタムした専用のスクリプト言語をバインドし，Siv3D の主要な API をスクリプトプログラムから呼び出し可能にしている．詳細を 3.4 で述べる．次のコードは，スクリプトプログラムで記述された関数をロードし，C++ プログラムの値を引数としてその関数を呼び出すサンプルである．

```
# include <Siv3D.hpp>

void Main()
{
  // スクリプトファイルのオープン
  const Script script{ U"test.as" };

  // 関数のロード．
  // 戻り値や引数の型はC++ コードで静的に記述
  auto Add = script.getFunction<int32(int32, int32)>(U"Add");

  // スクリプト関数の呼び出し
  Print << Add(100, 23); // 123
}
```

Siv3D では，次のように，アプリケーションのロジックをすべてスクリプトに記述し，C++ はそのプログラムを実行するだけという設計も可能である．この場合，アプリケーションの開発期間を通して，C++ コードのビルドは 1 回のみで済む．

```
# include <Siv3D.hpp>

void Main()
```

```
{
  const ManagedScript script{ U"hello.as" };

  while (System::Update())
  {
    // すべての処理をスクリプトに任せる
    script.run();
  }
}
```

3.1.3 描画 / 視覚要素の処理

Siv3D は 2D/3D 図形の様々なクラスを提供する．それぞれのクラスには，データ表現，描画や幾何処理のためのメンバが実装されている．例えば，いずれの 2D 図形も．その図形自身を画面に描画する.draw()や，その図形が左クリックされているかを返す.leftClicked(), その図形と別の図形が交差しているかを計算する.intersects()など，共通するメンバ関数を持つ．

```
# include <Siv3D.hpp>

void Main()
{
  // 長方形を作成
  const Rect rect{ 50, 50, 100, 200 };

  while (System::Update())
  {
    // 長方形を描画
    rect.draw();

    // 円を描画
    Circle{ 200, 200, 100 }.draw(ColorF{ 0.8, 0.9, 1.0 });

    // 線分を描画
    Line{ 100, 50, 300, 200 }.draw(Palette::Gray);

    // 長方形が左クリックされたら
    if (rect.leftClicked())
    {
      // ...
    }
  }
}
```

Siv3D が提供する，2D 図形に関連するクラスの一覧を表 5 に示す．ほとんどのプリミティブなクラスは，形状を表現するための最小限のデータから構成され，描画やインタラクションのためだけでなく，アルゴリズムの記述でも利用しやすいことが特徴である．

表 5　Siv3D の 2D 図形クラス

クラス	構成データ	説明
Point	int32 * 2	二次元ベクトル（要素が整数）
Float2	float * 2	二次元ベクトル（要素が float）
Vec2	double * 2	二次元ベクトル（要素が double）
Line	Vec2 * 2	線分
Circle	Vec2 + double	円
Rect	Point * 2	長方形（要素が整数）
RectF	Vec2 * 2	長方形（要素が double）
Triangle	Vec2 * 3	三角形
Quad	Vec2 * 4	凸である四角形
RoundRect	RectF + double	角丸長方形
Ellipse	Vec2 * 2	楕円
Polygon	Array<Vec2> + 管理情報	多角形
MultiPolygon	Array<Array<Vec2>> + 管理情報	多角形の集合
LineString	Array<Vec2>	線分の連続
Spline2D	(実装参照)	スプライン曲線
Bezier2	Vec2 * 3	二次ベジェ曲線
Bezier3	Vec2 * 4	三次ベジェ曲線

画像処理に関しては，C++ プログラムからアクセスできるメインメモリ上にデータを確保する画像クラス Imageと，画面への描画のために GPU メモリ上にデータを確保する画像 (テクスチャ) 管理クラス Textureが代表的なクラスである．後者はテクスチャ自身を画面に描画する.draw()や，画像を変形・加工した情報をまとめたオブジェクトを返すようなメンバ関数を提供する．次のように一連の操作をメソッドチェーンで表現できるため，複雑な操作であっても記述がコンパクトになる．画像ファイルは標準で 10 種類のコーデックに対応するが，利用者がプラグインで追加することもできる．

```
1  # include <Siv3D.hpp>
2
3  void Main()
4  {
5    // 画像ファイル"tedt.png" から画像データをロードする
6    const Texture texture{ U"test.png" };
7
8    while (System::Update())
9    {
10     // 画面の(100, 100) の位置に描画
11     texture.draw(100, 100);
12
13     // 0.5 倍に拡大して左右反転して(200, 200) の位置に描画
14     texture.scaled(0.5).mirrored().draw(200, 200);
```

```
15
16      // 一部を切り出して，1.5 倍に拡大して(300，300) の位置に描画
17      texture(100，100，40，40).scaled(1.5).draw(300，300);
18    }
19  }
```

テキスト描画に関しては, Fontクラスが, フォントファイルや, そこから読み取った各種グリフのデータを管理する．アウトライン情報をもとにビットマップデータにラスタライズしたグリフはテクスチャとしてキャッシュし再利用できるようにする．テキストのレンダリング方式として，SDF (Signed Distance Field)[31] および MSDF (Multi-channel Signed Distance Field)[11] を採用し，拡大しても文字のシャープさが保たれ，輪郭や影などのエフェクトを追加の描画パス無しで付加できる柔軟なレンダラを構築した．

3.1.4 物理演算

物理演算処理を行うサードパーティライブラリのラッパーである．Siv3D の図形型や座標系をそのまま扱えるようにしたり，Siv3D のカメラシステムと連動する仕組みを実現したりするなど，データ表現を内部で変換することで，外部の 2D 物理エンジンと Siv3D との親和性を高める役割を持つ．

3.1.5 HID (Human Interface Device)

マウスカーソルの座標やキーボードの状態，Web カメラやマイク等から入力されるデータ等を管理する．Web カメラなどの周辺デバイスは，利用者のプログラムからの要求が無ければデータ更新の処理をスキップし，実行時の負荷を抑える．

3.1.6 オーディオ処理

音声波形データの保持や再生管理，サウンドフォントを用いた MIDI のレンダリングなどを行う．再生中にリアルタイムで適用するフィルタ処理や FFT などの信号解析の機能を提供する．様々な音声コーデックのエンコードとデコードに標準で対応する．画像コーデック同様，ユーザが追加のコーデックをプラグインとして追加できる．

```
1  # include <Siv3D.hpp>
2
3  void Main()
4  {
5    // オーディオファイル"test.mp3" を，ストリーミング方式で読み込む
6    const Audio audio{ Audio::Stream, U"test.mp3" };
7
8    while (System::Update())
9    {
10     // Play ボタンを押したら
11     if (SimpleGUI::Button(U"Play", Vec2{ 200, 20 }))
12     {
13       // 再生・再開
14       audio.play();
15     }
16
```

```
17      // Pause ボタンを押したら
18      if (SimpleGUI::Button(U"Pause", Vec2{ 200, 60 }))
19      {
20        // 一時停止
21        audio.pause();
22      }
23
24      // Stop ボタンを押したら
25      if (SimpleGUI::Button(U"Stop", Vec2{ 200, 100 }))
26      {
27        // 停止して再生位置を最初に戻す
28        audio.stop();
29      }
30    }
31  }
```

3.1.7　通信・ネットワーク

コンピュータ上にある他プロセスや，他のコンピュータと通信するための機能を提供する．現実的なアプリケーションやシステムは，複数のプログラムやコンピュータの協調によって動作するものがほとんどであり，通信に関する API の整備はフレームワークの応用範囲を広げるうえで不可欠である．Siv3D における事例として，図 21 は，Siv3D を利用したペットロボットのシステム構成図である．Siv3D でコーディングされたコントロールプログラム（図の左中央）と同一コンピュータ上で実行される音源定位ソフトウェア（図左上）との TCP 通信，命令をロボット本体に無線送信するモジュール（図左下）とのシリアル通信は，Siv3D の API を利用して実装された．コントロールプログラムは 280 行というコンパクトな規模になり，開発のコストが低減された [100].

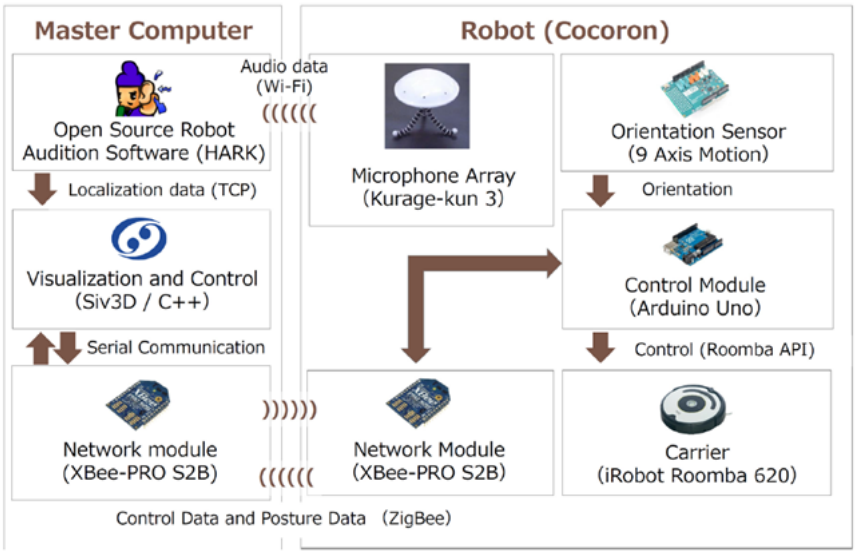

図 21　Siv3D を利用したペットロボット「ココロン」のシステム構成図（図版は [100] より引用）

3.1.8 アプリケーション開発のための高レイヤユーティリティ

コンピュータゲームやアプリケーションの開発生産性を向上させる機能を，Siv3D のパブリック API を用いて実装する，高レイヤのライブラリ群である．具体的には，ボタンやスライダーなどの GUI，画面の切り替え時のアニメーション表現などをサポートするライブラリである．

3.1.9 サードパーティ・ソフトウェア

フレームワークのすべての機能を自前で開発，メンテナンスすることは，必要な知識と労力の問題から現実的ではない．C/C++ は歴史が長く，多くのライブラリ資産がインターネット上に存在するため，自身以外の個人や集団，企業が提供するオープンソースのサードパーティ・ソフトウェアを採用することで開発を効率化できる．これによって，本来フレームワークが実現するコア機能の開発に集中できる．

Siv3D では，OS やハードウェアと密接に連係する処理や，物理演算，データ圧縮，画像・音声コーデックなど，自前での実装が難しい低レイヤの複雑な処理に関しては，デファクト・スタンダードであるか，多くの商用ソフトウェアでの使用実績を有する，十分に検証されているサードパーティ SDK やライブラリを利用し開発の労力を削減した．ソフトウェアの選定においては，Siv3D で開発したアプリケーションをユーザが商用利用するうえで支障にならないライセンスを選択した．おもに次のようなライセンスのソフトウェアが採用されている．

- Creative Commons Zero v1.0 Universal
- Unlicense（パブリックドメイン）
- Boost Software License
- zlib License
- Apache License 2.0
- MIT License
- BSD License
- GNU Lesser General Public License v2.1

これらのうち，GNU Lesser General Public License のライブラリを最終的なアプリケーションの実行ファイルと静的リンクすると，そのアプリケーションのオブジェクトコードまたはソースコードを配布する義務が利用者に生じるため，商用ソフトウェアを配布したい利用者には都合が悪い．そのようなライセンスのライブラリは実行ファイルとは分離して動的リンクする構成をとる．また，Siv3D ではプラットフォーム間での画面のビジュアルの差異を無くすために，共通のフォントや絵文字を同梱しており，それらもオープンソースライセンスのものが使われる．Siv3D で使用されているサードパーティ・ソフトウェアと，そのライセンス（SPDX short identifier による表記）を付録 A にまとめた．

3.2 デジタルコンテンツ制作を支援するためのアセットの提供

リッチなインタラクティブアプリケーションの開発においては，図形やテキストデータ以外にも，アイコンや画像，効果音など，様々な画像・音声素材データが必要になる．主要な商用ゲームエンジンは，アプリケーションで使える画像や 3D モデル，シェーダ，プラグインなど，一般ユーザーによって制作されたコンテンツのマーケットプレイスの仕組みを運営し，利用者がそれらを購入することで素材コンテンツを制作する労力を

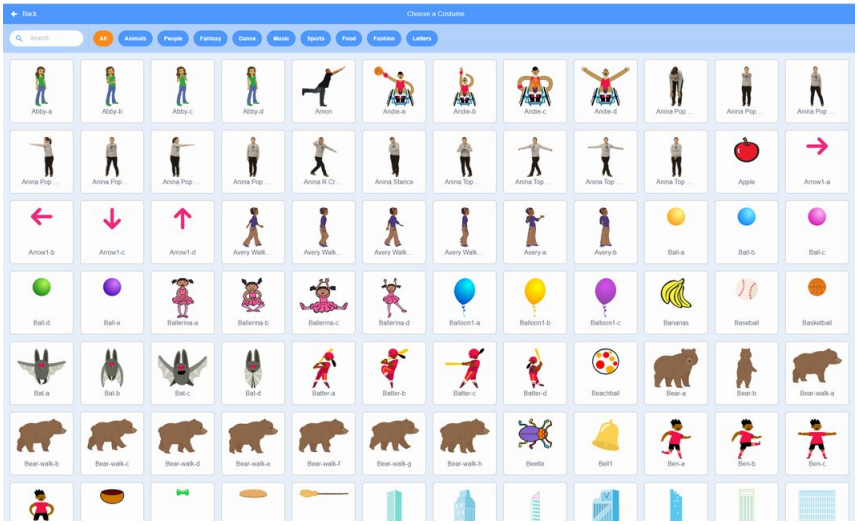

図 22　Scratch が標準で提供する画像素材集，Costume Library

減らす仕組みを提供している．また，マーケットの仕組みはないが，ビジュアルプログラミング言語 Scratch のように，標準のエディタから数百種類の画像・音声素材ライブラリ（図 22）にアクセスでき，利用者が簡単な操作で自身のプロジェクトにそれらの素材をインポートできるような仕組みを提供するツールも多い．

プログラミングツールを新規に開発する際，こうした画像・音声素材のライブラリをゼロから構築するのは多大な時間と経済的コストを要するため，新しいプログラミングツールが既存のツールに対して競争力を持つうえで，参入障壁の一つとなっていた．Siv3D では，独自に素材を開発する代わりに，オープンソースで提供される絵文字フォント，アイコンフォント，サウンドフォントに着目し，それらをフレームワークに統合することで，一定の量と品質を持つ素材ライブラリを非常に少ない労力で構築した．

絵文字フォントとは，Unicode Emoji で定義される絵文字を，カラー画像で収録したフォントファイルであり，次のような理由からデジタルコンテンツ制作の素材として利便性が高い．

- 最新の Unicode Emoji 13.1 規格では約 3,000 種類以上の絵文字が定義されている
- ジャンルは人，表情，動物，料理，植物，建物，道具など多岐にわたり，様々な用途に使える
- 人を表す絵文字は 3 種類のジェンダーと 6 種類のスキントーンの合計 18 種類のパターンを持ち，多様性に配慮されている
- 同一のフォントに収録されている絵文字はデザインスタイルが一貫していて使い勝手が良い
- 低解像度でも視認しやすいようデザインされている
- ベクター画像データを入手できる

Microsoft や Apple などのベンダーが OS で提供する絵文字フォントファイルはプロプライエタリであるが，Google の Noto Emoji fonts (`https://github.com/googlefonts/noto-emoji`) は SIL Open Font License Version 1.1, Twitter の Twemoji (`https://github.com/twitter/twemoji`) は CC-BY 4.0 で提供されている．

アイコンフォントとは，アプリケーションのグラフィカルユーザインタフェース上に表示するシンボルマークとして使用できる，通常は単色のアイコンのコレクションである．アイコンフォントには，Unicode Emoji には採用されていないが，デスクトップやモバイルアプリケーション，Web サービス等で慣習的に使用される

アイコンを，Unicode の私用領域 (Private Use Area) に収録してバリエーションを増やしているものが多い．

Siv3D の API を通して，絵文字やアイコンのアセットデータを利用するには，Texture, Image, Fontクラスを用いる．次のコードは任意の絵文字をテクスチャとして画面に描画するサンプルである．

```
const Texture texture{ U"🍣"_emoji }; // 使用したい絵文字1 字を記述する

// テクスチャを描画
texture.drawAt(100, 100);
```

4.2 で述べるユーザ定義リテラル_emojiにより，コンストラクタ引数は画像ファイルのパスと区別される．文字列のコードポイントに対応する絵文字のグリフをフォントファイル内で検索し，画像をロードする．ソースコード中に絵文字が登場するのはユニークであるが，直感的で読み書きしやすい．

次のコードは任意のアイコンをテクスチャとして画面に描画するサンプルである．

```
const Texture texture{ 0xF0493_icon, 128 }; // 使用したいアイコンとそのサイズ

// アイコンを描画
texture.drawAt(100, 100);
```

ユーザ定義リテラル_iconにより，コンストラクタ引数の整数は，アイコンフォントの Unicode の私用領域にマッピングされたコードポイントとみなされる．対応するグリフをフォントファイル内で検索し，画像をロードする．

次のコードはアイコンをテキスト内に含めて描画するサンプルである．Unicode の私用領域のコードポイントは通常のテキストエディタで表現できないため，コード中ではエスケープして記述する．

```
// 通常テキストのフォント
const Font font{ 20 };

// アイコンフォント
const Font iconFont{ 20, Typeface::Icon_MaterialDesign };
font.addFallback(iconFont);

// アイコンを含んだテキストを描画(Siv3D においてF0493 は歯車のアイコン)
font(U"\U000F0493 設定").draw(20, 20);
```

Siv3D ではこうした方法を用いることで，特別なデータ構造やデータベースを用意することなく，既存の描画 API と親和性の高い構文で，絵文字やアイコンの画像アセットをアプリケーションで活用できるようにした．

Siv3D アプリケーションの中で，絵文字による画像アセットを効果的に活用した事例が，ビジュアルプログラミング言語「Siv3D for Kids」[89][93] である．Siv3D for Kids は，プログラム内に登場する，キャラクターや物体の速さや大きさ，色，画像をタッチ操作の GUI を通して書き換えることで，プログラムの挙動をカスタマイズし，結果の変化を楽しむという，コンピュータの初学者向けの教育的なプログラミング体験ソフトウェアである．Siv3D for Kids の開発にあたっては，本物のゲームのようなビジュアル品質と楽しいインタラクション，優れたプログラミング体験のための柔軟なカスタマイズ性を要件としたため，Siv3D が提供していた絵文字アセットが実装の助けとなった．絵文字の画像データに対しては，Siv3D の画像処理機能を用い

て，影や輪郭を付ける視覚効果の付与や，絵文字の輪郭形状を抽出し，それに基づいた 2D 物理演算をゲームに取り入れるなど，インタラクションのバリエーションが豊富でエンタテインメント性のある表現を実現した [97]（図 23，図 24，図 25）．

図 23　Siv3D for Kids のスクリーンショット．ゲームに登場させる絵文字を GUI で選択する

図 24　Siv3D for Kids のスクリーンショット．絵文字に回転や拡大などを適用する

図 25　Siv3D for Kids のスクリーンショット．絵文字に物理演算を適用する

3.3　ロードに失敗したアセットの扱い方

利用者がアプリケーションの開発中にしばしば遭遇するエラーとして代表的なものが，画像やオーディオファイルなどアセットのロードの失敗である．プログラムに記述したファイルパスか誤っていることもあれば，読み込み対象のファイルの破損や移動・削除が原因ということもある．ファイルのロードに失敗した際のフレームワークの挙動として，エラーや例外を発生させる，画像の場合はまったく何も表示しない，音声の場合は無音にするなどの選択肢があるが，Siv3D ではアプリケーションをエラーなどで中断させないことを優先し，Llopis の手法 [45] を参考に，「ヌルアセット」と呼ばれる特殊な画像やオーディオにフォールバックさせる方式を採用した．

「ヌルアセット」は，未初期化であるか，ロードに失敗したテクスチャやオーディオのハンドルがデフォルトで指す，プレースホルダの役割を持つ画像やオーディオデータである．Siv3D ではアセットの種類に応じて「ヌルテクスチャ」「ヌルオーディオ」のように呼称する．Siv3D のヌルテクスチャの実データは黄色で塗りつぶされた小さなサイズの画像であり，ユーザがヌルテクスチャを使用すると，画面に黄色で塗りつぶされた四角形や，黄色の 3D モデルが登場する．Siv3D のヌルオーディオは，0.5 秒間の長さのフェードイン・フェードアウトするチャープ信号（周波数が時間の経過とともに高くなる信号）の音声波形（図 26）である．ヌルオーディオは，当初の Siv3D の仕様では 440Hz 固定周波数の正弦波であったが，ゲームにおける破壊の効果音のように，短時間に何度も重なり合う効果音がその波形にフォールバックすると，再生に失敗したタイミングや回数の聴き分けが困難であることから，チャープ信号に置き換えて改善を図った．

Siv3D におけるヌルオーディオ用のチャープ信号の生成コードは次の通りである．

```
1 Wave::Generate(0.5s, [](double t)
2   { return 0.5 * std::sin(t * Math::TwoPi) // チャープ信号
3     * std::sin(t * Math::TwoPi * 220.0 * (t * 4.0 + 1.0)); // フェードイン・アウト
4   });
```

ヌルテクスチャとヌルオーディオは，どちらも大きさや長さを持つ有効なアセットデータであり，アプリ

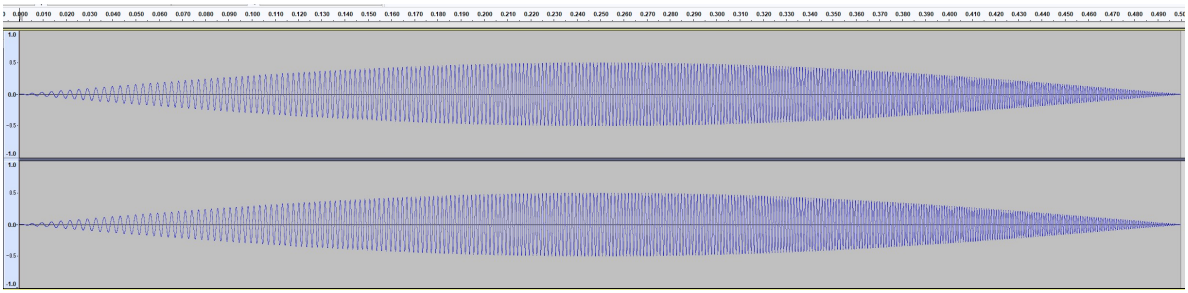

図 26　Siv3D のヌルオーディオに使われる，フェードイン・フェードアウトするチャープ信号の波形．異なるタイミングで複数回再生されても一つひとつを聴き分けられる

ケーションプログラムがそれらを参照しても，通常のアセットと同じように動作し続け，データが存在しないことよる意図せぬ中断やクラッシュ等の影響を回避できる．開発者は視覚と聴覚によって問題箇所を発見しやすくなり，プログラムの修正をスムーズに行えるようになる．

ゲームエンジン Unity においても，レンダリングにエラーが発生している視覚オブジェクトについて，マゼンタ色 (RGB=(1.0, 0.0, 1.0)) で塗りつぶされるという，似たようなフォールバックの仕組みがある．マゼンタ色は黄色と同様，自然物に少ない色であるためゲームシーン中でも目立つが，Siv3D の採用する黄色 (RGB=(1.0, 1.0, 0.0)) が優れる点は，色覚特性の中でも割合が多い P 型，D 型色覚（図 27）における影響がマゼンタよりも小さいことである．図 28 の左の画像は，3D ゲームのシーンとして想定される，空や地面に近い色を背景に設定し，左側にマゼンタ色の図形を，右側に黄色の図形を配置した画像である．この画像に対し，Adobe Photoshop の色覚シミュレータを用いて，P 型色覚特性を再現したものが中央画像，D 型色覚特性を再現したものが右画像である．P 型においてマゼンタは青系統の色と知覚され，D 型においては空の色に溶け込む現象もみられる．対照的に，黄色はいずれの色覚においても一貫して黄色として識別できるため，エラーを通知する色として適している．

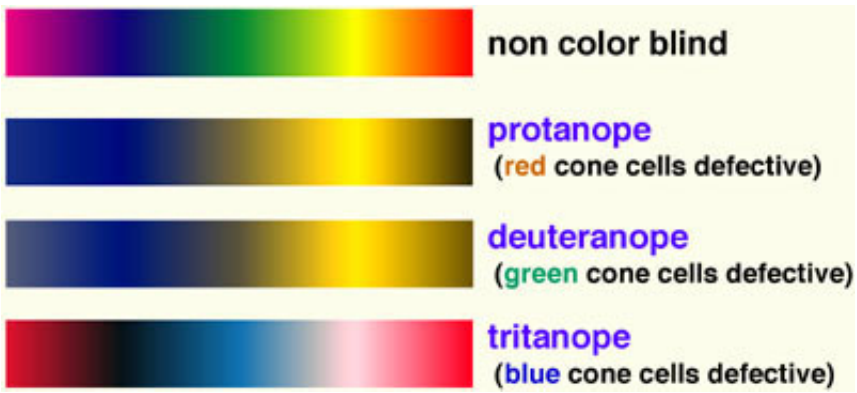

図 27　色見本と色覚特性シミュレーション．上から二段目が P 型色覚，三段目が D 型色覚（図版は [58] より引用）

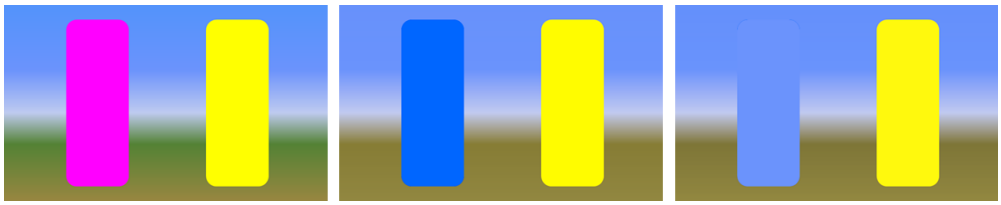

図 28　3D ゲームのシーンを想定した背景に，マゼンタと黄色の図形を配置した画像．左から順にオリジナル画像，P 型色覚特性のシミュレーション結果，D 型色覚特性のシミュレーション結果

3.4　スクリプト言語による開発イテレーションの高速化

C++ コードのビルド（コンパイルとリンク）に要する時間は，他のプログラミング言語と比較しても特に大きい．長いビルド時間は開発者がイテレーションやテストに使える時間を奪い，プログラミング生産性を低下させる原因となっていた．一般的な対策として，ソースコード中の変更頻度が多い部分については，コンパイルが高速なスクリプト言語で開発を行うテクニックが使われる．

Siv3D ではフレームワーク標準のスクリプト言語として AngelScript[35] を採用し，AngleScript コンパイラと，Siv3D の C++ API の AngelScript へのバインド，AngelScript のソースコードをプログラムで扱いやすくするための各種 API をフレームワークの機能として提供した [94]．バインドの実装は，Golodetz の手法 [26] をベースに，C++11 の Variadic templates を用いて引数の個数の上限を無くす改良を加えたものである．完全なコードを Siv3D のコードリポジトリから入手できる．

C++ と併用される代表的なスクリプト言語には Lua[47] があるが，Lua の構文は C++ と大きく異なり，利用者は追加で Lua プログラミングの習得が必要になる．一方，AngelScript は C++ に構文を似せて設計されたプログラミング言語であり，実際にほとんどの基本的な構文が共通する．AngelScript を採用することで，利用者が C++ とスクリプトプログラムのそれぞれの記述に対して，同じスタイルでコードを書ける利点がある．C++ と AngelScript の比較のために，Siv3D のサンプルプログラムを，それぞれの言語で記述した例を次に示す．

```
1  // C++ コード
2  # include <Siv3D.hpp>
3
4  void Main()
5  {
6          Scene::SetBackground(ColorF(0.8, 0.9, 1.0));
7          const Font font(60);
8          const Font emojiFont(60, Typeface::ColorEmoji);
9          font.addFallback(emojiFont);
10         const Texture texture(U"example/windmill.png");
11         const Texture emoji(U"🎏"_emoji);
12         Vec2 emojiPos(300, 150);
13         Print << U"Push [A] key";
14
15         while (System::Update())
16         {
```

```
                texture.draw(200, 200);
                font(U"Hello, Siv3D!").drawAt(Scene::Center(), Palette::Black);
                emoji.resized(100 + Periodic::Sine0_1(1s) * 20).drawAt(emojiPos);
                Circle(Cursor::Pos(), 40).draw(ColorF(1, 0, 0, 0.5));

                if (KeyA.down())
                {
                        Print << U"Hello!";
                }

                if (SimpleGUI::Button(U"Button", Vec2(640, 40)))
                {
                        emojiPos = RandomVec2(Scene::Rect());
                }
        }
}
```

```
// AngleScript コード

void Main()
{
        Scene::SetBackground(ColorF(0.8, 0.9, 1.0));
        const Font font(60);
        const Font emojiFont(60, Typeface::ColorEmoji);
        font.addFallback(emojiFont);
        const Texture texture("example/windmill.png");
        const Texture emoji(_emoji("🎏"));
        Vec2 emojiPos(300, 150);
        Print << "Push [A] key";

        while (System::Update())
        {
                texture.draw(200, 200);
                font("Hello, Siv3D!").drawAt(Scene::Center(), Palette::Black);
                emoji.resized(100 + Periodic::Sine0_1(_s(1)) * 20).drawAt(emojiPos);
                Circle(Cursor::Pos(), 40).draw(ColorF(1, 0, 0, 0.5));

                if (KeyA.down())
                {
                        Print << "Hello!";
                }

                if (SimpleGUI::Button("Button", Vec2(640, 40)))
                {
                        emojiPos = RandomVec2(Scene::Rect());
                }
```

```
62         }
63 }
```

サンプルコードにおける両者の違いは以下に示した箇所だけであり，それ以外のすべての部分でコードが共通している．

- C++ ではインクルードディレクティブが必要である
- Siv3D の C++ プログラムでは，標準的な文字列リテラルのプレフィックスとして `U` を使うが，AngelScript はそれに対応しない
- AngleScript ではユーザ定義リテラルが使えないため，`_emoji`や`_s`という関数に置き換えている

なお，前述のサンプルコードに含まれない重要な差異として，AngelScript には range-based for が存在しないこと，ユーザ定義型の typedef が存在しない．これらのことは C++ の Siv3D コードとより高い互換性を実現するうえで障壁となっているが，いずれもインデックスベースの for の使用と，エイリアス元の型名を使用することで回避できるため致命的ではない．

AngleScript で記述された Siv3D コードは，通常の C++ コードよりも数十〜数百分の一の時間でコンパイルされる [94][96]．実際の体感として，スクリプトコードに対する変更を行うと，実行中のアプリケーションにインタラクティブな速度で反映される．この速度を生かして，Siv3D では，試行錯誤や頻繁な調整が必要な段階ではスクリプトでコードを記述し，設計が安定してから，わずかな修正を施して C++ コードとに移植するというパターンで開発を行うことができる．これにより，開発におけるイテレーションとテストの高速化と，最終的なアプリケーションの実行時性能の維持を両立した．また，C++ に類似したスクリプト言語を採用することで，利用者が全く新しい体系のプログラミング言語を学習するために要するコストを省略した．

3.5 フレームワークの更新

3.5.1 バージョン番号

フレームワークのリリースにおいては，以前にリリースされたバージョンとの区別のために，リリースに一意のバージョン名を付ける．一般的なソフトウェア開発で用いられるのは，メジャー，マイナー，パッチ（またはリビジョン）の 3 つのバージョン番号による Semantic Versioning と，年や月などの日付を使った Calendar Versioning である．Siv3D は 2016 年まで Calendar Versioning（例: Siv3D August 2016）を採用していたが，次のような理由から，2016 年のオープンソース化を機に Semantic Versioning に移行した．

- 前バージョンとの互換性や機能の変化の度合いに関する情報が含まれない
- 次期バージョンのリリースの日付が確定するまで，文書や広報で次期バージョン名を使えない
- 内部 API で Semantic Versioning を使用していたため機能が重複する

2021 年時点で，Siv3D は正式リリース前であるため，メジャーバージョンは 0 に固定され，マイナーバージョンをメジャーバージョン相当，パッチバージョンをマイナーバージョン相当として運用している．つまり，0.3.0 から 0.4.0 へはメジャーアップデートであり，互換性が失われる変更が導入される可能性がある．0.4.0 から 0.4.1 へはマイナーアップデートであり，互換性が失われるような変更は通常行われない．

Siv3D において，バージョンに関連する API は次のようなマクロが用意される．

```
1 #pragma once
2
```

```
3  /// @biref ライブラリのメジャーバージョン| Library major version
4  #define SIV3D_VERSION_MAJOR   0
5
6  /// @biref ライブラリのマイナーバージョン| Library minor version
7  #define SIV3D_VERSION_MINOR   6
8
9  /// @biref ライブラリのリビジョンバージョン| Library revision version
10 #define SIV3D_VERSION_REVISION   3
11
12 /// @biref プレリリースタグ| Pre-release tag
13 #define SIV3D_VERSION_PRERELEASE_TAG U""
14
15 /// @biref ライブラリのバージョン| Library version
16 #define SIV3D_VERSION   ((SIV3D_VERSION_MAJOR * 100 * 100) + (SIV3D_VERSION_MINOR * 100) +
       (SIV3D_VERSION_REVISION))
17
18 /// @biref ライブラリのバージョンの文字列（簡易版）| Short Library version
19 #define SIV3D_VERSION_SHORT_STRING U"0.6.3"
20
21 /// @biref ライブラリのバージョンの文字列| Library version
22 #define SIV3D_VERSION_STRING U"0.6.3" SIV3D_VERSION_PRERELEASE_TAG
```

C++ では定数にマクロを使うべきではないが，次のように，利用者がプリプロセッサにおいて使用する場合に都合が良いため，例外的に用いられる．

```
1  #define MAJOR_VERSION 0
2
3  // これは正しく機能しない
4  //constexpr int MAJOR_VERSION = 0;
5
6  int main()
7  {
8  #if (0 < MAJOR_VERSION)
9    Version1Function();
10 #else
11   Version0Function();
12 #endif
13 }
```

なお，このコードを C++17 の constexpr ifで置き換える場合，プログラム中のユーザ関数 Version1Functionと Version0Functionの両方が利用可能である必要があり，プリプロセッサの完全な代替とならないことに注意する．

```
1  void Version0Function() {}
2  //void Version1Function() {}
3
4  constexpr int MAJOR_VERSION = 0;
```

```
5
6  int main()
7  {
8    if constexpr (MAJOR_VERSION < 0)
9      Version1Function();
10   else
11     Version0Function();
12 }
13
14 /*
15 prog.cc: In function 'int main()':
16 prog.cc:9:5: error: 'Version1Function' was not declared in this scope
17      Version1Function();
18      ^~~~~~~~~~~~~~~~
19 */
```

3.5.2 リリースノート

新しいバージョンのリリースに際して，変更点などを利用者に通知する文書がリリースノートである．Siv3D のリリースノートには以下の項目が箇条書きで記述される．

新機能 新しく追加された機能とそのサンプルコード
パフォーマンス向上 処理の高速化が行われた機能
ユーザビリティ向上 インストーラやドキュメントの改善
仕様変更 互換性が失われる機能
不具合・バグ修正 修正された不具合
注意点 将来の変更を予定している機能や，使い方に注意すべき変更が生じた API の説明
コントリビューション コミッタによる貢献（名前とその内容）

リリースノートには，コミュニティに関わる要素として，そのバージョンに貢献のあったコミッタの名前も合わせて記載する．貢献者の記載は，C++ フレームワークでは OpenCV をはじめとする多くのフレームワークでも見られる慣習である．Siv3D で同様の施策を行う理由は，コミットへのモチベーションを向上させる効果と，コミットを積極的に受け入れていることを将来のコミッタ候補である利用者に周知する効果をねらったものである．

3.5.3 API の破壊的変更

関数やクラスのインタフェースの変遷の過程で，古いインタフェースを削除することがある．C++14 からは，指定した関数やクラスの使用の非推奨を伝える `[[deprecated]]`属性が言語機能に追加された．Siv3D では非推奨の API の代替となる新しい API や，その移行方法を `[[deprecated]]`属性の引数として記述することで，利用者がコードを更新する助けとなる情報を提供する．

3.5.4 フレームワークの更新に関する Siv3D の失敗事例

失敗事例 1: 引数の順序の変更

Siv3D のあるバージョン更新において，関数の引数の順序を入れ替える次のような変更を行った．変更理由は，利用者が引数 `deltaTime`よりも `maxSpeed`を，デフォルト引数の値からカスタマイズするケースが多いため，なるべく多くのユースケースで，引数の記述個数を減らすためであった．

```
// 変更前
/// @brief 目標地点に向かってスムーズに移動させます．
/// @param from 現在地
/// @param to 目標地点
/// @param velocity 現在の速度
/// @param smoothTime 平滑化時間
/// @param deltaTime 前回からの経過時間．デフォルトではScene::DeltaTime()
/// @param maxSpeed 最大速度．無制限の場合はunspecified
/// @return 新しい現在地
[[nodiscard]]
Vec2 SmoothDamp(const Vec2& from, const Vec2& to, Vec2& velocity, double smoothTime,
  double deltaTime, const Optional<double>& maxSpeed = unspecified);

// 変更後
/// @brief 目標地点に向かってスムーズに移動させます．
/// @param from 現在地
/// @param to 目標地点
/// @param velocity 現在の速度
/// @param smoothTime 平滑化時間
/// @param maxSpeed 最大速度．無制限の場合はunspecified
/// @param deltaTime 前回からの経過時間．デフォルトではScene::DeltaTime()
/// @return 新しい現在地
[[nodiscard]]
Vec2 SmoothDamp(const Vec2& from, const Vec2& to, Vec2& velocity, double smoothTime,
  const Optional<double>& maxSpeed = unspecified, double deltaTime = Scene::DeltaTime());
```

しかし，`double`型の値は `const Optional<double>&`型にも暗黙変換が可能なため，API 変更後でも API 変更前に書かれたコードがそのままコンパイルできてしまう問題を引き起こし，バージョンアップ後にこの変更を見落とした利用者から，アニメーションの挙動の異常が報告されるトラブルがあった．暗黙に変換可能である異なる型の引数の順序の入れ替えという API の変更を，ソースコードでユーザに確実に伝える方法は無いため，この変更は導入されるべきではなかった．

失敗事例 2: Web 上に異なるバージョンのサンプルコードが混在

最新のバージョンと互換性のない，過去のバージョンを対象に書かれたサンプルコードが利用者によって参照され，その利用者からプログラムのビルドが通らないという問題報告を受ける事例が年間数件発生した．Siv3D の公式リファレンス内であれば，ページタイトルにバージョンが明記されているため，バージョンを取り違えることはほとんど発生しないが，第三者によってブログやコード共有サイトに投稿されたプログラムに

図 29　利用者が Web 上に投稿した Siv3D プログラムにバージョンが記載されている様子

ついては，読者が対象バージョンを知ることができず，混乱をきたす原因となっていた．

この問題への対策として，Siv3D のデフォルトプロジェクト（プロジェクト作成時にデフォルトで用意されるサンプルプログラム）のインクルードディレクティブの行に，コメントでバージョン番号を追記するという小さな変更を行った．

```
1 # include <Siv3D.hpp> // OpenSiv3D v0.6.3
2
3 void Main()
4 {
5   ...
```

この先頭行はすべての Siv3D プログラムで必要なコードであることから，利用者が意図的に削除しない限り，最後までコードに残り続ける．この変更の導入によって，特にバージョンに関する注意を周知せずとも，利用者によってコード共有サイトに投稿される Siv3D プログラムのほとんどにバージョン番号が含まれるようになった（図 29）．この方法は，API の追加や変更が頻繁に行われるフレームワークにおいて参考になるだろう．

4　C++ API 設計の課題と Siv3D での解決

C++ のライブラリ API の設計については，Reddy が基本的な作法を広く論じている [67]．Gregory は，C++ によるゲームエンジンの構成を解説するなかで C++ API 設計の事例についても述べている [32]．Nystrom は C++ によるゲームプログラミングに有益なデザインパターンを複数取り上げ，現代的なコードによる実装と用例を紹介している [56]．さらに汎用的な C++ のコーディングルールとしては C++ Core Guidelines[85] が充実している．本研究ではこれらの記事で扱っているルールや技法については踏襲したうえで，情報可視化や人と計算機のインタラクションに関する大規模フレームワークを開発する過程で顕在化した，C++ 言語の使い勝手に関する課題を複数指摘し，言語仕様の制約の中でそれらの課題の解決もしくは緩和を図った，Siv3D 独自の工夫に基づく API 設計を説明する．これらのアイデアは，Siv3D の長年にわたるフレームワーク運用を通して不整合などの問題が生じないことを実証している．

4.1　名前付き引数エミュレーションによるコンストラクタオーバーロードを用いた柔軟な図形定義

4.1.1　コンストラクタのオーバーロード

情報可視化のために図形データを扱う C++ ライブラリは，通常，図形をその幾何学的性質に応じていくつかのクラスに分類し，それぞれのクラスのコンストラクタについて，関数オーバーロード (function overloading) によって，目的の図形を最小限の入力から柔軟に定義できる方法を提供する．ここでは例として長方形を表現するクラス `Rect`を考える．

```
// 2D ベクトル
struct Point
{
  int x, y;
};

// 長方形
struct Rect
{
  int x, y, w, h;

  // 左上を (0,0)におく正方形の定義に便利
  explicit Rect(int size);

  // 左上を (0,0)におく長方形の定義に便利
  Rect(int w, int h);

  // 正方形の定義に便利
  Rect(int x, int y, int size);

  // 長方形の定義に便利
  Rect(int x, int y, int w, int h);
```

```

  // Point 型の値から構築
  Rect(Point pos, int size);

  // Point 型の値から構築
  Rect(Point pos, int w, int h);

  // ...
};

int main()
{
  // 左上の座標が(40,20)
  // 一辺が50 の正方形
  Rect rect{ 40, 20, 50 };
}
```

長方形の定義においては，中心座標を基準とするほうが都合が良いケースもあるし，左下や右下を基準にするほうが便利な場合もある．しかし，C++ のオーバーロードの解決は引数の型と個数の対応関係によって行われるため，左上座標を基準とするコンストラクタと，それとは別に中心座標を基準とするコンストラクタは単純には共存できない．このようなとき，オフセットの計算をラップした静的関数を提供するのが 1 つの解決方法であるが，静的関数はコンストラクタではないため，new 式や，一時オブジェクトの作成を介さずコンテナに要素を追加する emplacement では使えない制約が生じ，一貫性に欠く．

```
struct Rect
{
  // 中心座標と幅，高さからRect を作成
  static Rect FromCenter(Point center, int w, int h);

  // ...
};

int main()
{
  Point pos{ 10, 20 };

  std::vector<Rect> rects;

  rects.emplace_back(pos, 100);

  rects.emplace_back(pos, 100, 50);

  // emplacement で構築できない
  rects.push_back(Rect::FromCenter(pos, 100, 50));
}
```

コンストラクタのオーバーロードという文脈でこの問題を解決する方法として，タグ・ディスパッチング (tag dispatching) がある．タグ・ディスパッチングでは，オーバーロードを増やすために，タグとして機能するだけの，メンバを持たないクラスを登場させる．

```
struct TopLeft{};
struct Center{};

struct Rect
{
  Rect(TopLeft, Point topLeft, int w, int h);

  Rect(Center, Point center, int w, int h);

  // ...
};

int main()
{
  Point pos{ 10, 20 };
  Rect a{ TopLeft{}, pos, 30, 40 };
  Rect b{ Center{}, pos, 30, 40 };
}
```

列挙子によって動的に分岐する方法でも，コンパイル時定数に基づく最適化でゼロオーバーヘッドにできる可能性がある．しかし，図形が複数種類ある場合，その性質に合わせた選択肢を提供する列挙型をクラスごとに用意する必要があり，それらの名前の衝突を防ぐために，ユーザコードはタグ・ディスパッチングと比較して文字数が増えることになる．

```
struct Rect
{
  enum class Alignment
  {
    Center,
    TopLeft,
  };

  // Alignment:: を省略するため
  static constexpr auto Center = Alignment::Center;
  static constexpr auto TopLeft = Alignment::TopLeft;

  Rect(Alignment a, Point pos, int w, int h);

  // ...
};

// 円を表現する別のクラス
struct Circle
```

```
{
  enum class Alignment
  {
    Center,
    TopCenter, // TopLeft は無い
  };
};

int main()
{
  Rect a{ Rect::TopLeft, {10, 20}, 30, 40 };

  Rect b{ Rect::Center, {10, 20}, 30, 40 };
}
```

これ以外に，座標クラスとそのコンストラクタを継承した，型情報に基準位置を持つクラスを作ることも考えられる．この方法はコンストラクタに渡す実引数の個数と記述を少なくできる．

```
struct TopLeft : Point
{
  // コンストラクタの継承
  using Point::Point;
};

struct Center : Point
{
  // コンストラクタの継承
  using Point::Point;
};

struct Rect
{
  Rect(TopLeft topLeft, int w, int h);

  Rect(Center center, int w, int h);

  // ...
};

int main()
{
  Rect a{ TopLeft{10,20}, 30, 40 };
  Rect b{ Center{10,20}, 30, 40 };
}
```

しかし，そうしたクラスをジェネリックに扱うためにクラステンプレート化すると，次のようにユーザコードで型を明示的に記述するという追加の手間が生じる．

```
// 要素が整数型の二次元ベクトル
struct Point
{
  int x, y;
  Point() = default;
  Point(int _x, int _y);
};

// 要素が浮動小数点数型の二次元ベクトル
struct PointF
{
  float x, y;
};

template <class PointType>
struct TopLeft : PointType
{
  using PointType::PointType;
};

template <class PointType>
struct Center : PointType
{
  using PointType::PointType;
};

struct Rect
{
  Rect(TopLeft<Point> topLeft, int w, int h);

  Rect(Center<Point> center, int w, int h);

  // ...
};

int main()
{
  Rect a{ TopLeft<Point>{10,20}, 30, 40 };
  Rect b{ Center<Point>{10,20}, 30, 40 };
}
```

Siv3D では，ユーザの記述のコストを抑えつつコンストラクタのバリエーションを増やすという課題の解決法として，呼び出し側がコンストラクタ引数とタグをセットにしたオブジェクトを簡易に構築できる仕組みを考案した [95][96]．その構文の見た目から「名前付き引数」と呼ぶ．当然 C++ のコア言語機能には本物の名前付き引数の仕組みはないため，疑似的に再現したものである．実装法は次の節で述べる．

```
struct Rect
{
  Rect(TopLeft_<Point> topLeft, int w, int h) {}

  Rect(Center_<Point> center, int w, int h) {}

  // ...
};

int main()
{
  Point pos{ 11, 22 };
  Rect a{ TopLeft = pos, 30, 40 };
  Rect b{ Center = pos, 30, 40 };
  Rect c{ TopLeft(10,20), 30, 40 };
  Rect d{ Center(10,20), 30, 40 };
}
```

名前付き引数によるオーバーロードにより，Siv3D の長方形クラスや円クラスのコンストラクタは，様々な位置を基準にして図形を定義できるようになった．線分クラスは，2 点の座標の組からだけでなく「始点と，始点から終点へのベクトル」や「始点と，始点から終点への角度と距離」といった様々なパターンでの構築が可能になった．

```
void Main()
{
  using namespace Arg;
  Point pos1{ 11, 22 }, pos2{ 33, 44 };
  Rect a{ topLeft = pos1, 30 };
  Circle b{ topCenter = pos2, 30 };
  Line c{ pos1, pos2 };
  Line d{ pos1, direction(40, 50) };
}
```

4.1.2 名前付き引数のエミュレーション

本節では Siv3D における名前付き引数エミュレーションの仕組みについて説明する．ある名前付き引数 dayを例にとり，どのような仕組みで動作するのか順を追って述べる．再利用可能な実装コード全体を付録 B に掲載したので，それも参考にされたい．

まず，ユーザによって記述される SIV3D_MAKE_NAMED_PARAMETER(day) マクロが，プリプロセス時に次のコードに置換される．

```
constexpr auto day = s3d::NamedParameterHelper<struct day_tag>{};

template <class Type> using day_ =
  s3d::NamedParameterHelper<struct day_tag>::named_argument_type<Type>;
```

NamedParameterHelper<day_tag>型のオブジェクト dayには operator=と operator()が定義されており，day=31または day(31)のように渡された値を NamedParameter<day_tag, int>(31)とラップして返す．dayはコンパイル時定数であり，代入を行っているように見えるコードの印象に反して，dayの状態は一切変化しない．

```
// result はNamedParameter<day_tag, int> 型
auto result = (day = 31);
```

名前付き引数を用いる関数では，仮引数の型として day_<int>を記述する．これは NamedParameter<day_tag, int>型の typedef であり，代入式で生成された NamedParameterを受け取ることができる．渡されたオブジェクトから実際の値を取り出すには.value()を用いる．

```
void Day(day_<int> d)
{
    std::cout << d.value() << '\n';
}

int main()
{
    auto result = (day = 31);
    Day(result);
}
```

4.1.3 名前付き引数エミュレーションの性能とユーザビリティへの影響

Siv3D の名前付き引数エミュレーションの実装（付録 B）は constexprに対応しており，名前付き引数を使用する前にコンパイル時定数式であったコードは，名前付き引数を導入したあとでもコンパイル時定数式を維持できる．また GCC と Clang を使った調査では，関数がインラインで定義されていれば，コンパイラの最適化によって通常の関数呼び出しと同じ実行コードが生成されることが確認された．付録 B に記載した URL からは，オンラインコンパイラで生成されたバイナリを確認できる．

本手法は，求められる実引数の型と名前が IDE の入力支援機能により表示されるほか，引数の不一致などの誤りに対しては理解しやすいコンパイルエラーを出力する．setter のメソッドチェーンを用いて名前付き引数を再現するような古典的な手法に比べ，C++ 開発者にとって親しみやすい文法で導入でき，デフォルト引数や出力パラメータの対応といった，関数の引数としての機能面においても優位性があると評価した [95].

4.1.4 C++ における名前付き引数の議論

C++ 言語機能への名前付き引数の導入に関する最近の議論として，Brown による提案 [7] がある．オーバーロードの多いコンストラクタでの利用や，引数の順序の間違いを防ぐことなど，解決したい問題は本研究と共通している．Brown は，引数名の文字列を型情報に含むタグで引数の値をラップするクラスと，コア言語の文法での f(label: value);という簡潔な表現の組み合わせを提案したが，標準化委員会では強い関心は示されず，以降続く提案はない．委員会の関心が低かった理由として，標準ライブラリの範疇ではコンストラクタのオーバーロードが多数必要になるケースは限られ，その一例である std::pairにおいても既にタグ・ディスパッチングが使われていることから，コア言語を変更するというコストには見合わず，優先度が低いと判断されたと考えられる．Siv3D における図形型のような，コンストラクタのバリエーションの要求が大きい

API の導入が標準ライブラリで検討されるようになるまで，標準での名前付き引数のサポートの議論は再開されないだろう．このようなところに，Siv3D の DSL としての独自性，柔軟性の側面が見える．

4.2 ユーザ定義リテラルと，bool 型の strong typedef を用いたコードの表現力向上

関数やパラメータの意味を推測しにくいコードは，間違いを生みやすい．次のコードは昔のバージョンの Siv3D のコードであるが，このコードだけから挙動を正しく読み当てるのは難しい．

```
Timer timer{ 10, true };
```

上記のコードの正解の挙動は，構築と同時に「カウントがスタート」する 10「秒」のタイマーであるが，時間の単位や第二引数の意味は，コメント無しでは伝わらない．このような問題が Siv3D の API の各所に見られたため，

- ユーザ定義リテラルを積極的に活用し，数値リテラルに単位の情報を付与する
- boolean literal を型安全で説明的な定数に置き換える
- 抽象化による実行時性能のオーバーヘッドは発生させない

という方針で，ユーザ定義リテラルと，`bool`型の strong typedef を用いてコードの表現力を向上させた，現在の Siv3D では次のように記述しないとコンパイルできない．

```
Timer timer{ 10s, StartImmediately::Yes };
```

4.2.1 ユーザ定義リテラルの活用

C++11 で導入されたコア言語機能であるユーザ定義リテラルは，指定したサフィックスの付いた数値・文字列リテラルについて，その値を引数とする関数の糖衣構文として機能する特殊な文法である．例えば，次のようなコードでは，`_pi`サフィックスによって，`main()`関数では，円周率の 0.5 倍を表現する `double`型の値を，関数呼び出し演算子や乗算演算子を用いずに得られる．

```
double operator ""_pi(long double x)
{
  return (x * std::numbers::pi);
}

int main()
{
  double a = 0.5_pi; // a == 1.5707...
}
```

C++ 標準ライブラリにおいても，C++14 以降，数値リテラルを時間型に変換する `s`サフィックス，`h`サフィックスなどが定義され，活用事例が増えている．ユーザ定義リテラルを活用することで，コード中のリテラル値に単位が付与され，コードの情報量が増えるほか，関数呼び出しよりも短いコードで，値に対する追加の計算処理を行うことができる．Siv3D が提供するユーザ定義リテラルを表 6 にまとめ，実際のコードでの用法を次に示す．

表 6　Siv3D が提供するユーザ定義リテラル

サフィックス	戻り値の型	引数の型	説明
_pi	double	long double unsigned long long	π
_piF	float	long double unsigned long long	π
_deg	double	long double unsigned long long	度数法からラジアン角へ変換
_degF	float	long double unsigned long long	度数法からラジアン角へ変換
_tau	double	long double unsigned long long	2 π
_tauF	float	long double unsigned long long	2 π
_big	BigInt	const char*	多倍長整数
_bigF	BigFloat	const char*	多倍長浮動小数点数
_dp	DecimalPlaces	unsigned long long	小数点以下の桁数を指定するマニピュレータ
_fmt	FormatHelper	const char32*	Python スタイルの文字列フォーマットの適用
_re	RegExp	const char32*	文字列を正規表現オブジェクトへ変換
_emoji	Emoji	const char32*	文字列を絵文字オブジェクトに変換
_icon	Icon	unsigned long long	整数をアイコン ID に変換

```
64 void Main()
65 {
66   // サイズ10 の正方形を45 度回転させた図形を作成
67   Rect{ 10 }.rotated(45_deg);
68
69   // 文字列を画像ファイルパスとみなし，
70   // 画像ファイルをオープン
71   Texture a{ U"image.png" };
72
73   // 文字列を絵文字とみなし，
74   // 対応する絵文字データを読み込む
75   Texture b{ U"🍣"_emoji };
76
77   // 多倍長浮動小数点数
78   //  （リテラル部分は文字列と見なされる）
79   auto c = 9.99999999999999999999_bigF;
80
81   // Python 風の文字列フォーマット
```

```
  Print << U"{} {}"_fmt(U"Hello", 20);

  // 正規表現のマッチ
  U"(\\d+)-(\\d+)-(\\d+)"_re.match(U"2021-09-01");
}
```

4.2.2 強く型付けされたブーリアン型

boolean literal を関数の実引数で使うと，その値の示す意味が読み手に伝わらず，引数の取り違えやコードの可読性低下につながる．プログラマが名前を付けた定数を用意することで意図を表現できるようにはなるが，強い型付けは無く，引数の順番の取り違えの危険性は残る．

```
void G(bool);

void F(bool formatDrive, bool enableLog)
{
  if (enableLog) { }
  G(formatDrive);
}

int main()
{
  constexpr bool EnableLog = true;
  F(EnableLog, false); // 本当に正しい？
}
```

C++11 以降では，型付けのある enum class を使うことで，型による区別でコンパイル時にエラーを発生させ，取り違えを防ぐことができるが，enum class の値は暗黙に `bool`型に変換できないため，`if ()`や条件演算子などの文脈で `bool`型のように扱うことができず，コードが冗長になる問題がある．

```
enum class FormatDrive { No, Yes };
enum class EnableLog { No, Yes };

void G(bool);

void F(FormatDrive formatDrive, EnableLog enableLog)
{
  if (enableLog) { } // エラー
  // 面倒
  if (enableLog == EnableLog::Yes) { }
  // 面倒
  G(formatDrive == FormatDrive::Yes);
}

int main()
{
  F(FormatDrive::No, EnableLog::Yes);
```

```
  // 取り違えに対してエラーを発生させる
  //F(EnableLog::Yes, FormatDrive::No);
}
```

Siv3D は，このギャップを埋めるために，次のような bool型に特化した strong typedef のクラステンプレートを実装し，ライブラリの API に広く導入した.

```
// 異なるタグのYesNo 間の変換は不可
template <class Tag>
struct YesNo
{
  struct Helper { bool yesNo; };

  explicit constexpr YesNo(bool yesNo) noexcept
    : m_yesNo{ yesNo } {}

  constexpr YesNo(Helper helper) noexcept
    : m_yesNo{ helper.yesNo } {}

  // contextually converted to bool
  // の文脈でbool 型の値としてふるまう
  explicit constexpr operator bool() const noexcept
  {
    return m_yesNo;
  }

  // それ以外の文脈ではこの関数を使う
  constexpr bool getBool() const noexcept
  {
    return m_yesNo;
  }

  // 比較演算子
  constexpr auto operator <=>(const YesNo&) const noexcept = default;

  // ○○::Yes と記述できるように
  static constexpr Helper Yes{ true };

  // ○○::No と記述できるように
  static constexpr Helper No{ false };

private:

  bool m_yesNo;
};
```

このクラスは, operator boolのオーバーロードにより, if ()や条件演算子のような contextually converted

to bool の文脈で bool型の値のようにふるまうため，enum class を使う方法と比較して，bool型の変数を利用している既存コードからの変更が最小限で済む．

```
// タグ名は任意
using FormatDrive = YesNo<struct FormatDrive_tag>;
using EnableLog = YesNo<struct EnableLog_tag>;

void G(bool) {}

void F(FormatDrive formatDrive, EnableLog enableLog)
{
  if (enableLog) { }
  G(formatDrive.getBool());
}

int main()
{
  F(FormatDrive::No, EnableLog::Yes);
  // 取り違えに対してエラーを発生させる
  //F(EnableLog::Yes, FormatDrive::No);
}
```

Siv3D では，全部で 33 種類の YesNoクラス特殊化が使用されている．実際のコード例 [87] を次に示す．YesNoクラスの導入前は，これらのパラメータが true，falseで記述されていたことを想像すると，このクラスがコードの表現力向上に果たした役割は明らかである．

```
# include <Siv3D.hpp>

void Main()
{
	// ストップウォッチを即座に開始
	Stopwatch stopwatch{ 15s, StartImmediately::Yes };

	Image image{ 256, 256, Palette::White };

	// アンチエイリアスを有効にして円を画像に描き込む
	Circle{ 128, 128, 60 }.paint(image, Palette::Orange, Antialiased::Yes);

	// 画像をWebP のロスレス形式で保存
	image.saveWebP(U"image.webp", Lossless::Yes);

	// ファイルをゴミ箱に送らず削除する
	FileSystem::Remove(U"image.webp", AllowUndo::No);

	const LineString lines
	{
		Vec2{ 500, 100 }, Vec2{ 700, 200 }, Vec2{ 600, 500 },
```

```
        };

        // 各点を結んだ連続する線分の長さを計算（終点と始点をつないでリング状にする）
        const double length = lines.calculateLength(CloseRing::Yes);
}
```

なお，主要なコンパイラの最適化を有効にしたビルドにおいて，YesNoクラスの使用は，単純に bool型の値を使用したときと同じアセンブリが生成されることを確認している．

4.3 サブシステムの初期化と終了処理の効率的な実装

Siv3D のように多くのサブシステムから構成されるライブラリの一般的なデザインパターンの 1 つは，各サブシステムをシングルトンクラスとすることである．ただし，サブシステムには依存関係があり，正しい順番で初期化と終了処理が行われる必要があるため，順序を制御できるような工夫が必要である [4][32].

Siv3D では，サブシステム管理の設計にあたって，サブシステムを追加する際のコードの変更を最小にするような，シングルトンとして作成されるサブシステム管理クラスを設計した．すべてのサブシステムを単一の std::tupleにまとめて格納することが特徴で，std::get()を個別のサブシステムの getter の代わりにでき，また，サブシステムの初期化と終了処理のタイミングを短いコードで一括して制御できるという利点がある．

Siv3D のサブシステム管理のコードを次に示す．例として IRendererと IAudioというサブシステムを管理している．

```
// SubSystem.hpp
# pragma once
# include <cassert>

template <class ISubSystem>
class SubSystem
{
public:

  ~SubSystem()
  {
    // Manager::~Manager() によって破棄済み
    assert(pSubSystem == nullptr);
  }

  ISubSystem* get() noexcept
  {
    return pSubSystem;
  }

  void release()
  {
    delete pSubSystem;
    pSubSystem = nullptr;
```

```
  }

private:

  // ファクトリ関数でサブシステムを生成
  ISubSystem* pSubSystem = ISubSystem::Create();
};
```

```
// Manager.hpp
# pragma once
# include <tuple>
# include "SubSystem.hpp"

// 前方宣言だけで OK.
// サブシステムのコード変更の影響を受けない
class IRenderer;
class IAudio;

class Manager
{
public:

  Manager() noexcept;

  ~Manager();

  template <class Interface>
  static auto* Get() noexcept
  {
    return std::get<SubSystem<Interface>>(pManager->m_subsystems).get();
  }

private:

  inline static Manager* pManager = nullptr;

  // サブシステムをこのtuple に追加
  std::tuple<
    SubSystem<IRenderer>,
    SubSystem<IAudio>>
    m_subsystems;
};

// 短いコードでサブシステムに
// アクセスできる便利なマクロ
#define GET(SUBSYSTEM) Manager::Get<I##SUBSYSTEM>()
```

```
// Manager.cpp
# include "Manager.hpp"

// サブシステムのヘッダ
# include "IRenderer.hpp"
# include "IAudio.hpp"

template <size_t I, class Tuple>
static void Release(Tuple&& t)
{
  std::get<I>(t).release();
  if constexpr (I > 0)
    Release<I - 1>(t);
}

template <class Tuple>
static void ReleaseAll(Tuple&& t)
{
  return Release<std::tuple_size_v
    <std::remove_reference_t<Tuple>>-1>
    (std::forward<Tuple>(t));
}

Manager::Manager() noexcept
{
  pManager = this;
}

Manager::~Manager()
{
  // サブシステムを初期化と逆順ですべて終了
  ReleaseAll(m_subsystems);
  pManager = nullptr;
}
```

```
// IRenderer.hpp
#pragma once

class IRenderer // サブシステムの例
{
public:

  static IRenderer* Create();

```

```
  virtual ~IRenderer() = default;

  virtual void draw() = 0;
};
```

```
// IAudio.hpp
#pragma once

class IAudio // サブシステムの例
{
public:

  static IAudio* Create();

  virtual ~IAudio() = default;

  virtual void play() = 0;
};
```

```
// Library.cpp
# include "Manager.hpp"
# include "IRenderer.hpp"

void DrawSomething()
{
  // IRenderer に直接アクセス
  GET(Renderer)->draw();
}

int main()
{
  // シングルトンオブジェクトを作成
  Manager mgr;

  DrawSomething();
} // デストラクタでサブシステムを順番に破棄
```

ここに新しくサブシステムを追加したい場合,

- Manager.hpp に前方宣言を追加
- サブシステムを tuple の要素に追加
- Manager.cpp でヘッダをインクルード

という 3 行の変更だけで済む簡便さが特長である.

4.4 RAII を利用したレンダーステートの管理

Resource Acquisition Is Initialization (RAII) は，オブジェクトの初期化にリソースの確保を，破棄に解放を関連付ける，C++ の基本的なイディオムである．標準ライブラリでは `std::fstream`や，スマートポインタ，`std::lock_guard`等で活用されていてる．Siv3D においても，テクスチャや音声データなど，メモリ上のリソースを管理するクラスで必ず使われており，そのおかげで利用者はあらゆる Siv3D のクラスについて，明示的なリソースの解放や破棄の操作をコードに記述する必要がない．

ところで，RAII における管理対象をリソースからステートに拡張すると，グラフィックスのプログラムで頻繁に登場する，レンダーステート（描画の方法）や，使用するシェーダプログラム（GPU 用のプログラム）の切り替え，座標変換行列のスタック等の操作を，コードで記述するうえで便利になる．Siv3D では，レンダーステートを，そのオブジェクトのスコープが有効な間だけ適用する `ScopedRenderStates2D`クラスや，コンストラクタで新しい座標変換行列を現在の行列スタックに push し，デストラクタで pop するような`Transformer2D`といったクラスを提供する．グローバルなレンダーステートに対して行った変更がデストラクタで自動的にリセットされるため，関数の呼び出し先で行われた変更が呼び出し元に影響することが無くなり，プログラムの見通しが良くなるほか，コードの再利用性が高まる．Siv3D におけるプログラムの例を次に示す．

```
1  void A()
2  {
3    ScopedRenderStates2D s{
4      BlendState::Additive,
5      SamplerState::ClampNearest };
6
7    Draw();
8  } // s のデストラクタが
9    // レンダーステートをリセット
10
11 void B()
12 {
13   A();
14
15   // ここではA() の中での
16   // レンダーステート変更の影響を受けない
17   Draw();
18 }
```

このコードでは，関数 `B()`が呼び出す `A()`において，`ScopedRenderStates2D`型の変数 `s`のコンストラクタに渡された 2 つのレンダーステートが，コンストラクタを通して有効化され，この際，古い設定がメンバ変数に記録される．新しいレンダーステートに基づいて `Draw()`による描画が行われたあと，`A()`関数の終了に伴い，`s`のデストラクタが実行される．このとき，記録されていた古い設定を用いて，現在のレンダーステートをリセットする処理が実行される．したがって，関数 `B()`の本体に記述された `Draw()`は，`A()`の内部で記述されたレンダーステート変更の影響を受けない．

```
void C()
{
  // スケールをpush
  Transformer2D t1{ Mat3x2::Scale(2.0) };

  // 2 倍のスケールで描画される
  Draw();
  {
    // スケールをpush
    Transformer2D t2{ Mat3x2::Scale(1.5) };

    // 3 倍のスケールで描画される
    Draw();
  } // t2 のデストラクタがスケールをpop

  // 2 倍のスケールで描画される
  Draw();
} // t1 のデストラクタがスケールをpop
```

このコードでは，関数 C()において作成される Transformer2D型の変数のコンストラクタが，グローバルステートである座標変換行列スタックに新しい行列を push し，デストラクタが pop の操作を行う．t1は 2 倍の大きさで描画する拡大縮小行列を push し，t2は 1.5 倍の大きさで描画する拡大縮小行列を push することで，2 回目に登場する Draw()は，それらの積である 3 倍の大きさでグラフィックスの描画を行う．直後に実行される t2のデストラクタによって，最後の行列スタックに最後に追加された要素が pop されることで，3 回目に登場する Draw()は，2 倍の大きさで描画されることになる．

サンプルコードにおいて，変数 s，t1，t2は明示的に使われず，無意味にも見えてしまうため，最初はこの挙動についての慣れが必要である．実際にこれらのコードの意味を尋ねる質問もコミュニティに複数投稿された．C++ 標準ライブラリにおいて，デストラクタのためだけに変数を作成するような機能の事例が少ないことが違和感の原因だろう．標準ライブラリで似たような使い方をする API としては，使用頻度の高い例に std::lock_guardが存在するが，マルチスレッドプログラムを書かない C++ 開発者にとってはなじみが薄い．現在，C++ 標準ライブラリにより汎用的なスコープガードや RAII ラッパを導入する提案 [79] が出されており，こうしたテクニックが一般化することで，Siv3D のコードへのハードルも下がるのではないかと期待される．

4.5 ヘッダファイルのドキュメント性の向上

C++ の IDE は，関数や定数の宣言箇所にコメントで記述されたドキュメントを認識し，コーディングのヒントとして，コードエディタ上にツールチップ等で表示する機能を有する．また，エディタ上でプログラマが関数や値を選択して，それらの宣言箇所へジャンプできる機能も，近年のソースコードエディタは備えている．こうしたことから，ライブラリのヘッダファイルにはユーザのための情報が適切に記述されていることが望ましい．Siv3D のヘッダファイルの書き方について，採用した方針を説明する．

4.5.1 ドキュメンテーションタグ

2021 年現在の主要なコードエディタで最も広くサポートされている C++ ドキュメンテーションタグの記述方式は Doxygen[19] である（そのほかに，一部のコードエディタでサポートされる記述方式として XML コメント [53] がある）．Siv3D では，Doxygen タグのうち，関数やクラスの機能の概要を説明する`@brief`タグ，引数を説明する`@param`タグ，テンプレート引数を説明する`@tparam`タグ，戻り値がある場合にその意味を説明する`@return`タグをすべての API に記述することを目標とした．必要に応じて`@remark`タグにおいて，関数の副作用や使い方の注意など補足の説明を行った．例えば，`std::basic_string::size()`と`std::basic_string::length()`がそうであるように，同じ挙動をする別名の関数については，ユーザが混乱しないよう，お互いが同じものであることを明記した．

4.5.2 public メンバと private メンバの順序

ヘッダに記述するクラスの public メンバを private メンバよりも前に配置することで，ヘッダファイルを閲覧したユーザが必要とする，公開メンバに関する情報に早くたどり着けるような工夫をした．ただし，public メンバで使用するために必要な private 指定の宣言がある場合は，この原則にこだわらず，順序を部分的に変更する場合もあった．

4.5.3 .ipp ファイルの導入

ヘッダファイルに記述されるインライン関数の実装は，ヘッダファイルの行数を増加させる．ヘッダファイルを閲覧するユーザにとって関心が高い，インタフェースやドキュメンテーションに関する情報がソースコードエディタの画面に出現する頻度が低下し，ライブラリ機能の一覧性や検索性が損なわれる．Siv3D では当初，.hpp ファイル + .cpp ファイルの構成による運用をしていたところ，インライン関数で記述されなければならない constexpr である関数が多用されるにつれ，こうした問題が顕在化してきたことから，ファイル構成を見直し，インライン関数の定義の記述に特化したヘッダファイル（拡張子 .ipp）を，インライン関数を持つほとんどのヘッダファイルに追加した．導入後，.hpp ファイルには基本的にドキュメントとインタフェースのみが記述されるようになった．

ただし，インライン関数の実装のすべてを.ipp ファイルに移動することはできない．Hidden Friedns [6] の関数の実装をヘッダ内のクラス定義の外に記述すると，そのヘッダをインクルードした翻訳単位では Hidden Friends としての効力が失われる．部分的な解決方法として，Hidden Friends の関数の本体を.cpp ファイルに記述することができる．その場合，関数を実装した翻訳単位（通常はユーザに公開されないライブラリの内部コード）においては Hidden Friends としての効力を失うが，ヘッダをインクルードしたソースファイルでは Hidden Friends としてふるまう．ただし，関数のインライン指定が必要な，constexpr である関数ではこの解決方法が使えないという制約があるため，constexpr かつ Hidden Friends であるメンバ関数の本体は必要に応じて.hpp ファイルに残されたままである．

.ipp ファイルの導入前後で，各ファイルに含まれるコードがどのように変化するか，説明用のクラス `A`を用いたサンプルコードを示す．.hpp ファイルの読みやすさの向上に注目されたい．

.ipp ファイル導入前:

```
1 // A.hpp
2 /// @brief ドキュメント
3 class A
```

```
{
public:

  /// @brief ドキュメント
  A() = default;

  /// @brief ドキュメント
  constexpr A(int a, int b)
    : m_data{ a * b } {}

  /// @brief ドキュメント
  constexpr int getData() const noexcept
  {
    return m_data;
  }

  /// @brief ドキュメント
  constexpr void increment() noexcept
  {
    ++m_data;
  }

  /// @brief ドキュメント
  void update();

  /// @brief ドキュメント
  friend constexpr bool operator ==(int lhs, const A& rhs) noexcept
  {
    return (lhs == rhs.getData());
  }

  /// @brief ドキュメント
  friend std::ostream& operator << (std::ostream& os, const A& a);

private:

  int m_data;
};
```

```
// A.cpp
# include "A.hpp"

void A::update()
{
  // ...
}
```

```
8
9  std::ostream& operator << (std::ostream& os, const A& a)
10 {
11   return os << a.getData();
12 }
```

.ipp ファイル導入後:

```
1  // A.hpp
2  /// @brief ドキュメント
3  class A
4  {
5  public:
6
7    /// @brief ドキュメント
8    A() = default;
9
10   /// @brief ドキュメント
11   constexpr A(int a, int b);
12
13   /// @brief ドキュメント
14   constexpr int getData() const noexcept;
15
16   /// @brief ドキュメント
17   constexpr void increment() noexcept;
18
19   /// @brief ドキュメント
20   void update();
21
22   /// @brief ドキュメント
23   friend constexpr bool operator ==(int lhs, const A& rhs) noexcept
24   {
25     return (lhs == rhs.getData());
26   }
27
28   /// @brief ドキュメント
29   friend std::ostream& operator << (std::ostream& os, const A& a);
30
31 private:
32
33   int m_data;
34 };
35
36 # include "detail/A.ipp"
```

```
1  // A.ipp
2  constexpr A::A(const int a, const int b)
```

```
  : m_data{ a * b } {}

constexpr int A::getData() const noexcept
{
  return m_data;
}

constexpr void A::increment() noexcept
{
  ++m_data;
}
```

```
// A.cpp
# include "A.hpp"

void A::update()
{
  // ...
}

std::ostream& operator << (std::ostream& os, const A& a)
{
  return os << a.getData();
}
```

.ipp ファイルの導入はライブラリコードの合計行数を増大させるが，ユーザ視点ではヘッダの一覧性が改善される．またクラス A のインライン関数が，別の同様に書かれたクラス B に依存していた場合，

- A.hpp → B.hpp

の順でのインクルードはできないが，.ipp ファイルの導入後は

- A.hpp → B.hpp → A.ipp → B.ipp

の順で（前方宣言を適切に用いて）インクルードすることにより，依存関係の問題を容易に回避してコンパイルできる．また，依存関係のためにヘッダ内で constexpr 化できなかったメンバ関数を constexpr 化できるようになり，ライブラリ API 設計面でメリットがある．

4.6 C++ の難しさを低減させるための標準ライブラリ拡張

C++ においては，コンテナクラスの要素を無作為にシャッフルしたり，ある条件を満たす要素だけを削除したりするといった場合には，一般的に Standard Template Library (STL) の複数の関数を組み合わせた，「イディオム」と総称される種々の操作を用いる．こうした STL の独特なイディオムはコードの汎用性，再利用性の観点からは合理的であり，習得するほど C++ プログラムを効率的に記述できるようになることから，C++ の生みの親である Stroustrap も「expert friendly（熟練者に優しい）」と評している [62] が，つまりは入門者や，カジュアルに C++ コードを書きたいユーザにとっては余計な学習コストが生じ，不親切であるこ

図 30　IDE によるメンバ関数の入力補完は，オブジェクト指向言語の構文がもたらすメリットの 1 つである

とを示唆している．こうしたイディオムを知らないプログラマが非効率にコードを再実装してしまうのは非合理的である．

Siv3D ではこの問題の対策として，標準のコンテナクラスを包含する拡張コンテナクラスを用意し，より使いやすいコードによって，要素に対する様々な操作を実現できるメンバ関数の提供を行った．次にその例を示す．

```
1  // 標準ライブラリのコンテナ
2  std::vector<int> v1;
3
4  // Siv3D の拡張コンテナ
5  Array<int32> v2;
6
7  // 偶数である要素を削除する
8  v1.erase(std::remove_if(v1.begin(), v1.end(), IsEven), v1.end());
9  v2.remove_if(IsEven);
10
11 // 42 という要素が存在するか調べる
12 Print << (std::find(v1.begin(), v1.end(), 42) != v1.end());
13 Print << v2.includes(42);
14
15 // すべての要素が偶数であるかを調べる
16 Print << std::all_of((v1.begin(), v1.end(), IsEven);
17 Print << v2.all(IsEven);
```

コード例からわかるように，既存のイディオムはメンバ関数に対する IDE の入力補完を用いたコーディングの省力化と相性が悪い．Siv3D のように基本的な操作をコンテナに対するメンバ関数に集約することで，具体的なイディオムのパターンを習得しなくても，IDE の入力補完によって選択肢が提示されることは，プログラマの助けになる (表 30)．Siv3D で標準ライブラリコンテナを拡張したクラスのメンバ関数は基本的にイン

ライン関数であり，コンパイラの最適化によって抽象化のオーバーヘッドは発生しない．

4.7 演算子オーバーロードの柔軟な活用

C++ の演算子オーバーロードは，プログラム定義型に対して，+や<<など，各種演算子のオーバーロードを定義できる機能である．例えば，線形代数ライブラリにおいて，教科書に書かれるような数式と C++ コードを同じような見た目で記述できるよう，ベクトルクラスや行列クラスに +や-, *などの演算子オーバーロードを定義するといった使い方がある．

```
1 Vec2 va{ 1.0, 2.0 }, vb{ 0.1, 0.2 };
2 Vec2 vc = va + 2 * vb; // { 1.2, 2.4 }
```

演算子オーバーロードを言語機能として持たない言語，例えば Java では，任意精度整数を表現できる`BigDecimal`型で数式を表現する際に，`.add()`, `.subtract()`のようなメソッドを用いる必要があり，コードの可読性が損なわれ，計算順序の間違いにも気づきにくくなるといった問題を引き起こす [1].

C++ の演算子オーバーロードは，記号が本来持つ数学的な定義から外れた使い方も可能である．実際に標準ライブラリにおける例として，文字列の結合における +演算子や，ストリームへの出力における左シフト演算子が挙げられる．

```
1 std::string a = "Hello, ", b = "world!";
2 std::string c = a + b; // "Hello, world!"
3 std::cout << c << '\n';
```

演算子オーバーロードは通常の関数の代替となるもので，コードの記述の短縮につながる．ただし，Herb Sutter らが指摘 [86] するように，演算子の意味があいまいもしくは非直感的な場合には，説明的な名前を持つ関数を使うべきであり，やみくもに演算子のオーバーロードを使用してコードに疑問の余地を残すことは，避けなければならない．

Siv3D では，誤った解釈の余地が少なく，コードが明快になると考えられるケースに限り，数学的用法以外の文脈でも演算子のオーバーロードを活用している．その一例が 4.1 で述べた名前付き引数における=演算子であるが，本節ではそれ以外に 2 つの事例を紹介する．

4.7.1 キーボード入力の組み合わせ

コンピュータで 2 つのキーボード入力を行う操作は，図 31 のように，慣習的に「Ctrl + S を押す」「W キーまたは↑キーを押す」といった言葉で表現される．

To do this	Press
Open a document.	Ctrl+O
Create a new document.	Ctrl+N
Save the document.	Ctrl+S
Close the document.	Ctrl+W
Cut the selected content to the Clipboard.	Ctrl+X

図 31　ショートカットキーの説明で慣例的に使われる + 記号（Microsoft Word のドキュメント [51] より）

ここで注意すべきことは，「Ctrl + S」は，単純に 2 つのキーを押すという意味ではなく，Ctrl キーが押され続けている状態で S キーを 1 度押すことを意味している．この操作を，まずは単純に Siv3D のコードで表現すると次のようになる．

```
// Ctrl キーが押されている，かつS キーが押された
if (KeyControl.pressed() && KeyS.down())
  ...

// W キーがが押された，または↑キーが押された
if (KeyW.down() || KeyUp.down())
  ...
```

Siv3D では，個別のキーの表現に用いられる Inputクラスに operator+と operator|を定義し，キーの集合を表現する複数のクラス（InputCombination, InputGroup）を戻り値として返すことで，前述のような，「同時に押された」「どちらかが押された」という関係を，自然言語での表現に近い簡潔なコードで記述できるようにした．

```
// Ctrl + S キーが押されたら
if ((KeyControl + KeyS).down())
 ...

// W キーまたは↑キーが押されたら
if ((KeyW | KeyUp).down())
  ...
```

この Input型の定数はキーボードのキーだけでなく，マウスのボタンやゲームコントローラの各ボタンにも割り当てられているため，キーボード，マウス，ゲームコントローラの 3 種類の方法で操作するようなインタラクションを，次のように並べて記述できる利点がある．

```
// エンターキーまたはマウスの左ボタンまたはゲームコントローラのA ボタンが押されたら
```

```
2 if((KeyEnter | MouseL | XInput(0).buttonA).down())
3   ...
```

Inputクラスと，それらの集合を表現するクラスにおける演算子の定義は，次のような組み合わせにより，必要なパターンを網羅した．

```
1  class Input
2  {
3  public:
4    InputCombination operator +(Input other) const;
5    InputGroup operator |(Input other) const;
6    InputGroup operator |(const InputCombination& other) const;
7    InputGroup operator |(const InputGroup& other) const;
8    ...
9  };
10
11 class InputCombination
12 {
13 public:
14   InputGroup operator |(Input other) const;
15   InputGroup operator |(const InputCombination& other) const;
16   InputGroup operator |(const InputGroup& other) const;
17   ...
18 };
19
20 class InputGroup
21 {
22 public:
23   InputGroup operator |(Input other) const;
24   InputGroup operator |(const InputCombination& other) const;
25   InputGroup operator |(const InputGroup& other) const;
26   ...
27 };
```

4.7.2 時間計測クラスの比較演算子

ストップウォッチやタイマーの働きをするクラスのオブジェクトに対して，経過時間を問い合わせる場合，メンバ関数を用いて経過時間を数値型で取得し，それを別の数値と比較するのが，最も単純な実装になるだろう．その場合，問い合わせるメンバ関数で時間の単位を決定し，比較する数値の単位をそれに揃えるという認知負荷がプログラマに発生し，間違いの原因になりうる．

```
1 Stopwatch stopwatch = ...;
2
3 // （間違い）ストップウォッチが3 秒以上経過していたら，というつもりで書いた
4 if (3 <= stopwatch.ms())
5   ...
```

時間を問い合わせるとき，メンバ関数が std::chrono::duration型で時間を返せば，プログラマは時間リテラル (4.2 参照) を用いて，左辺の時間の単位だけに注目すればよくなり，間違いの可能性を減らせる．

```
Stopwatch stopwatch = ...;

// ストップウォッチが3000 ミリ秒以上経過していたら
if (3000ms <= stopwatch.elapsed())
  ...
```

さらに，Siv3D では，時間計測クラスと比較演算子の組み合わせは，経過時間の問い合わせ以外の文脈で用いられることが考えにくい点に注目し，時間計測クラス自体に std::chrono::duration型との比較演算を定義することで，間違いの可能性を減らしつつ，さらにコードを短縮できる API を実装した．

```
Stopwatch stopwatch = ...;

// ストップウォッチが3000 ミリ秒以上経過していたら
if (3000ms <= stopwatch)
  ...
```

本節で紹介した，「A+B キーを押す」「ストップウォッチが N 秒以上経過していたら」のような文脈で演算子オーバーロードを活用する事例は，ある機能を説明する自然言語の文章に，演算子に相当する表現が登場する場合，それを記述するプログラムに演算子のオーバーロードを自然に導入できることを示唆しており，C++ フレームワークの API デザインにおける演算子オーバーロードの適切な活用機会を増やすヒントになるだろう．

5　Siv3D の普及とユーザコミュニティの発展のための施策とその評価

開発したフレームワークが世の中の利用者のニーズに合致し，フレームワークを継続的に使いたいと思う利用者が増えると，ユーザコミュニティが形成される．コミュニティ内で発生する情報交換を通して，利用者はフレームワークをより効果的に活用するための知識を獲得し，一方でフレームワーク開発者はコミュニティからの報告や要望から，フレームワークのニーズや課題を発見する．本章では Siv3D の普及とユーザコミュニティの発展に関する方針と施策を説明し，その評価を報告する．

5.1　ユーザコミュニティ形成の目的

Siv3D は公開当初から，次のようなねらいのためにユーザコミュニティの育成と拡大を目指してきた．

- 多数のユーザによって，様々な実行環境，ツール，デバイス，技術との組み合わせが提案，検証される
- Siv3D という「共通の言語」を多くの人が共有することで，アプリケーション開発，プログラミング技術の情報交換が円滑になり，個々人の学習，コミュニティ，文化の発展が促進される
- 増加するユーザの質問やトラブルに対応できる熟練ユーザを育成することができる

実際に，Siv3D の主要な機能のいくつか（Linux 対応，Web 対応，2D 物理演算機能，HTTP クライアント）は，ユーザのコントリビューションにより実現された．また，日本のオンラインの技術記事共有サイト Qiita において 2011 年から毎年実施されている「Qiita Advent Calendar」と呼ばれるイベント（25 日間のイベント期間中にそれぞれの技術コミュニティの中で毎日技術記事を投稿する [65]）では，Siv3D ジャンルが 2015 年に初登場して以降，毎年 20 人前後が技術記事を投稿するなど情報交換が活発に行われている．Siv3D ユーザコミュニティの中核となっているグループウェアのワークスペース「Siv3D ユーザコミュニティ Slack」には 2021 年時点で 535 人が登録し，使い方に困っている初心者の質問に対し，経験のあるユーザが回答を投稿するなど，利用者へのサポート負担の軽減にも効果を上げている．

5.2　ユーザコミュニティ発展のための施策とその評価

Siv3D は，フレームワークの普及とユーザコミュニティの発展のために，次の 3 つの施策を継続してきた [98].

勉強会　Siv3D の開発環境のセットアップ方法やチュートリアルを解説する 2〜6 時間のイベント．数名〜数十名が参加する．年に数回の頻度で開催.

実装会　Siv3D のアクティブユーザおよびコミッタを対象に，コードレビューや技術的な助言，開発ツールの使用法指導などを行う 2〜4 時間のイベント．数名〜十数名が参加する．毎月開催.

チャレンジ　筆者が Siv3D に実装したい新機能を課題形式にしてまとめ，ユーザコミュニティに解決をしてもらう，オンラインで数カ月間にわたって開催するイベント．年に 1 回の頻度で開催.

5.2.1　勉強会

Siv3D の勉強会は，筆者が講師となり，Siv3D プログラミングの初体験者に質の高いチュートリアルを提供する 2〜6 時間のイベントである．入門でのつまづきを直接フォローし，継続して利用・学習するユーザを増

やすことを目的としている．様々な地域で実施しているのが特徴で，2014 年〜2019 年の期間で，9 都県，11 か所が会場となった [77]．筆者の拠点から遠隔の地域での開催にあたっては，現地で活動するプログラミングコミュニティ（CoderDojo など）や，大学の情報系サークルなど，ローカルのコミュニティとの連携を重視した．イベントをローカルのコミュニティの主催とすることで，コミュニティの活動実績に貢献すると同時に，ローカルのスタッフが行うことが合理的であるような会場の手配や集客を手伝ってもらう，という形式が特徴である．次の URL で，実際に Siv3D の勉強会を主催したコミュニティ当事者による開催報告を読める．

CoderDojo **静岡（非営利のプログラミング教育コミュニティ）** `https://coderdojo-shizuoka.org/9th` (also at Internet Archive `https://web.archive.org/web/20210119074813/https://coderdojo-shizuoka.org/9th`)

千葉大学 CCS**（大学の情報系サークル）** `https://mattyan1053.hatenablog.com/entry/2020/02/21/202016` (also at Internet Archive `https://web.archive.org/web/20200821231445/https://mattyan1053.hatenablog.com/entry/2020/02/21/202016`)

勉強会の運営の工夫として，ローカルコミュニティに所属するメンバーや参加者の活動を発表し合うライトニングトーク (LT) 時間を設けるなど，一方向ではなく双方向の情報交換ができるイベントをデザインしてきた．LT の発表テーマは Siv3D に限らないため，直接 Siv3D フレームワークの発展に寄与しない場合もあるが，Siv3D を体験した人たちが，他者に対して情報や知見を発信する経験を磨くことで，それぞれの参加者が所属する別のコミュニティに対しても，Siv3D についての情報を発信・共有してくれることを期待して実施してきたものである．

5.2.2 実装会

Siv3D の実装会とは，Siv3D のフレームワーク本体およびドキュメントのコードリポジトリへのコミッタを育成することを目的としたイベントで，Siv3D のアクティブユーザおよびコミッタを参加者として想定し，技術的な議論や共同開発を推進するような内容になっている．勉強会よりも参加定員を少なく設定することで，一人あたりに多くの時間を割き，高度な技術サポートやコードレビューを行う．地方開催においては，勉強会とセットで開催されることもある．図 32 は早稲田大学で開催された実装会の様子である．参加者の操作画面が，それぞれの背後に設置された 1 人 1 台のモニターに表示され，コードレビューや動作の確認などをスムーズに行える環境となっている．

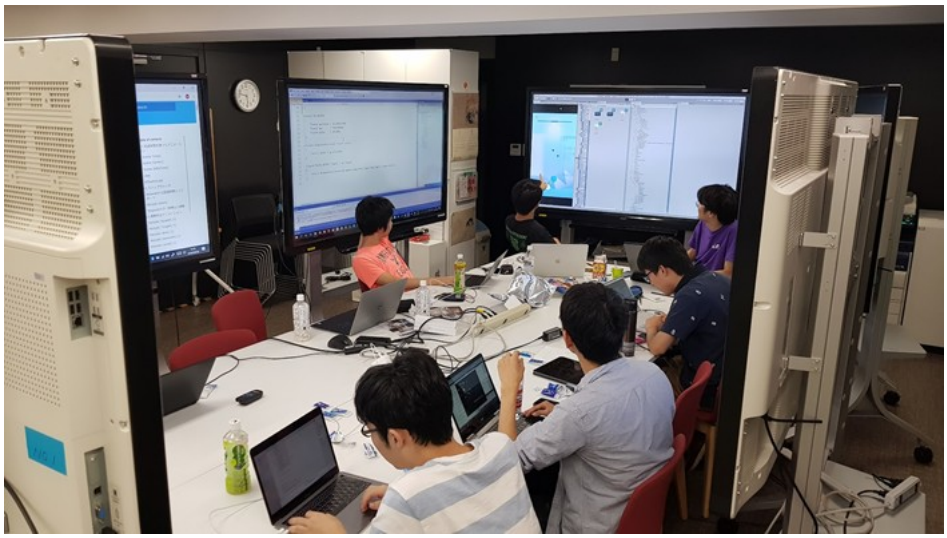

図 32　実装会の様子（2019 年，筆者撮影）

2021 年時点で Siv3D のコードリポジトリにおいて一行でも変更を行ったコミッタは 32 名いるが，そのうち 18 名が，実装会参加後にコミッタとなったメンバーであり，実装会の施策は協働体制の発展に顕著な効果を上げている．2020 年のパンデミック以降は，オンラインでの開催にシフトしたことで，地理的に遠隔の地域からの参加者の割合が増えている．

5.2.3　チャレンジ

Siv3D の「チャレンジ」は，Google Summer of Code(GSoC)[29][74] を参考にしたイベントで，Siv3D に実装されると便利である様々な機能の実装課題を筆者がリストアップし，ユーザコミュニティの有志に挑戦してもらう取り組みである．GSoC と異なり金銭的な支援はないが，挑戦者どうしがコミュニケーションできる専用のグループチャットを用意したり，筆者が技術的なサポートやコードレビューを頻繁に提供したりすることで参加者をサポートする．それぞれの課題は成果物が Siv3D のコードリポジトリにマージ可能な品質に到達するまで続き，コードを提案したり，議論に参加したりするなど，課題の解決過程に少しでも参加した挑戦者の名前が，コードリポジトリのソースコード内にコメントで記録される．

Challenge 05 | Saphe2D::Squircle()

正方形と円形の中間の形状で、`RoundRect` よりもなめらかなカーブを持つ「Squircle」という形状があります。スマホアプリのアイコンなどで使われています。この形状を Shape2D で作成できるようにしてください。

実装方法

最終的には `Shape2D::Hexagon()` のように `Shape2D` の静的メンバ関数として実装しますが、そのためのリファレンス実装を、フリー関数（非メンバ関数）として Main.cpp に実装してください。

`Shape2D` は **頂点配列** `Array<Float2>` と **(三角形)インデックス配列** `Array<TriangleIndex>` (v0.4.3 以前は `Array<uint16>`) で構成されます。Shape2D における頂点配列は、図形を構成する頂点座標を時計回りに並べた配列で、後者はその何番目の頂点 (3 つ、時計回り) を組み合わせてポリゴンの一部を構成する三角形を描画するかを指定するデータです。例えば長方形であれば 4 つの頂点と 2 つの三角形インデックスで表現できます。五角形 (`Shape2D::Pentagon()`) であれば 5 つの頂点と 3 つの三角形インデックスで表現できます。

```
# include <Siv3D.hpp> // OpenSiv3D v0.6

Shape2D Squircle()
{
	return{};
}

void Main()
{
	while (System::Update())
	{
		Squircle().draw();
	}
}
```

Siv3D Slack チャンネル

- `#ch05-squircle`

参考コード

- チュートリアル 多角形を描く
- Shape2D.hpp
- SivShape2D.cpp

参考資料

- https://en.wikipedia.org/wiki/Squircle
- https://mathworld.wolfram.com/Squircle.html
- https://www.johndcook.com/blog/2018/02/13/squircle-curvature/

より完成度を高めるために

- なるべく計算量を小さくする
- カーブの品質（頂点数）を引数で制御できるようにする

図 33　チャレンジ課題の例（2021 年のイベントページ [88] のスクリーンショット）

チャレンジ課題の出題形式の例を図 33 に示す．一般的なオープンソースソフトウェアで新機能をリクエスト・提案するような Issue と比べ，実装者にとって丁寧な記述になっている．具体的には，実装方針や参考コード，筆者による解決のためのヒントが示されており，課題のための専用のグループチャット (Slack チャンネル）が用意されることで，挑戦者が課題解決に着手しやすいことが特徴である．

過去のチャレンジでは，課題発表時点で具体的な解決法の見当がついていないものもあったが，挑戦者の調査や研究によって 10 件以上の課題が達成された．例えば「Siv3D で世界地図や日本地図を表示できるようにしたい」という目標は，地理空間情報を記述するテキストフォーマット GeoJSON のパーサと，読み取ったデータを Siv3D の基本図形クラス Polygon 型に変換する関数の実装によって実現された．

5.3 コミュニティ運営に利用したサービス

フレームワークの利用者が求めるコミュニケーションは大きく次の通りに分類される．

- 技術的な問題の解決への助けを求める
- 他の利用者を助ける
- 近い趣味を持つ他の利用者と交流する
- 自分の成果物を披露し，反応やフィードバックを得る
- フレームワーク開発者の考えを知る
- フレームワークの開発を助ける

Twitter のような SNS だけでも，これらについて一定のことは達成できるが，情報が散逸し再利用が難しくなる．また，一般向けの SNS のコミュニケーション機能は，複数人による共同作業や，情報へのアクセス権限の細かいコントロールを想定していないため，生産的な作業に向いていない．そこで Siv3D では，ユーザコミュニティの発展のために，利用者とのコミュニケーションを促進するツールやサービスを導入し，公式 Web サイト上で参加を呼びかけてきた．その中で現在まで継続して活用され，Siv3D のユーザコミュニティの発展に貢献したツールを紹介する．

Slack　登録しているメンバーとメッセージやファイルをやり取りできるグループウェアである．Siv3D が提供するワークスペースでは，ユーザが質問を投稿すると誰かが回答してくれる質問チャンネルや，特定の新機能の開発に関心のあるメンバーが進捗報告や技術的な相談をするチャンネルなどが活発である．2021 年時点で 535 人が Siv3D のワークスペースに登録している．

Discord　無料で利用できるビデオ通話サービスである．若い世代にはコンピュータゲームプレイ時のボイスチャットサービスとしても普及している．グループチャット機能や画面共有機能を持つため，遠隔で動作を確認しながらのサポートにも活用できる．Siv3D コミュニティでは，2020 年のパンデミック以降，オンラインの勉強会や実装会を開催する場合のプラットフォームとして活用機会が増えた．2021 年時点で 85 人が Siv3D のサーバに登録している．

Whereby　Web ブラウザ上で追加のアプリのインストールや，ホスト以外のアカウント登録が不要で使えるビデオ通話サービスである．複数人が同時に利用できる画面共有機能を持つ．2〜3 人が参加する小規模な会議で用いた．

GitHub Sponsors　ソースコードホスティングサービス GitHub 上で 2019 年に始まった，OSS 開発者を金銭的に支援できるサービスである．支援の対価として氏名の掲載や個別のユーザサポートを設定するケースが多い．Siv3D では，スポンサーの特典として，Web サイトへの名前の表示のほか，Slack のダイレクトメッセージによる質問の受け付け（通常は，質問や回答に含まれる情報をコミュニティと共有するために，ユーザからダイレクトメッセージを用いた問い合わせは受け付けていない），有料で販売しているサンプルへの無料アクセス，新機能への早期アクセスなどを提供している．GitHub Sponsors を開始して最初の 2 年間で，Siv3D は約 2,000 米ドルの資金を得た．

5.4 フィードバックの収集と対応

Siv3D は，公式なユーザサポートの場として，次の 4 つを設定している．

- コミュニティ全体に対して発言できるグループチャット (Slack および Discord)
- ソースコードや画像を添付して質問や意見を匿名で投稿できる電子掲示板 (BBS)
- GitHub 上のコードリポジトリの Issues
- SNS 上でのコミュニケーション (Twitter)

グループチャットへの投稿には，メールアドレスを用いたアカウントの登録が必要で，主に使用歴の長いコアユーザの議論の場として使われることが多い．BBS はアカウント登録が不要で，新規の質問ごとにスレッドが作成され，基本的に，質問→解決法の提示→質問者による解決の報告，という流れで 1 つのスレッドが完結する．Twitter は最も頻繁に要望や質問が投稿される場で，大抵の場合，投稿者との間で 1〜3 回のメッセージをやり取りすることで問題が解決する．2021 年までに BBS には約 500 件，グループチャットには約 15,000 件のメッセージが投稿され，質問に対してはほぼ 100% の回答率を維持している．回答方針として，現在の機能で解決できない問題があっても，思いつく限りの代替手法を提案することで，ユーザコミュニティに対して「質問しても回答がつかない，何も得られないのでは」という不安を感じさせないような工夫をしている．

BBS やグループチャット，SNS に投稿された Siv3D へのフィードバックのうち，Siv3D 自体に修正や改良が必要であると判断された問題については，筆者によって Siv3D のコードリポジトリの Issues 管理に追加され．本人に Issue として問題を管理することを伝達する．合わせて，問題の発見につながった SNS の投稿のリンクを Issue にコメントとして残し，後日問題が修正された際に，報告のあった SNS の投稿に返信をする形で解決を当人に伝達する取り組みを行っている．この方式により，SNS で不具合やリクエストを報告した当事者が，報告事象の解決の進捗状況をトラッキングしやすくなった．

GitHub 上のコードリポジトリでホスティングされている Siv3D は，不具合報告や要望のすべてを，リポジトリの Issues で管理できることが，情報集約と共同開発の観点から理想的であるが，Issues の投稿には GitHub アカウントの登録が必要であり，海外 Web サイトである GitHub の英語のユーザインタフェースや，Issues の書き方について詳しくない利用者などは，Issues の投稿に抵抗を感じるケースも少なくない．このハードルを緩和するため，Siv3D では過去に利用者によって投稿された Issues の中から，模範的な書き方のものをリファレンスで紹介したり，SNS で頻繁にフィードバックを報告してくれる利用者に対して Issues の利用を促したりするなどの取り組みを続けた．その結果，2021 年 11 月時点で，直近 1 年間に筆者以外によって投稿された Issues は 25 件と，過去最大の件数となった．

6　C++ およびツール開発者へのフィードバック

この章では，C++ の標準化や開発ツールに対するフィードバックとして，Siv3D での最新の C++ 仕様や開発ツールの活用事例と評価について説明する．また，Siv3D が抱える課題の中で，C++ の規格やツールの整備など，Siv3D 単体では解決が難しいものを挙げ，将来のフレームワークの品質向上，開発促進につながる目標の提言を行う．

6.1　C++17, C++20 機能に対するフィードバック

Siv3D は最新の C++ 言語仕様の活用を特長とし，Siv3D がビルドに要求するコンパイラは，Siv3D の該当リリースの直近 1〜2 年内に登場したような新しいコンパイラに限っている．そのため，数多くの C++17 や C++20 機能が実用されており，API 設計における最新 C++ 規格の活用に関する評価や知見が蓄積されている．本節では，C++17 および C++20 の機能の中で，特に Siv3D のコードの改善に恩恵が大きかったものを紹介する．

6.1.1　コンストラクタへの nodiscard 属性

[[nodiscard]]属性は，関数宣言やクラスの宣言の中で指定することにより，関数の戻り値が無視された場合にコンパイラに警告を発生させる，C++17 で導入されたコア言語機能である．当初の C++17 規格には，[[nodiscard]]属性をコンストラクタに付与できるという文面が無かった．Siv3D では，一時オブジェクトを作成したのちにメンバ関数で必要な操作を行うというパターンを多用するため，次のコード例のように，オブジェクトを作成しただけで何もしない，誤ったコードをしばしば書いてしまうことがある．このコードはコンストラクタの呼び出しを行うため，単純な未使用変数とは異なり，どのようなコンパイラにおいても[[nodiscard]]属性を利用して警告を発生させることができないという問題があった．

```
1  void Main()
2  {
3    // 無意味であり，未使用変数として警告される
4    int i = 123;
5
6    // 無意味だが，コンストラクタへの[[nodiscard]] 属性指定が可能になるまでは警告されなかった
7    Rect{ 100, 50, 40, 30 };
8
9    Rect{ 100, 50, 40, 30 }.draw();
10 }
```

C++20（および C++17 への遡及適用）[78] によって，コンストラクタへの [[nodiscard]]属性指定が可能になったことを受け，Siv3D では原則としてすべてのクラスのコンストラクタに [[nodiscard]]属性を指定した．その結果，Siv3D の API の範囲では，戻り値の無視に対するあらゆるケースにおいて，コンパイラが警告メッセージを発生させられるようになり，誤ったコードを書きにくくなるという，使い勝手の向上が得られた．

6.1.2 Designated Initialization

C++20 から，集成体 (aggregates) に対してメンバ変数名を指定した初期化 (designated initialization) が可能になった [73]．これまでは，メンバ変数の多い構造体を初期化する場合，集成体初期化で一斉にパラメータを記述するか，初期化後に名前を使って値を代入するかの二択であった．前者は同じ型の値を取り違えても気付かない，後者は構造体を const にできず，また，一部のメンバ変数の初期化を忘れる可能性があり，誤ったコードを書きやすい問題があったが，designated initialization により解決された．

```
struct BoundingBox
{
  double xMin = Math::Inf;
  double yMin = Math::Inf;
  double xMax = -Math::Inf;
  double yMax = -Math::Inf;
};

// 通常の集成体初期化:
// どの値がどのメンバ変数にセットされるか，この局所的なコードだけではわからない
BoundingBox A()
{
  return{ 0.1, 0.2, 0.3, 0.4 };
}

// あとで値を代入:
// 値をセットし忘れたメンバ変数があっても気付かない
BoundingBox B()
{
  BoundingBox b;
  b.xMin = 0.1; b.yMin = 0.2;
  b.xMax = 0.3; b.yMax = 0.4;
  return b;
}

// Designated Initialization
BoundingBox C()
{
  return{
    .xMin = 0.1, .yMin = 0.2,
    .xMax = 0.3, .yMax = 0.4
  };
}
```

実際の Siv3D のコードを抜粋し，具体的に効果があった事例を紹介する．次の 2 つのコードは BMP 画像ファイルのヘッダを作成する関数について，designated initialization の導入前と導入後のコードを示している．designated initialization の導入により，説明的なコードへと改善されていることがわかる．

```
// Designated Initialization 導入前のコード
static constexpr BMPHeader Make(int32 width, int32 height, uint32 size_bytes) noexcept
{
  return{
    0x4d42,
    static_cast<uint32>(sizeof(BMPHeader) + size_bytes),
    0, 0, sizeof(BMPHeader), 40,
    width, height,
    1, 24, 0,
    size_bytes,
    0, 0, 0, 0
  };
}
```

```
// Designated Initialization 導入後のコード
static constexpr BMPHeader Make(int32 width, int32 height, uint32 size_bytes) noexcept
{
  return
  {
    .bfType = 0x4d42,
    .bfSize = static_cast<uint32>(sizeof(BMPHeader) + size_bytes),
    .bfReserved1 = 0,
    .bfReserved2 = 0,
    .bfOffBits = sizeof(BMPHeader),
    .biSize = 40,
    .biWidth = width,
    .biHeight = height,
    .biPlanes = 1,
    .biBitCount = 24,
    .biCompression = 0,
    .biSizeImage = size_bytes,
    .biXPelsPerMeter = 0,
    .biYPelsPerMeter = 0,
    .biClrUsed = 0,
    .biClrImportant = 0
  };
}
```

6.1.3 Concepts

C++20 で Concepts が導入されるまで，テンプレートの型に制約を加えるには，SFINAE[18] と呼ばれる複雑なイディオムを用いる必要があった．Siv3D では Concepts の登場以前においても，型制約を用いたオーバーロードが必須でない単純なテンプレートであっても，ユーザが誤った型を引数に渡した際に生じる難解なエラーメッセージを回避するために SFINAE を用いていたが，Concepts によりそれらのコードを単純なコードに置き換えることができた．2021 年時点では，Concepts をサポートするコンパイラとそうでないコンパイ

ラが混在するため，Siv3D では次のようなマクロを定義している．

```
// Concepts をサポートするコンパイラであるかを判定する機能テストマクロ
#if __cpp_lib_concepts

// Concepts をサポートするコンパイラ用
# define SIV3D_CONCEPT_INTEGRAL template <std::integral Integral>
# define SIV3D_CONCEPT_INTEGRAL_ CONCEPT_INTEGRAL

#else

// Concepts をサポートしないコンパイラ用．SFINAE イディオムを使用
# define SIV3D_CONCEPT_INTEGRAL template <class Integral, std::enable_if_t<std::
    is_integral_v<Integral>>* = nullptr>
# define SIV3D_CONCEPT_INTEGRAL_ template <class Integral, std::enable_if_t<std::
    is_integral_v<Integral>>*>

#endif

//
// 使用例
//

SIV3D_CONCEPT_INTEGRAL
auto Square(Integral n);

// 末尾に_ がつくマクロは，SFINAE 版におけるデフォルトテンプレート引数の再定義回避のため
SIV3D_CONCEPT_INTEGRAL_
auto Square(Integral n) { return n * n; }

int main()
{
  // コンパイルエラー，
  // 比較的読みやすいエラーメッセージが出力される
  Square(0.5);

  Square(10);
}
```

SIV3D_CONCEPT_*は単純なマクロであるが，フレームワーク内での登場回数は 120 ファイル，700 回を超える．この方法の制約として，複数のテンプレート引数に対応すると，型の組み合わせ数が膨大になり，必要なマクロが複雑になる，したがって，複数のテンプレート引数が登場する場面では，単純に機能テストマクロ__cpp_lib_conceptsを用いて，2 通りのコードを記述した．

6.2 Siv3D の開発を支えた C++ ツール

2000 年代後半以降，C++ は他のプログラミング言語と比較して，開発支援ツールの発展が見劣りする状況であったが，近年は新たなツールの登場により状況が改善している [83]．大規模な C++ プロジェクトである Siv3D においても，一般的な IDE (Visual Studio や Xcode) の使用に加え，C++ コーディングの生産性を高めるための様々なツールを活用した．それらの機能概要と Siv3D の開発における活用状況を紹介する．

Wandbox[105]　数十種類以上の C++ コンパイラによるコンパイルと，コードの実行結果を確認できるオンラインコンパイラである．コンパイラごとの C++ コードの解釈の差異の検証と，エラーや警告メッセージの確認のために使用した．

Compiler Explorer[24]　様々なバージョンのコンパイラやコンパイルオプション設定で C++ コードをコンパイルし，そのアセンブリ出力を確認できるオンラインコンパイラである．抽象度の高い C++ コードを書く際に，各コンパイラでオーバーヘッドが生じないかを確認するために使用した．

Quick C++ Benchmark[103]　複数の異なるコードの実行時間を相対的に比較して可視化する，ベンチマーク機能を備えたオンラインコンパイラである．バックエンドに Google Benchmark[28] を使用し，Google Benchmark の API である，コード中の未使用変数の最適化を抑止する命令などを記述できる．Siv3D のコードに近似計算やアルゴリズムの変更を導入する際，性能への影響を測定するプロファイリングのために使用した．

PVS-Studio[63]　C++ や C#, Java を対象とした静的コード解析ツールである．実行時性能に影響するコードや，ガイドラインに違反するコード，冗長なコード，タイプミス，バグが疑われる箇所等を検出できる．商用ソフトウェアであるが，オープンソース開発者向けの無料ライセンスを提供している (2021 年現在)．

ツールの活用事例として，Siv3D のコードの問題が静的解析ツール PVS-Studio によって検出された場面を 2 つ紹介する．次のコードでは，`add()`の第一引数でムーブされているオブジェクトの use-after-move が，第二引数の`_fmt()`内で発生している．

```
m_fonts.add(std::move(font), U"({0} {1} size: {2})"_fmt(
    font->getFamilyName(), font->getStyleName(), fontSize));
```

次の例は目視で気付きにくいタイプミスである．3 行目の `std::min()`関数のリストの最初の要素は `bezier.p0.y`が正しい．PVS-Studio により，疑わしいコードとして検出された．

```
const double minX = std::min({ bezier.p0.x, bezier.p1.x, bezier.p2.x });
const double maxX = std::max({ bezier.p0.x, bezier.p1.x, bezier.p2.x });
const double minY = std::min({ bezier.p0.x, bezier.p1.y, bezier.p2.y });
const double maxY = std::max({ bezier.p0.y, bezier.p1.y, bezier.p2.y });
```

このような C++ のエコシステムは，Siv3D のマルチプラットフォーム向け開発と，コードの不具合の発見に役立った．C++ 標準はツールに関するガイドラインを提供しないため，ここで紹介したような非標準のツールは，C++ プログラミングの入門書などで言及される例はほとんど目にしない．しかし，C++ の難しさや間違いやすさを緩和する有用性は確かであり，より適切な評価を受け，注目される機会を得られるようにすべきだと考える．

6.3 将来の C++ への提言

Siv3D における C++ API 設計の経験の中で，将来のフレームワークの品質向上，開発促進に影響すると筆者が考える C++ 言語の課題について，C++ 規格策定プロセスへのリクエストという形で 3 項目を提言する．

6.3.1 移植性のある乱数分布，シャッフル

現在の C++20 の規格 [8] では，標準ライブラリの乱数分布や範囲要素のシャッフル操作に使われる具体的なアルゴリズムが規定されていないため，`std::uniform_int_distribution`や `std::shuffle()`などの内部アルゴリズムが実装によって異なる．そのため，疑似乱数生成エンジンに同じシードを与え，エンジンからの乱数出力の再現性を確保したとしても，乱数分布やシャッフル操作を通過することで，最終的に得られる乱数が環境によって一致せず，プログラムの移植性が妨げられる問題がある．わかりやすい例として，標準の C++ では，次のような単純なコードでさえ，実装によって実行結果が異なることを確認できる (表 7).

```
# include <iostream>
# include <random>
# include <cstdint>

int main()
{
  std::mt19937 rng{ 12345 };
  std::uniform_int_distribution<std::int32_t> ud{ 0, 10000 };
  // 0 以上10000 以下の無作為な整数を出力
  std::cout << ud(rng) << '\n';
}
```

Siv3D では，このように一貫性の無い乱数分布とシャッフル関数については，標準ライブラリには依存せず，各プラットフォームで共通となる代替実装を用意した．乱数分布については専門的な数学知識が要求されるため，自前での実装は行わず，ヘッダオンリーで必要な機能がそろう Abseil C++ library[27] を用いた．

本来この問題は標準ライブラリの範疇で解決できれば C++ 開発者にとって便利であるが，一方で，規格において実装が明示的に規定されていなかったことで，内部実装が，規格策定の数年後に提案された高速なアルゴリズムに置き換えられたという事例 [40][41] もあるため，一貫性の無さを一概に否定することはできない．そこで，C++ 標準ライブラリにおいて，内部アルゴリズムが変更される可能性のある API と並行して，一貫した結果を保証するような API が提供されるようになれば，マルチプラットフォーム開発における労力の削減につながるはずだ．

表 7　各実装における，サンプルコード実行時の標準出力

実装	出力される整数
gcc 11.1.0 + libstdc++	9297
clang 12.0.1 + libc++	4578
MSVC 2022 + MSVC STL	1463

6.3.2 数学関数の constexpr 対応

3D 空間における平面の法線，光源の方向などの表現に使う 3 次元ベクトルについて，軸に平行でないベクトルを単位ベクトル化してコンパイル時定数にしようとする際，ベクトル正規化の計算で使われる C++ 標準ライブラリの平方根関数 `std::sqrt()`は constexpr でないため，コンパイル時定数化が阻まれる．

```
// コンパイルエラー
constexpr Vec3 v = Vec3{ 1.0, 1.0, 1.0 }.normalized();
```

この制約により，Siv3D の 3D シーンにおけるデフォルトの平行光源（太陽）の方向ベクトル定数は，ライブラリ内で次のように計算後の数値で記述される．

```
namespace s3d::Graphics3D
{
  inline constexpr Vec3 DefaultSunDirection{ 0.4082482904638631, 0.4082482904638631,
      -0.8164965809277261 }; // Vec3{ 1, 1, -2 }.normalized()
}
```

これは `Vec3(1, 1, -2).normalized()`の計算結果であるが，コメント無しでは根拠が不明な定数に見えてしまうという問題がある．

また，標準ディスプレイの色表示をエミュレートする sRGB 変換 [109] を行うコードでは，浮動小数点数の累乗を計算するが， 累乗を計算する C++ 標準ライブラリの `std::pow()`関数も constexpr でないため，プログラマにとってなじみのある sRGB 色空間の色定数から，リニア空間の色定数へ変換するコードについて，コンパイル時定数化が阻まれる．

```
double RemoveSRGBCurve(double x)
{
  return ((x < 0.04045) ? (x / 12.92) : std::pow((x + 0.055) / 1.055, 2.4));
}

double ApplySRGBCurve(double x)
{
  return ((x < 0.0031308) ? (12.92 * x) : (1.055 * std::pow(x, (1.0 / 2.4)) - 0.055));
}
```

そのため，Siv3D のライブラリコードでは，リニア色空間向けのカラーパレットの値を定数にするために，sRGB 色空間における色定数とは別に，リニア空間への変換後の数値リテラルを記述する手間が発生している．

```
// リニア色空間のための色定数
namespace s3d::Linear::Palette
{
  ...
  inline constexpr ColorF Mediumpurple{ 0.29177, 0.16203, 0.70838 };
  inline constexpr ColorF Slateblue{ 0.14413, 0.10224, 0.61050 };
  inline constexpr ColorF Mediumslateblue{ 0.19807, 0.13843, 0.85499 };
}
```

数学関数が constexpr に対応していれば，次のように sRGB 色空間の色定数からの計算結果を用いて，冗長な数値の記述を省略できる．

```
// リニア色空間のための色定数
namespace s3d::Linear::Palette
{
  ...
  inline constexpr ColorF Mediumpurple = s3d::Palette::Mediumpurple.removeSRGBCurve();
  inline constexpr ColorF Slateblue = s3d::Palette::Slateblue.removeSRGBCurve();
  inline constexpr ColorF Mediumslateblue = s3d::Palette::Mediumslateblue.removeSRGBCurve();
}
```

このように，3DCG では平方根や累乗などの数学関数が多用されることから，関連する標準ライブラリの関数を constexpr 化できれば，ベクトルや色に関するより多くの計算をコンパイル時に評価し，実行時性能の向上やエラーの検出，コードの記述性の向上につなげることができる．

将来の C++ に向けて，標準ライブラリの数学関数の constexpr 化が提案されているが，副作用の問題や浮動小数点数の丸めモードの扱いなどの解決すべき課題があり，まずはそれらの影響を受けないごく一部の数学関数から constexpr 化を進めている段階 [70] であり，コード例に登場した `std::sqrt()`, `std::pow()`関数はその対象となっていない．数学関数のコンパイル時計算対応は，3DCG に関わるプログラムの記述性を向上するうえで重要度の高い要素であるため，改善されることが望ましい．

6.3.3 標準ライブラリの Unicode サポート強化

第 3 章で述べたように，Siv3D では UTF-32 を標準の文字コードとして採用しているが，C++ 標準ライブラリにおける文字列関連の API は，`std::basic_string`や `std::basic_string_view`など一部を除き，`char`型，よくても `wchar_t`型までのサポートしか保証されていない．標準ライブラリ API のクラステンプレート引数にこれら以外の型 (`char16_t`, `char32_t`) を指定すると，コンパイルに問題が生じる場合がほとんどである．標準ライブラリにおける文字列関連の関数の Unicode サポート状況を表 8 にまとめた．

C++11 で `char16_t`および `char32_t`型がコア言語機能に入り，C++20 でそれらの UTF-16 および UTF-32 規格への適合性が保証 [21] されるなど，C++ 言語における Unicode サポートの改善は少しずつ進んでいるが，現在でもなお UTF-16 や UTF-32 でエンコードされたテキストデータに対する，移植性のある文字列処理は，標準ライブラリに頼らず自前で用意する必要があり，C++ フレームワークにおける UTF-16 や UTF-32 のサポートをの難易度を上げる要因となっている．C++ には，こうしたギャップの解消を求めたい．

6.4 ツール開発者への提言

Siv3D での経験を踏まえ，将来のフレームワークの品質向上，開発促進に効果が高いと筆者が考える技術課題の中で，筆者個人の力だけでは取り組むことが難しく，世界の誰かに解決を求めたい課題について，提言の形で 3 項目をまとめた．

6.4.1 絵文字データの高圧縮な保存形式の開発

第 3 章で述べたように，Siv3D ではプラットフォーム間におけるテキストレンダリング結果の一貫性のために，ビルド後のアプリケーション実行ファイルに，カラー絵文字の TrueType フォントファイルを含んで

表 8　C++20 標準ライブラリの文字列処理 API における char16_t, char32_t サポート状況

API	wchar_t サポート	char16_t, char32_t サポート
標準入出力	○	
`std::basic_string`	○	○
`std::basic_string_view`	○	○
`std::hash`	○	○
`std::basic_regex`	○	
`std::basic_stringstream`	○	
`std::basic_fstream`	○	
`std::format`	○	
`std::to_string`	○	
`std::stoi`	○	
`std::to_chars`		
`std::from_chars`		

表 9　Unicode Emoji 13.0 以降をサポートする代表的な絵文字フォントのサイズ

フォント	ファイルサイズ	Zstandard 1.5.1 圧縮後のサイズ (圧縮設定=最大圧縮)	オープンソース ライセンスであるか
Noto Emoji (Unicode 14.0)	9.4 MB	8.6 MB	○
Twemoji (Emoji 13.1)	1.4 MB	0.61 MB	○
OpenMoji 13.1	15 MB	1.6 MB	○
Segoe UI Emoji (Windows 11)	1.4 MB	1.2 MB	

いる．このフォントファイルには，現在のバージョンで約 3,700 種類の絵文字が収録され，サイズは 9.4MB（Zstandard 圧縮は後 8.6 MB）と，アプリケーションを構成するファイル群の中でも大きな割合を占める．Unicode で規格化される絵文字 (Unicode Emoji) は，近年は 100〜300 文字前後/年のペースで追加されているため [104]，今後もファイルサイズがかさむことが見込まれる．アプリケーション内で使用する絵文字の種類が限られる場合は，サブセットフォントに置き換えることでサイズを削減できる．また，省サイズ指向で簡素なデザインのフォント (表 9) を選択することで，サイズを 1.4MB（Zstandard 圧縮後は 0.61MB）まで抑えることができる．しかし依然として，数十キロバイト単位でのサイズ削減に取り組んでいる Web 向けの Siv3D では無視できないデータサイズである．絵文字フォントが要求するデータサイズは，Siv3D に限らず，絵文字フォントを利用するような Web サービスのページ表示速度にも関わる，広範で一般的な問題である．

カラー絵文字フォントのデータ形式にはいくつかの種類があり，Noto Emoji のように使用色数が多く精細な絵文字は PNG 形式で圧縮したビットマップフォントを格納する CBDT/CBLC 形式が使われ，Segoe UI Emoji のように色数が少なく形状が単純なものは，単色のアウトライングリフを複数組み合わせる COLR/CPAL 形式が使われる．これ以外に，SVG テーブルを利用するものもある．いずれにせよ，画像の精

細さとファイルサイズはトレードオフである．ファイルサイズの問題を緩和するためには，カラー絵文字を現在よりも小さいサイズで保存できるベクター画像形式の開発が必要である．既存のフォーマットを置き換えることができそうな形式として，IconVG[30] がアクティブに開発されている．IconVG を用いることで，サイズのかさむ CBDT/CBLC や SVG でしか格納できなかった，色のグラデーションを含むような複雑な絵文字形状を，より小さいサイズで表現できることが期待される．ただし，IconVG がすぐに標準化され，一般的なフォントフォーマットに導入されるような見通しは立っていないため，IconVG を絵文字データの記録形式として採用したいアプリケーションは，当面は独自のフォーマットを構築する必要があるだろう．こうしたニーズに対して，何らかのツールや，場合によっては IconVG の競合を提供できると役立つだろう．

6.4.2 様々な種類のメディアファイルへの適合性ベンチマーク

画像や音声，動画などの様々な形式のメディアファイルをフレームワークで効率的に扱うために，フレームワーク開発者はデータのエンコード，デコードの処理を独自に記述したり，サードパーティ・ライブラリに任せたりするが，実装されたエンコーダ・デコーダが，対象とするメディアフォーマットの規格にどれだけ準拠しているか，また脆弱性のない実装になっているかを，自前ですべて検証することは困難である．そこで，各種フォーマットへの対応の検証に使える，あらゆる形式のテストファイルやバリデータが集約されているデータベースが整備されると便利である，例えば JPEG ファイルであれば，画像の大きさが最小のもの，最大のもの，インターレース形式，Exif データを含むもの，破損したものなど，ユーザ環境で登場しうるあらゆる種類の検証用ファイルが用意され，あるプログラムがそれらを適切に扱えるかを網羅的に検証できるシステムである．こうしたものを，画像以外も含む数百種類以上のファイル形式についてカバーし，開発者がアクセスしやすい形で提供されると，Siv3D のように，様々なファイル形式を扱うライブラリ開発者の助けになるだけでなく，個別のフォーマットに関して，性能を向上させるような新しいエンコーダやデコーダの開発も促進され，世の中のソフトウェアの実行時性能や安全性の向上に寄与するだろう．

6.4.3 言い換え表現に強いドキュメント検索

Siv3D の利用者を長年にわたって観察した経験則として，利用者が問題解決のために，フレームワークの機能や操作についてドキュメントやライブラリヘッダ，Web でキーワード検索をする際，本文に含まれる用語とは異なる言い換え表現を検索キーワードに選んでしまうことで，本来正しい用語を使うことで得られたはずの情報にアクセスできないケースがしばしば見受けられた．Siv3D プログラミングで頻繁に使われる用語のうち，言い換え表現が存在するものを，表 10 に具体例としてまとめた．

こうした言い換え表現のリストは，フレームワークの開発者であれば経験に基づいて列挙できるため，あらかじめ用語リストを検索データベースに登録できるようなドキュメント生成ソフトウェアや，IDE 向けの検索エンジンが整備されると，利用者の体験が向上するだろう．また，IDE の提供する検索機能に自然言語処理を導入することも，将来の IDE の姿として考えられる．関連する事例として，Microsoft 社が公開するソースコードエディタ Visual Studio Code は，エディタ自体の設定に関する JSON ファイルを，同社の Bing 検索エンジンを通して自然言語で検索できる機能を提供しており [46]，その構成が参考になるだろう．

表 10　Siv3D に登場する用語と，その言い換え例

用語	言い換え例
描画	表示，描く，出す，出力
音声	オーディオ，サウンド，音楽，BGM (background music), 効果音，SE (sound effect)
透過	半透明，透ける
座標	位置，場所，地点
動画	ビデオ, ムービー
テキスト	文字列，文章，メッセージ
フレームレート	FPS (frames per second)

7　結論

本研究では，プログラミング言語 C++ に着目し，C++ が持つ豊富なライブラリやコードベースなどの蓄積を生かしつつ，プログラミングに要求されるスキルを下げる工夫を行うことによって，C++ を用いた高度な情報処理へのアプローチを容易にするための新たな手法を考案し，それを実現するために「Siv3D」と呼ぶプログラミングフレームワークの提案を行った．本論文では，Siv3D を構築するアイデア，コミュニティ，プロセスを文書化し，重要な要素や技術的ポイントについて説明した．主に議論された Siv3D の方法論は次の通りである．

- C++ API 設計におけるコードの表現力の拡張
  - より高度な抽象化と，冗長・曖昧さの排除
  - コンパイル時におけるエラー検出の強化
- プログラミングに直接関連しない要素
  - フレームワークへのアセット同梱による利便性向上
  - 開発参加を促進するユーザコミュニティ育成

本論文では，第 1 に，フレームワークが提供する機能セットや実装手法について，Siv3D で取り組んだ課題を中心に整理を行い，新規のフレームワーク開発者が参照できるガイドラインを提示した．第 2 に，Siv3D における C++ API 設計について議論を行い，独自実装である名前付き引数のエミュレーションによる関数オーバーロードの拡張や，ユーザ定義リテラルの活用，`bool`型の strong typedef などによる C++ コードの表現力向上などの工夫が，使い勝手の良い C++ API の実現につながったことを示した．第 3 に，Siv3D で取り組んだユーザコミュニティ運営や普及のための活動を振り返り，その評価を報告した．第 4 に，Siv3D における最新の C++ の仕様や開発ツールの活用事例について議論し，Siv3D のようなフレームワークの発展につながるフィードバックと提言を行った．

本研究成果や，それが反映されたフレームワーク Siv3D を用いることで，C++ アプリケーションのプログラムを従来よりも短く明快に記述できるようになり，C++ をアプリケーションの開発言語として採用できるシナリオが拡大する．

Siv3D のソースコードはオープンソースライセンスで公開されている．Siv3D の SDK のダウンロード数は年間 1 万回に及び，Siv3D を活用したプログラムは，ソースコード共有サイト上において 1 万ファイル超が共有されている．実験や可視化用のソフトウェア開発に Siv3D を活用した研究成果も 12 報発表されている．こうした実績が，Siv3D が現実のソフトウェア開発に貢献していることを示している．

Siv3D 本体を構成するソースコードは，独自に実装した部分だけで 2,200 ファイル，22 万行に及び，これ以外に 90 のサードパーティ・ソフトウェアが追加で組み込まれている．本研究成果は，登場して間もない C++17/C++20 の最先端の言語仕様を効果的に活用しつつ，大規模で競争力のあるフレームワークを設計するという先駆的な取り組みであることから，とくに C++ のフレームワーク開発者にとって有用な知見集であり，将来のフレームワークの品質向上と，アプリケーション開発の発展に寄与するものである．

7.1 本研究の貢献

7.1.1 表現工学分野への貢献

本研究における表現工学分野への貢献は以下の 2 点である．

研究開発を支援するツールセットを提供

HCI 研究やデジタルメディア表現研究のためのソフトウェア開発に活用できる C++ プログラミングフレームワーク Siv3D を開発し，オープンソースライセンスで公開した．これまでに実験や可視化用のソフトウェア開発に Siv3D を活用した学術論文が少なくとも 12 報発表されている．また，Siv3D により提供される機能を組み合わせることで，新しいメディア表現の開発が容易に達成されることを，実際に稼働したアプリケーションの事例を通して示した．本論文ではとくに，オープンソースの絵文字フォントとスクリプティング機能を活用した教育用ビジュアルプログラミング言語と，各種通信機能によるハードウェア・ソフトウェア連係や 3D グラフィックスの表示機能を活用した，音源定位機能を有するペットロボットについて述べた．

情報可視化やインタラクションを扱うオープンソースソフトウェアの発展に役立つ知見を共有

デジタルコンテンツ制作の高度化にともない，プログラミングフレームワークの進化は今後も要求されていくだろう．これまで無かったプログラミング技術やハードウェアの進化を取り入れた新規のフレームワークが今後も提案され続けることが望ましいが，個人開発でオープンソースのフレームワークを開発し，ユーザコミュニティや協働体制を構築・発展させることは難しく，この分野に挑戦して活躍できるプレイヤーは限られていた．本研究では将来のフレームワーク開発者がより少ないコストで Siv3D と同等以上のフレームワークを開発できるよう，Siv3D の設計決定と，開発省力化に関する情報を報告した．また，個人のオープンソースソフトウェア開発者が参考にできる，Siv3D のユーザコミュニティに関する 3 つの施策「勉強会」「実装会」「チャレンジ」の実施手順と効果を説明し，オープンソースソフトウェアのコミュニティ運営を改善するための知見を共有した．

7.1.2 プログラミング分野への貢献

本研究におけるプログラミング分野への貢献は以下の 2 点である．

大規模フレームワーク開発における C++ の使い勝手を改善する技法を提案，実証

現在最も普及しているプログラミング言語の 1 つである C++ について，情報可視化や人と計算機のインタラクションに関する大規模フレームワークを開発する過程で顕在化した，言語の使い勝手に関する課題を複数指摘し，言語仕様の制約の中でそれらの課題の解決もしくは緩和を図る独自の工夫に基づく API 設計を提案した．これらのアイデアは，長年にわたるフレームワーク運用を通して不整合などの問題が生じないことを実証した．ソースコードは開発者が容易に再利用できるよう，オープンソースライセンスで公開した．

新しい C++ 開発環境へのフィードバックを提示

プログラミング言語の発展のためには，言語仕様のアップデートや開発ツールの改善とともに，それらに対する利用者視点からの検証が必要である．本研究では，直近に策定された C++ 規格や，新しい C++ ツールなど，開発環境の進化を先取的にプロジェクトで採用・実践し，言語仕様策定プロセスや，将来の C++ アプ

リケーション開発者が参照できるよう，それらの活用事例と評価を取りまとめた．

7.2 今後の課題と展望

最後に，これからの Siv3D の開発にあたって，取り組みたい課題と興味深い研究テーマを紹介する．

7.2.1 より新しい C++ 規格を活用する API 設計の研究

C++20 で規格化されたモジュール機能 [69] は，ヘッダファイルのインクルードを，よりクリーンで効率的な仕組みに置き換える機能である，2021 年時点で主要な C++ コンパイラにおけるモジュールのサポートは不十分であり [17]，Siv3D を含む，既存の主要なフレームワークにおいてモジュールを導入した事例は見つけられない．ここ数年の C++ の新機能と比較しても，モジュールの導入は広範囲のコードの更新が要求されることから，普及は迅速には進まないだろう．また，ヘッダファイルからの完全な脱却のためには，現在規格化に向けた議論が続いている，標準ライブラリのモジュール対応 [13][84] も欠かせない．大規模なフレームワークにおいて，部分的もしくは全面的にモジュールを導入することでビルド性能がどう変化するかは C++ プログラマの関心事となっているが，コンパイラによるモジュールサポートが未成熟である現在は適切な評価を下すことが難しい．環境が整い次第，速やかに検証して，ビルドのベンチマークやモジュールの構成法に関するフィードバックをコミュニティに提供したい．

C++26 での導入が見込まれる契約プログラミング [68][39] は，ソースコード中のある変数の値について，指定した時点における条件を宣言し，それを満たさないケースをバグであるとツールに指示できる機能である．ツールによるバグの検出と，バグを検出した際の挙動の予測可能化に役立つことが期待される．Siv3D の C++ API 設計において契約プログラミングが有効なケースの 1 つに，引数の値の範囲に制約がある関数での利用が挙げられる．ここでは画像クラスのメンバ関数 `.saveJPEG()` を例に，契約プログラミングの機能をコードの比較により説明する．メンバ関数 `.saveJPEG()` における引数 `quality` は，画像を JPEG エンコードする際の品質を設定する値で，0 以上 100 以下であることが要求される．次のコード例では，条件がドキュメントに記述されているが，コンパイラや静的解析ツールはこの情報を理解できないため，条件に沿わないコードを記述したとしても，ツールはそれをバグとして検出しない．

```
class Image
{
public:
  /// @brief 画像をJPEG 形式でファイルに保存します．
  /// @param path 保存するファイルパス
  /// @param quality JPEG 圧縮後の品質．[0，100] の範囲
  /// @return 保存に成功した場合true，それ以外の場合はfalse
  bool saveJPEG(FilePathView path, int32 quality) const;
};
```

現時点での契約プログラミングの規格案 [39] に基づく C++ コードは，次のとおり関数宣言の末尾に事前条件を記述するものである．この情報をツールが解釈し，バグの検出や最適化のヒントとして活用する．例えば利用者が `.saveJPEG()` に整数リテラルで 1000 のような範囲外の値を渡したときに，コンパイルエラーとして検出され，バグを減らせることが期待される．

```
class Image
```

```
{
public:
  /// @brief 画像をJPEG 形式でファイルに保存します．
  /// @param path 保存するファイルパス
  /// @param quality JPEG 圧縮後の品質．[0, 100] の範囲
  /// @return 保存に成功した場合true，それ以外の場合はfalse
  bool saveJPEG(FilePathView path, int32 quality) const
  [[pre: 0 < path.size()]] // precondition
  [[pre: 0 <= quality]] // precondition
  [[pre: quality <= 100]]; // precondition
};
```

C++ コミュニティが，モジュールや契約プログラミングなどの新機能を大規模なフレームワークで活用するプラクティスを確立する過程において，Siv3D が先駆的な事例を示して貢献したい．

7.2.2 C++ から Web アプリケーションを開発する手法の洗練と共有

コンピューティング利用者の所有デバイスの多様化にともない，従来デスクトップアプリケーションとして提供されていたアプリケーションが，Web アプリケーションに移植されるケースが増えている [54][61]．最終的な出力を Web アプリケーションとする C++ プロジェクトの数は今後ますます増えていくだろう．Siv3D は 2020 年から Web 対応 (図 34) が始まったが，Web 対応における課題は，ビルド後に出力される，アプリケーションの実行に必要なファイルのサイズであった．Web アプリケーションはネットワークを利用し，Web ブラウザを用いて通常の Web ページのようにアクセスされるため，ファイルサイズがユーザ体験に影響を及ぼしやすい．当初の Siv3D Web 版は，デスクトップ版をビルドするためコードやアセットを，ほぼそのまま転用しただけであり，ごく簡単なプログラムでも，アクセス時に数十 MB 規模のファイルダウンロードを要求した．現在では，不要なファイルの分離やビルド構成・設定の見直しを重ねたことで，当初より機能を向上させる一方で，必要なファイルサイズを数 MB 程度までに圧縮した．ファイルサイズ削減のための改良は現在も進行中であり，最終的には最小サイズを 1〜2MB にすることを目標としている．C++ フレームワークにおける，Web アプリケーション向けのサイズの圧縮について，Siv3D の Web 版開発の過程で得られた知見を整理・共有することは，将来の C++ 開発の活性化に寄与するだろう．

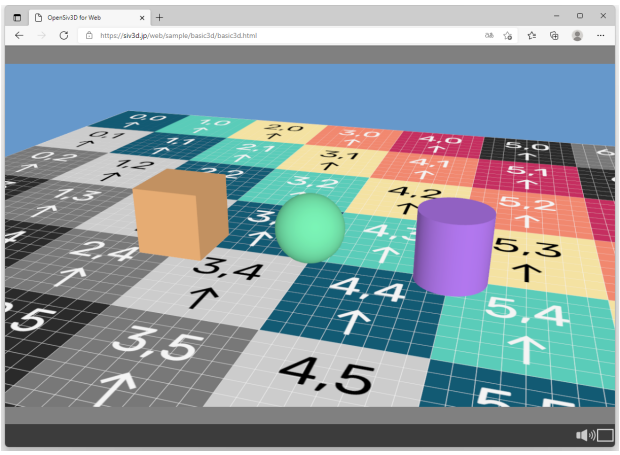

図 34　Web ブラウザ上で実行される Siv3D プログラムのスクリーンショット

7.2.3 Siv3D のサブセットライブラリの提供

Siv3D を利用するアプリケーションプログラムは，プログラムのエントリポイントのほか，サブシステムの初期化や状態更新，連係に関する処理をフレームワーク側が担う．グラフィックスやオーディオなど，抽象度の高い API であるほど複数のサブシステムに依存するため，利用者は Siv3D が提供する機能の一部だけを切り出して再利用することが難しい．現在，Siv3D を使っていない既存のプロジェクトにおいて，Siv3D の一部の機能だけを使おうとする場合，Siv3D がエントリポイントを管理するプログラムのほうに既存のコードを持ち込むという主従関係が要求される．この制約は，Siv3D のコード資産が多くのプロジェクトで活用される機会を明らかに狭めている．

したがって，Siv3D の一部と同等の機能を提供する自己完結型のサブセットライブラリを提供できることが望ましい．サブセットライブラリの提供により，Siv3D のコード資産が利用しやすくなり本研究成果が広く普及するだけでなく，コードリポジトリの規模がメンテナンスしやすい小さな単位になることでライブラリの検証や改良が促進されることが期待される．それらの成果を逆輸入することで，フレームワーク本体の進化にもつながる．現在の Siv3D を構成するサブシステム群の中で，他のサブシステムと疎結合でありスタンドアローン化しやすく，また利用者の需要が見込まれる機能として次が挙げられる．

- 標準ライブラリを拡張した配列クラス
- 標準ライブラリを拡張した文字列クラス
- 二次元配列クラス
- 画像処理クラス
- ファイル I/O クラス
- 線形代数クラス
- ノイズ生成クラス
- 乱数生成クラス

これらのうち，ノイズ生成クラスと乱数生成クラスについては，すでにサブセットライブラリ [91][92] を公開し，Siv3D 本体はそれらに依存する構成となっている．それぞれのサブセットライブラリは独立して一定の評価 (GitHub のスター数はそれぞれ 382 個, 93 個) を獲得しており，利用者からの Issues(不具合報告や要望) およびコードの改善案の議論は Siv3D 本体から独立して進められている．結果として，Siv3D 本体を利用しない開発者が Siv3D の進化に寄与するような仕組みが実現している．将来的には Siv3D 本体から 10〜20 個のサブセットライブラリを独立させる計画である．

サブセットライブラリの設計にあたって注意する点は，C++ の代表的なライブラリである Boost[5] のようなサブセットライブラリ間の依存を避けることである．Boost は，各種ライブラリのリポジトリが独立して管理され，コードリポジトリの規模を分割することには成功しているが，インクルードによるライブラリ同士の依存関係が大きく，利用者は Boost の提供する大半のライブラリをスタンドアローンで利用できない設計となっている．Boost の一部の機能を利用するために，1 万を超えるファイルから構成されるライブラリを導入する必要があることは，Boost が敬遠される一因となっている．Siv3D は各種サブセットライブラリの自己完結化を徹底し，Boost のような問題を回避する．

7.2.4 機械学習によるコード自動補完技術への適応

近年の機械学習や言語モデルの技術進化を踏まえると，大量のパブリックコードやドキュメントを学習データとして用いることで，プログラマが記述したコメントやコードから，それに続くコードを予測して提案できるようになることは容易に想像可能である．実際に，Microsoft 社の IDE である Visual Studio に搭載されたコーディング支援機能 IntelliCode[52](2019 年発表) や，コード用にファインチューニングした GPT-3 モデル OpenAI Codex[10] をベースとする GitHub Copilot[23](2021 年発表) のように，製品または技術プレビューとして一般に公開される事例も登場している．2021 年時点の GitHub Copilot が提案する C++ コードについて筆者が検証した範囲では，これに頼るだけで実用レベルのアプリケーションを構築できるレベルには達していないが，提案される数行〜数十単位のコードをそのまま，または手直しして利用するような対話的なコーディングを行うことで，プログラムを書く際のタイピング量を大幅に削減する体験ができた．

GitHub Copilot は GitHub に保存されているパブリックコードをベースにしている．したがって，2.5 で述べたように GitHub に一定量のパブリックコードを有する Siv3D に関する知識も持ち合わせており，例えば Siv3D で図形や色を表現するクラスやそのメンバの情報，標準で用いる文字列リテラルが UTF-32 であること，描画やウィンドウ操作に用いる API など，様々な知識を提案コードから読み取ることができた．実例を紹介すると，次のコードを上から順に記述し，19 行目にコメントを記述したところ，20 行目から 30 行目までのひと固まりのコードが，コメントも含め提案された．これは筆者から見ても適切な Siv3D プログラムである．また，それより前の 9 行目と 12 行目，15 行目も，一文字もタイプせずに，コメントに基づいた提案を採用して得られたコードである．検証時点で GitHub 上に「Siv3D」と「パネル」の両方の語を含むようなコードは存在しないことから，これは既存のコードのコピーではなく，複数の知識を統合して新しく生み出されたコードであることがわかる．

```
1  # include <Siv3D.hpp>
2
3  void Main()
4  {
5    // パネルの色
6    const ColorF panelColor{ 0.8, 0.9, 1.0 };
7
8    // パネルのサイズ
9    const Size panelSize{ 100, 100 };
10
11   // パネルの間隔
12   const int32 panelMargin = 10;
13
14   // 最初のパネルの位置
15   const Point firstPanelPos{ panelMargin, panelMargin };
16
17   while (System::Update())
18   {
19     // パネルを4x3 枚表示
20     for (int32 y = 0; y < 3; ++y)
21     {
```

```
22      for (int32 x = 0; x < 4; ++x)
23      {
24        // パネルの位置
25        const Point panelPos = firstPanelPos + Point(x * (panelSize.x + panelMargin), y * (
              panelSize.y + panelMargin));
26
27        // パネルを描画
28        Rect(panelPos, panelSize).draw(panelColor);
29      }
30    }
31  }
32 }
```

筆者の印象では，GitHub Copilot は少なくとも Siv3D のチュートリアル [90] を一通り読み，いくつか練習でコードを書いたことのある，C++ に関して 1 年以上勉強した人間と同等以上のアウトプットをすると感じられた．一方で，様々なケースを検証した結果，システムの提案コードにおける課題として，存在しないメンバの使用や，コンパイルエラーを引き起こすような型の整合性の無視，古いライブラリバージョンの API の使用などが見受けられた．これらは，ユーザ環境における C++ コンパイラの知識との連係により改善できるだろう．

こうしたコーディング支援技術が，プログラミングにおける生産性向上の根幹要素となるのは時間の問題であり，ソフトウェア開発者がフレームワークやプログラミング言語を選択する際には，コード自動補完技術との適合性が重要な指標となるだろう．すると，今後のフレームワーク開発者は，コード自動補完で良いパフォーマンスを得るために何をすべきかという課題について関心を払う必要が出てくる．利用人口の大きいフレームワークは学習データが多く集まる点で有利かもしれないし，逆に利用者によるバッドノウハウや削除された古い API の情報がモデルに含まれることで理想的な結果を得にくくなるかもしれない．まったく新しいフレームワークであっても，体系的に記述されたドキュメントと一定水準のサンプルプログラムを備えていれば，先行者を超える優れたモデルを生成できるかもしれない．さらに将来を見据えると，アプリケーションコードの大部分が AI により自動生成されることを前提とした API 設計についての議論も求められるだろう．コード自動補完・生成技術は，従来の API 設計における通例を陳腐化させる可能性もあるだろうし，人間にとって使いやすい API がいつの時代にも高いパフォーマンスを発揮する可能性も考えられる．少なくとも現時点でフレームワーク開発者にできる対策は，学習のデータベースとなるパブリックコードの質と量を向上させることであり，品質の高いコードを利用者が共有することを促進したり，開発者自らが整備したりすることで備えることになる．

コード自動補完技術は登場したばかりの技術であり，GitHub Copilot もプロプライエタリなサービスであることから，現時点でフレームワーク開発者にとっての何らかのプラクティスを導くのは早計だろう．継続して既存・新規のサービスや関連研究 [33] の動向を追っていきたい．

謝辞

本研究プロジェクトの遂行にあたって，指導とアドバイス，そして奔放な私の研究活動をさまざまな形でサポートしてくださった，早稲田大学の長幾朗先生，坂井滋和先生，上田和紀先生，河合隆史先生，タリン工科大学の Dr. Sven Nõmm, 早稲田大学実体情報学博士プログラムの関係者各位に感謝いたします．

また，学部時代から長年にわたり Siv3D に有益なフィードバックを提供してくれた高橋卓人さん，増田健太さん，研究室で博士課程を共に過ごし，多大な協力をしてくれた竹之内要人さんに感謝いたします．

Siv3D のこれまでの発展は，コードを共に開発した 32 名のコミッタと 22 名のスポンサー（2022 年 1 月時点）に支えられています．ありがとうございます．

本研究の大部分は「早稲田大学実体情報学プログラム」の支援を受けました．Siv3D の開発は情報処理推進機構（IPA）による「未踏 IT 人材発掘・育成事業」の支援を受けました．Siv3D を活用したビジュアルプログラミング言語 Siv3D for Kids の開発は公益財団法人 I-O DATA 財団の支援を受けました．Siv3D を活用したビジュアルプログラミング言語 Enrect の開発は科学技術振興機構の「戦略的創造研究推進事業（ACT-I）」の支援を受けました．

実体情報学コースの修了にあたって

筆者は大学院修士課程以降において，副専攻相当のコースとして早稲田大学博士課程教育リーディングプログラム「実体情報学博士プログラム」に参加しました．このコースでは，専攻学科の境界を超えた学生や教員，研究者との交流，専門分野とは異なる機械系の講義や活動の体験，留学サポート，研究費および毎月の奨励金支給など，大学院生向けとしては非常に手厚い支援を受けて研究活動に取り組むことができました．「実体情報学 (Embodiment Informatics)」はこのコースのスローガンであり，コースの卒業生は学位論文に各々の「実体情報学とは何か」を振り返りとして書くことが伝統となっているため，筆者もそれを記します．

人間の身体は，生活における快の増加や負荷の軽減のために，実世界環境からの何らかの作用や，機械からのフィードバックを求めます．大昔の人間は道具や機械を工作して操作することでそれらを実現していましたが，やがて人間よりも強力なエネルギー源（家畜や自然エネルギー，燃料）を利活用するようになり，さらに人体の情報処理能力がボトルネックとなると，人間がアルゴリズムを記述し，それに基づいた機械操作や機構設計，情報収集，データ処理を，計算機が自律的に行う様式へと進化してきました．そのようにして，人間は高度で複雑な実世界環境への干渉と機械からのフィードバックを実現できるようになりました．図 35 はその流れを示しています．

実体情報学は，この図の中にある「矢印」に注目し，矢印に相当する要素技術やデザインを最適化することを通して，人類が合理的かつ効率的な労力で，適切なベネフィットやフィードバックを得られるようにすることを追究する「connection」の学問であると考えます．実体情報学コースでは，情報と機械の融合を意識した学習プログラムと研究設備の提供が行われ，各々の修了生が，こうした矢印にマッチするような研究に取り組み，成果を発表してきました．本論文の研究もこの図においては，プログラミングとフィードバックの矢印を強化する部分に相当します．

21 世紀の人類の挑戦として，自動運転車や気候変動対策，食糧問題，教育や医療の機会の平等などスケールの大きな課題が挙がります．どれも単独の学問領域で解決できる問題ではなく，あらゆる領域の connection の強化が必要であることは当然認識されています．実体情報学コース発の成果が，こうしたニーズの高い領域で注目され，社会に貢献できることを修了生の一員として願いつつ，今後の研究開発の邁進に励む所存です．

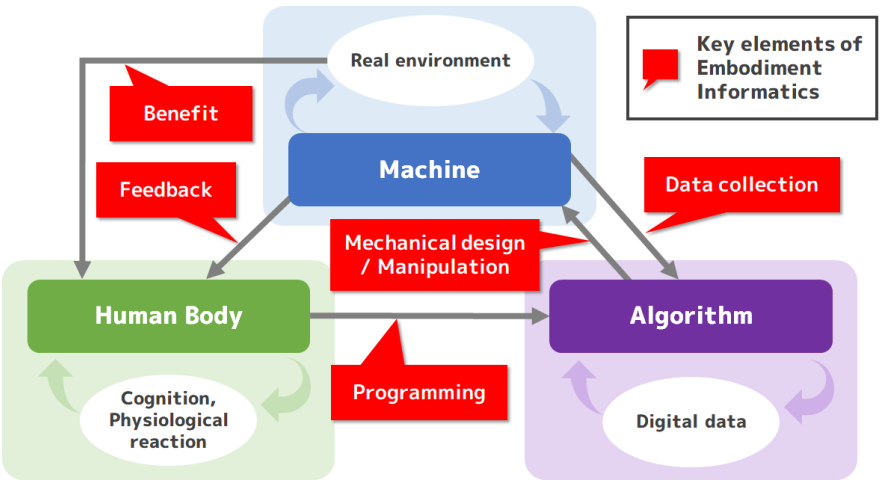

図 35　人体とアルゴリズム，機械をつなぐ矢印が実体情報学の主要素

付録 A　Siv3D を構成するオープンソースのソフトウェア一覧

Siv3D のライブラリコード内で使用される，サードパーティ・ソフトウェアと，そのライセンス（SPDX short identifier による表記．独自ライセンスである場合はベースになっているライセンス）を，Siv3D コードリポジトリの資料 [76] をもとに表 11 にまとめた．

表 11: Siv3D を構成するサードパーティ・ソフトウェア

名称	機能概要	ライセンス
Abseil	標準ライブラリの拡張	Apache-2.0
AngelCode Scripting Library	スクリプト言語	Zlib
asio	ネットワークライブラリ	BSL-1.0
Anti-Grain Geometry	2D 図形のソフトウェアレンダラー	BSD-3-Clause
Boost	様々なライブラリ集	BSL-1.0
Box2D	2D 物理エンジン	MIT
Catch2	テストフレームワーク	BSL-1.0
cereal	シリアライゼーション	BSD-3-Clause
concaveman-cpp	凹包の計算	BSD-2-Clause
CoreRT	ユーティリティ	MIT
cpptoml	TOML ファイルのパーサ	MIT
cpu_features	CPU の命令セットの取得	Apache-2.0
DirectXTK	Windows アプリ開発補助	MIT
DirectXMath	SIMD 対応の数学ライブラリ	MIT
DirectX Mesh Geometry Library	3D メッシュデータの処理	MIT
double-conversion	浮動小数点数の文字列化	BSD-3-Clause
Earcut	ポリゴンの三角形分割	ISC
Easing Equations	イージング式	MIT/BSD-3-Clause
EnumBitmask	C++ ユーティリティ	CC0-1.0
easyexif	EXIF データパーサ	BSD 2-Clause
fmt	文字列フォーマット	MIT
Font Awesome Free	アイコン画像のコレクション	OFL-1.1
FreeType	フォントのレンダリング	FTL
gif_load	GIF 画像デコーダ	Unlicense
GIFLIB	GIF 画像エンコーダ	MIT
GLEW	OpenGL 拡張	BSD-3-Clause/MIT
GLFW	ウィンドウ処理	Zlib
GMGSx.sf2	サウンドフォント	Unlicense
GSL	C++ ユーティリティ	MIT

表は次ページに続く

前ページからの続き

名称	機能概要	ライセンス
HarfBuzz	フォントのレンダリング	MIT 風独自
infoware	システム情報の取得	CC0-1.0
kld-polynomial	図形の交差判定	BSD-3-Clause
levenshtein-sse	レーベンシュタイン距離の計算	MIT
libcurl	URL 転送ライブラリ	MIT 風独自
libjpeg-turbo	JPEG デコーダ・エンコーダ	IJG/BSD-3-Clause/Zlib
libogg	Ogg デコーダ・エンコーダ	BSD-3-Clause
libpng	PNG デコーダ・エンコーダ	libpng-2.0
libtiff	TIFF デコーダ・エンコーダ	libtiff
libvorbis	Vorbis デコーダ・エンコーダ	BSD-3-Clause
libwebp	WebP デコーダ・エンコーダ	BSD-3-Clause
Lua	スクリプト言語	MIT
lunasvg	SVG パーサ	MIT
M+ Fonts	日本語のフリーフォント	独自
Material Design Icons	アイコン画像のコレクション	Apache-2.0
miniutf	Unicode 変換処理	MIT
minizip	ZIP アーカイブの処理	Zlib
msdfgen	フォントの SDF データ生成	MIT
muparser	数式パーサ	BSD-2-Clause
nanoflann	kd 木ライブラリ	BSD-2-Clause
Native File Dialog Extended	ファイルダイアログ	Zlib
nlohmann/json	JSON データ処理	MIT
Noto CJK	CJK 対応のフリーフォント	OFL-1.1
Noto Emoji	絵文字フォント	Apache-2.0
Obfuscate	埋め込み文字列の難読化	Unlicense
Oniguruma	正規表現処理	BSD-2-Clause
OpenCV	画像加工とコンピュータビジョン	Apache-2.0
Opus	Opus デコーダ・エンコーダ	BSD-3-Clause
opusfile	Opus ファイルの処理	BSD-3-Clause
par_shapes	3D 形状の生成	MIT
pffft	高速フーリエ変換	BSD 風独自
plutovg	2D 図形のソフトウェアレンダラー	MIT
qr-code-generator-library	QR コードの生成	MIT
Quirc	QR コードの検出	ISC
Recast & Detour	ナビメッシュによる経路探索	Zlib
rectpack2D	長方形詰め込み	MIT
RtAudio	音声入力の処理	MIT 風独自

表は次ページに続く

前ページからの続き

名称	機能概要	ライセンス
scope-guard	C++ ユーティリティ	MIT
sdf-glyph-foundry	フォントの SDF データ生成	BSD-2-Clause
Serial Communication Library	シリアル通信	MIT
SFMT	SIMD 使用の疑似乱数生成エンジン	BSD-3-Clause
simde	SIMD 使用コードの移植性向上	MIT
Sol2	Lua スクリプトの補助	MIT
SoLoud	オーディオエンジン	Zlib
SoundTouch	音声波形のピッチシフト	LGPL-2.1
SplineLib	スプライン曲線の計算	Unlicense
Star Nest	星空のレンダラー	MIT
stb_truetype	TrueType フォントの読み込み	Unlicense
stduuid	UUID の生成	MIT
The Parallel Hashmap	高速なハッシュテーブル	Apache-2.0
tinycolormap	カラーマップ	MIT
tinyobjloader	WaveFront .obj ファイルのパーサ	MIT
TinySoundFont	サウンドフォントのレンダラー	MIT
TinyXML-2	XML パーサ	Zlib
Wicked Engine	3D レンダラー	MIT
WinToast	Windows におけるトースト通知	MIT
Xoshiro-cpp	疑似乱数生成エンジン	MIT
xxHash	高速なハッシュ値計算	BSD-2-Clause
zlib	データ圧縮	Zlib
Zstandard	高性能なデータ圧縮	BSD-3-Clause/GPL-2.0-only

表はこれで終わり

付録 B　C++ 名前付き引数ライブラリの実装

本論文 4.1 で説明された，C++ での名前付き引数エミュレーションの実装コードを次に示す．コンパイルには C++20 に対応したコンパイラが必要である．最新のコードは Siv3D のコードリポジトリ内のヘッダ NamedParameter.hpp から入手できる．

```
1  // NamedParameter.hpp
2  //-----------------------------------------------
3  //
4  // This file is part of the Siv3D Engine.
5  //
6  // Copyright (c) 2008-2021 Ryo Suzuki
7  // Copyright (c) 2016-2021 OpenSiv3D Project
8  //
9  // Licensed under the MIT License.
10 //
11 //-----------------------------------------------
12
13 # include <functional>
14 # include <iostream>
15 # include <memory>
16 # include <tuple>
17 # include <type_traits>
18 # include <utility>
19
20 namespace s3d
21 {
22   template <class Tag, class Type>
23   class NamedParameter
24   {
25   public:
26
27     [[nodiscard]]
28     NamedParameter() = default;
29
30     [[nodiscard]]
31     constexpr NamedParameter(const Type& value)
32       : m_value(value) {}
33
34     template <class U, class V = Type, std::enable_if_t<std::is_convertible_v<U, V>>* =
           nullptr>
35     [[nodiscard]]
36     constexpr NamedParameter(const NamedParameter<Tag, U>& other)
37       : m_value(other.value()) {}
38
```

```
39    template <class... Args, class V = Type, std::enable_if_t<std::is_constructible_v<V,
          Args...>>* = nullptr>
40    [[nodiscard]]
41    constexpr NamedParameter(const NamedParameter<Tag, std::tuple<Args...>>& tuple)
42      : m_value(std::make_from_tuple<Type>(tuple.value())) {}
43
44    [[nodiscard]]
45    constexpr const Type* operator ->() const noexcept
46    {
47      return std::addressof(m_value);
48    }
49
50    [[nodiscard]]
51    constexpr const Type& operator *() const noexcept
52    {
53      return m_value;
54    }
55
56    [[nodiscard]]
57    constexpr const Type& value() const noexcept
58    {
59      return m_value;
60    }
61
62  private:
63
64    Type m_value;
65  };
66
67  template <class Tag, class Type>
68  class NamedParameter<Tag, Type&>
69  {
70  public:
71
72    [[nodiscard]]
73    NamedParameter() = default;
74
75    [[nodiscard]]
76    constexpr NamedParameter(Type& value) noexcept
77      : m_ref(std::addressof(value)) {}
78
79    [[nodiscard]]
80    constexpr Type* operator ->() const noexcept
81    {
82      return m_ref;
83    }
```

```
84
85    [[nodiscard]]
86    constexpr Type& operator *() const noexcept
87    {
88      return *m_ref;
89    }
90
91    [[nodiscard]]
92    constexpr Type& value() const noexcept
93    {
94      return *m_ref;
95    }
96
97  private:
98
99    Type* m_ref = nullptr;
100 };
101
102 template <class Tag>
103 struct NamedParameterHelper
104 {
105   template <class Type>
106   using named_argument_type = NamedParameter<Tag, Type>;
107
108   template <class Type>
109   [[nodiscard]]
110   constexpr NamedParameter<Tag, std::decay_t<Type>> operator =(Type&& value) const
111   {
112     return NamedParameter<Tag, std::decay_t<Type>>(std::forward<Type>(value));
113   }
114
115   template <class... Args>
116   [[nodiscard]]
117   constexpr NamedParameter<Tag, std::tuple<std::decay_t<Args >...>> operator() (Args
          &&... args) const
118   {
119     return NamedParameter<Tag, std::tuple<std::decay_t<Args >...>>(std::make_tuple(std::
          forward<Args>(args)...));
120   }
121
122   template <class Type>
123   [[nodiscard]]
124   constexpr NamedParameter<Tag, Type&> operator =(std::reference_wrapper<Type> value)
          const
125   {
126     return NamedParameter<Tag, Type&>(value.get());
```

```
    }

    template <class Type>
    [[nodiscard]]
    constexpr NamedParameter<Tag, Type&> operator()(std::reference_wrapper<Type> value)
        const
    {
      return NamedParameter<Tag, Type&>(value.get());
    }
  };
}

# define SIV3D_NAMED_PARAMETER(name) \
inline constexpr auto name = s3d::NamedParameterHelper<struct name##_tag>{};\
template <class Type> using name##_ = s3d::NamedParameter<struct name##_tag, Type>
```

次にサンプルプログラムを示す.

(参考) オンラインコンパイラによる実行結果: https://godbolt.org/z/a71Gond8K

```
# include <iostream>
# include "NamedParameter.hpp"

namespace Arg
{
  SIV3D_NAMED_PARAMETER(month);
  SIV3D_NAMED_PARAMETER(day);
}

void Date_impl(int month, int day)
{
  std::cout << "month: " << month << " day: " << day << '\n';
}

void Date(Arg::month_<int> m, Arg::day_<int> d)
{
  Date_impl(m.value(), d.value());
}

void Date(Arg::day_<int> d, Arg::month_<int> m)
{
  Date_impl(m.value(), d.value());
}

int main()
{
  Date(Arg::month = 12, Arg::day = 31);
```

```
28   Date(Arg::day = 31, Arg::month = 12);
29 }
```

参考文献

[1] Abdelghany, M.: Is it time for operator overloading in Java?, Jun 2020. https://blogs.oracle.com/javamagazine/post/is-it-time-for-operator-overloading-in-java.

[2] Berrow, J.: 2D Graphics: A Brief Review, Technical report, December 2019. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P2005R0.

[3] bgfx: bgfx (code repository). https://github.com/bkaradzic/bgfx.

[4] Bilas, S.: An Automatic Singleton Utility, *Game Programming Gems. Mass: Charles River Media, Inc*, (2000).

[5] Boost: Boost C++ Libraries (website). https://www.boost.org/, (also at Internet Archive https://web.archive.org/web/20211017063118/https://www.boost.org/).

[6] Brown, E. W. and Sunderland, D.: Recommendations for Specifying "Hidden Friends", Technical report, March 2019. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P1601R0.

[7] Brown, J.: Labelled Parameters, Technical report, October 2018. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P1229R0.

[8] C++ Standards Committee: ISO International Standard ISO/IEC 14882: 2020, Programming Language C++, (2020).

[9] Card, M.: *Readings in information visualization: using vision to think*, Morgan Kaufmann, 1999.

[10] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code, *arXiv preprint arXiv:2107.03374*, (2021).

[11] Chlumský, V.: Shape decomposition for multi-channel distance fields, Master's thesis, Czech Technical University, 2015. https://github.com/Chlumsky/msdfgen/files/3050967/thesis.pdf.

[12] cinder: cinder (code repository). https://github.com/cinder/Cinder.

[13] Clow, M., Dawes, B., Reis, D. G., Lavavej, T. S., O' Neal, B., Stroustrup, B., and Wakely, J.: Minimal module support for the standard library, Technical report, February 2018. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P0581R1.

[14] Cocos2d-x: Cocos2d-x (code repository). https://github.com/cocos2d/cocos2d-x.

[15] Cohen, S.: Is Wordle too easy for you? Try it in Japanese. (online news article), *The Japan Times*. https://www.japantimes.co.jp/life/2022/02/02/language/wordle-easy-try-japanese/.

[16] Colubri, A. and Fry, B.: Introducing Processing 2.0, *ACM SIGGRAPH 2012 Talks*, SIGGRAPH '12, New York, NY, USA, ACM, 2012, pp. 12:1–12:1.

[17] cppreference.com: C++ compiler support (website). https://en.cppreference.com/w/cpp/compiler_support, (also at Internet Archive https://web.archive.org/web/20211120051146/https://en.cppreference.com/w/cpp/compiler_support).

[18] cppreference.com: SFINAE (website). https://en.cppreference.com/w/cpp/language/sfinae, (also at Internet Archive https://web.archive.org/web/20211024071552/https://en.cppreference.com/w/cpp/language/sfinae).

[19] Doxygen: Doxygen (website). https://www.doxygen.nl/index.html, (also at Internet Archive

https://web.archive.org/web/20211205182454/https://www.doxygen.nl/index.html).

[20] ebiten: Ebiten (code repository). https://github.com/hajimehoshi/ebiten.

[21] Fernandes, R. M.: Make char16_t/char32_t string literals be UTF-16/32, Technical report, February 2019. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P1041R4.

[22] GIGAZINE: 大小 2 種類のブロックくずしを 2 つ同時にプレイするブラウザゲーム「Brickception」は忙しすぎて脳が 2 つ欲しくなるレベル (online news article), *GIGAZINE*. https://gigazine.net/news/20190611-brickception/.

[23] GitHub: GitHub Copilot (website). https://copilot.github.com/, (also at Internet Archive https://web.archive.org/web/20211125164449/https://copilot.github.com/).

[24] Godbolt, M.: Compiler Explorer (website). https://godbolt.org/, (also at Internet Archive https://web.archive.org/web/20210803150815/https://godbolt.org/).

[25] Godot Engine: Godot Engine (code repository). https://github.com/godotengine/godot.

[26] Golodetz, S.: Simplifying the C++/AngelScript Binding Process, *Overload*, No. 95(2010).

[27] Google: Abseil C++ library (code repository). https://github.com/abseil/abseil-cpp.

[28] Google: Benchmark (code repository). https://github.com/google/benchmark.

[29] Google: Google Summer of Code Archive (website). https://summerofcode.withgoogle.com/archive/, (also at Internet Archive https://web.archive.org/web/20211105075626/https://summerofcode.withgoogle.com/archive/).

[30] Google: IconVG (code repository). https://github.com/google/iconvg.

[31] Green, C.: Improved alpha-tested magnification for vector textures and special effects, *ACM SIGGRAPH 2007 courses*, 2007, pp. 9–18.

[32] Gregory, J.: *Game engine architecture*, crc Press, 2018.

[33] Heyman, G., Huysegems, R., Justen, P., and Van Cutsem, T.: Natural language-guided programming, *Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2021, pp. 39–55.

[34] ImGui, D.: Dear ImGui (code repository). https://github.com/ocornut/imgui.

[35] Jonsson, A.: AngelScript. http://www.angelcode.com/angelscript/.

[36] Kato, A., Hirabayashi, M., Matsurnoto, Y., Nakashima, Y., Kobayashi, Y., Fujie, M. G., and Sugano, S.: Continuous wrist joint control using muscle deformation measured on forearm skin, *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 1818–1824.

[37] KATO, A., MATSUMOTO, Y., KATO, R., KOBAYASHI, Y., YOKOI, H., FUJIE, M. G., and SUGANO, S.: Estimating wrist joint angle with limited skin deformation information, *Journal of Biomechanical Science and Engineering*, Vol. 13, No. 2(2018), pp. 17–00596.

[38] Kato, T., Kato, A., Okamura, N., Kanai, T., Suzuki, R., and Shirai, Y.: Musasabi: 2D/3D intuitive and detailed visualization system for the forest, *ACM SIGGRAPH 2015 Posters*, 2015, pp. 1–1.

[39] Krzemieński, A., Berne, J., and McDougall, R.: Contract Support: Defining the Minimum Viable Feature Set, Technical report, November 2020. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P2182R1.

[40] Lemire, D.: Doubling the speed of std::uniform_int_distribution in the GNU

C++ library (libstdc++) (website). https://lemire.me/blog/2019/09/28/doubling-the-speed-of-stduniform_int_distribution-in-the-gnu-c-library/, (also at Internet Archive https://web.archive.org/web/20210511055643/https://lemire.me/blog/2019/09/28/doubling-the-speed-of-stduniform_int_distribution-in-the-gnu-c-library/).

[41] Lemire, D.: Fast random integer generation in an interval, *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, Vol. 29, No. 1(2019), pp. 1–12.

[42] libGDX: Community - libGDX (website). https://libgdx.com/community/.

[43] libGDX: libGDX (code repository). https://github.com/libgdx/libgdx.

[44] Liszewski, A.: Play This Devious Browser Game That Makes Pop-up Windows Infuriating in a Whole New Way (online news article), *GIZMODO*. https://gizmodo.com/play-this-devious-browser-game-that-makes-pop-up-window-1835382140.

[45] Llopis, N.: The Beauty of Weak References and Null Objects, *Game Programming Gems 4*, Kirmse, A.(ed.), Charles River Media, 2004, pp. 61–68.

[46] Lourens, R. and Karat, A.: Bing-powered settings search in VS Code. https://code.visualstudio.com/blogs/2018/04/25/bing-settings-search, (also at Internet Archive https://web.archive.org/web/20211123021658/https://code.visualstudio.com/blogs/2018/04/25/bing-settings-search).

[47] Lua: What is Lua? (website). https://www.lua.org/about.html, (also at Internet Archive https://web.archive.org/web/20210701000000*/https://www.lua.org/about.html).

[48] Maddock, J.: Boost.Multiprecision (website). https://www.boost.org/libs/multiprecision/).

[49] McLaughlin, M. B., Sutter, H., Zink, J., Davidson, G., and Michael, K.: A Proposal to Add 2D Graphics Rendering and Display to C++, Technical report, October 2019. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P0267R10.

[50] Meyers, S.: The most important design guideline?[User interfaces], *IEEE software*, Vol. 21, No. 4(2004), pp. 14–16.

[51] Microsoft: Keyboard shortcuts in Word (website). https://support.microsoft.com/en-us/topic/keyboard-shortcuts-in-word-95ef89dd-7142-4b50-afb2-f762f663ceb2, (also at Internet Archive https://web.archive.org/web/20211103075325/https://support.microsoft.com/en-us/topic/keyboard-shortcuts-in-word-95ef89dd-7142-4b50-afb2-f762f663ceb2).

[52] Microsoft: Visual Studio IntelliCode (website). https://visualstudio.microsoft.com/services/intellicode/, (also at Internet Archive https://web.archive.org/web/20211125053654/https://visualstudio.microsoft.com/services/intellicode/).

[53] Microsoft: XML Documentation. https://docs.microsoft.com/en-us/cpp/build/reference/xml-documentation-visual-cpp?view=msvc-160.

[54] Nattestad, T. and Al-Shamma, N.: Photoshop's journey to the web. https://web.dev/ps-on-the-web/, (also at Internet Archive https://web.archive.org/web/20211207221939/https://web.dev/ps-on-the-web/).

[55] Nielsen, J.: *Usability engineering*, Morgan Kaufmann, 1994.

[56] Nystrom, R.: *Game programming patterns*, Genever Benning, 2014.

[57] ogre: ogre (code repository). https://github.com/OGRECave/ogre.

[58] Okabe, M. and Ito, K.: Color Universal Design (CUD) - How to make figures and presentations that are friendly to Colorblind people -. https://jfly.uni-koeln.de/color/, (also at Internet Archive https://web.archive.org/web/20211128183741/https://jfly.uni-koeln.de/color/).

[59] Olsen, D., Burylov, I., and Dominiak, M.: Extended floating-point types and standard names, Technical report, September 2021. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P1467R5.

[60] openframeworks: openframeworks (code repository). https://github.com/openframeworks/openFrameworks.

[61] Pignotti, A.: WebVM: server-less x86 virtual machines in the browser. https://medium.com/leaningtech/webvm-client-side-x86-virtual-machines-in-the-browser-40a60170b361, (also at Internet Archive https://web.archive.org/web/20220201140107/https://medium.com/leaningtech/webvm-client-side-x86-virtual-machines-in-the-browser-40a60170b361).

[62] Pontin, J.: The Problem with Programming. https://www.technologyreview.com/s/406923/the-problem-with-programming/, (also at Internet Archive https://web.archive.org/web/20190825012646/https://www.technologyreview.com/s/406923/the-problem-with-programming/).

[63] PVS-Studio: PVS-Studio (website). https://pvs-studio.com/en/pvs-studio/, (also at Internet Archive https://web.archive.org/web/20210724220841/https://pvs-studio.com/en/pvs-studio/).

[64] pygame: pygame (code repository). https://github.com/pygame/pygame.

[65] Qiita: Qiita Advent Calendar について (website). https://help.qiita.com/ja/articles/qiita-adcal-1, (also at Internet Archive https://web.archive.org/web/20211101051246/https://help.qiita.com/ja/articles/qiita-adcal-1).

[66] Raman, N., Cao, M., Tsvetkov, Y., Kästner, C., and Vasilescu, B.: Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions, *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, 2020, pp. 57–60.

[67] Reddy, M.: *API Design for C++*, Elsevier, 2011.

[68] Reis, D. G., Garcia, D. J., Lakos, J., Meredith, A., Myers, N., and Stroustrup, B.: Support for contract based programming in C++, Technical report, June 2018. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P0542R5.

[69] Reis, D. G.: Working Draft, Extensions to C++ for Modules, Technical report, January 2018. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper N4720.

[70] Rosten, E. J. and Rosten, O. J.: constexpr for <cmath>and <cstdlib>, Technical report, June 2021. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P0533R8.

[71] SDL: SDL (website). https://www.libsdl.org/index.php.

[72] SFML: SFML (code repository). https://github.com/SFML/SFML.

[73] Shen, T. and Smith, R.: Designated Initialization Wording, Technical report, July 2017. ISO/IEC

JTC1/SC2/WG21: C++ Standards Committee paper P0329R4.

[74] Silva, J. O., Wiese, I., German, D. M., Treude, C., Gerosa, M. A., and Steinmacher, I.: Google summer of code: Student motivations and contributions, *Journal of Systems and Software*, Vol. 162(2020), pp. 110487.

[75] Siv3D: OpenSiv3D (code repository). https://github.com/Siv3D/OpenSiv3D.

[76] Siv3D: OpenSiv3D/ThirdParty.md. https://github.com/Siv3D/OpenSiv3D/blob/main/ThirdParty.md.

[77] Siv3D: Siv3D 勉強会 (website). https://siv3d.github.io/ja-jp/community/study-meeting/, (also at Internet Archive https://web.archive.org/web/20210123210121/https://siv3d.github.io/ja-jp/community/study-meeting/).

[78] Sommerlad, P.: [[nodiscard]] for constructors, Technical report, July 2019. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P1771R1.

[79] Sommerlad, P. and Sandoval, A. L.: Generic Scope Guard and RAII Wrapper for the Standard Library, Technical report, February 2019. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P0052R10.

[80] Standard C++ Foundation: Standardization (website). https://isocpp.org/std, (also at Internet Archive https://web.archive.org/web/20210801000000*/https://isocpp.org/std).

[81] Stroustrup, B.: C++ Applications (website). https://www.stroustrup.com/applications.html, (also at Internet Archive https://web.archive.org/web/20211030140629/https://www.stroustrup.com/applications.html).

[82] Stroustrup, B.: The Evolution of C++ - Past, Present, and Future (slide). https://github.com/CppCon/CppCon2016/tree/master/Keynotes/The%20Evolution%20of%20C%2B%2B%20-%20Past%2C%20Present%2C%20and%20Future.

[83] Stroustrup, B.: Thriving in a crowded and changing world: C++ 2006–2020, *Proceedings of the ACM on Programming Languages*, Vol. 4, No. HOPL(2020), pp. 1–168.

[84] Stroustrup, B.: Minimal module support for the standard library, Technical report, July 2021. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P2412R0.

[85] Stroustrup, B. and Sutter, H.: C++ Core Guidelines (website). https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines, (also at Internet Archive https://web.archive.org/web/20210803200203/https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines).

[86] Sutter, H. and Alexandrescu, A.: *C++ coding standards: 101 rules, guidelines, and best practices*, Pearson Education, 2004.

[87] Suzuki, R.: [C++] bool 型を強くする YesNo クラス (website). https://zenn.dev/reputeless/articles/cpp-article-yesno, (also at Internet Archive https://web.archive.org/web/20211222144402/https://zenn.dev/reputeless/articles/cpp-article-yesno).

[88] Suzuki, R.: OpenSiv3D Challenge 2021 (website). https://zenn.dev/reputeless/scraps/79865055750784, (also at Internet Archive https://web.archive.org/web/20210407140327/https://zenn.dev/reputeless/scraps/79865055750784).

[89] Suzuki, R.: Siv3D for Kids (website). https://siv3d-for-kids.github.io/, (also at Internet Archive https://web.archive.org/web/20201125060104/https://siv3d-for-kids.

github.io/).

[90] Suzuki, R.: Siv3D Reference v0.6.3. https://zenn.dev/reputeless/books/siv3d-documentation-en.

[91] Suzuki, R.: siv::PerlinNoise (code repository). https://github.com/Reputeless/PerlinNoise.

[92] Suzuki, R.: Xoshiro-cpp (code repository). https://github.com/Reputeless/Xoshiro-cpp.

[93] Suzuki, R.: コンピュータに習熟した先生がいなくても使える、「正解のない」プログラミング学習ツールの開発, 技術報告, 公益財団法人 I-O DATA 財団. 公益財団法人 I-O DATA 財団第 2 回成果報告書 https://0da32848-7670-49f0-a39a-e0ffac224ca7.filesusr.com/ugd/e3a272_b9d9fe34856f40cfb8cc16663efcd071.pdf.

[94] Suzuki, R.: リアルタイム C++ コーディングのための C++/AngelScript ハイブリッド開発・実行環境の提案, Master's thesis, 早稲田大学, 2016.

[95] Suzuki, R. and Choh, I.: C++14 における名前付き引数の実装, 情報処理学会第 *79* 回全国大会講演論文集, Vol. 2017(2017).

[96] Suzuki, R. and Choh, I.: Siv3D: インタラクティブアプリケーションのための C++ フレームワーク (サイバー増大号)–(特集 ソフトウェア論文), コンピュータソフトウェア= *Computer software*, Vol. 34, No. 4(2017), pp. 17–38.

[97] Suzuki, R. and Choh, I.: Using emoji as image resources in educational programming tools, *HCI International 2020 - Posters - 22nd International Conference, HCII 2020, Proceedings*, Stephanidis, C. and Antona, M.(eds.), Communications in Computer and Information Science, Springer, 2020, pp. 325–331.

[98] Suzuki, R., Kazunori, U., and Shigekazu, S.: 情報可視化やインタラクションのためのライブラリ Siv3D の機能強化と C++17, C++20 への対応, 日本ソフトウェア科学会第 *38* 回大会講演論文集, (2021).

[99] Suzuki, R., Takahashi, T., Masuda, K., and Choh, I.: Implementing Node-Link Interface into a Block-Based Visual Programming Language, *International Conference on Human-Computer Interaction*, Springer, 2018, pp. 455–465.

[100] Suzuki, R., Takahashi, T., and Okuno, H. G.: Development of a robotic pet using sound source localization with the hark robot audition system, *Journal of Robotics and Mechatronics*, Vol. 29, No. 1(2017), pp. 146–153.

[101] Takahashi, T., Okuno, H. G., Sugano, S., Coros, S., and Thomaszewski, B.: Computational Design of Balanced Open Link Planar Mechanisms with Counterweights from User Sketches, *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 6466–6471.

[102] three.js: three.js (code repository). https://github.com/mrdoob/three.js.

[103] Tingaud, F.: Quick C++ Benchmark (website). https://quick-bench.com/, (also at Internet Archive https://web.archive.org/web/20210620011638/https://quick-bench.com/).

[104] Unicode: Emoji Versions (website). https://unicode.org/emoji/charts/emoji-versions.html, (also at Internet Archive https://web.archive.org/web/20210729211342/https://unicode.org/emoji/charts/emoji-versions.html).

[105] Wandbox: Wandbox (website). https://wandbox.org/, (also at Internet Archive https://web.

archive.org/web/20210708011025/https://wandbox.org/).

[106] Zaitsev, A. and Polukhin, A.: C++ Numerics Work In Progress, Technical report, December 2019. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P1889R1.

[107] Zverovich, V.: std::format improvement, Technical report, February 2021. ISO/IEC JTC1/SC2/WG21: C++ Standards Committee paper P2216R3.

[108] 阿部法寛, 丸谷圭一, 青木広宙: LeapMotion を用いた疑似力触覚提示における脳波計測, 精密工学会学術講演会講演論文集 *2021* 年度精密工学会春季大会, 公益社団法人 精密工学会, 2021, pp. 683–684.

[109] 河村尚登, 杉浦博明: sRGB 色空間と国際標準化, 画像電子学会誌, Vol. 35, No. 6(2006), pp. 935–943.

[110] 吉館要, 深山覚, 後藤真孝, ほか: 既存楽曲の再生に合わせた即興演奏の支援システム, 第 *79* 回全国大会講演論文集, Vol. 2017, No. 1(2017), pp. 103–104.

[111] 権藤克彦, 明石修, 伊知地宏, 岩崎英哉, 河野健二, 豊田正史, 上田和紀: なぜソフトウェア論文を書くのは難しい (と感じる) のか, コンピュータソフトウェア, Vol. 26, No. 4(2009), pp. 4_17–4_29.

[112] 古屋翠, 速水治夫, ほか: コンサート会場の座席自動配置システムの提案, 研究報告グループウェアとネットワークサービス *(GN)*, Vol. 2016, No. 5(2016), pp. 1–6.

[113] 松田弘樹, 山地秀美: 特別支援教育における Kinect を用いたフィジカルトレーニング支援システムの提案, *IEICE Conferences Archives*, The Institute of Electronics, Information and Communication Engineers, 2016.

[114] 神展彦, 芳賀直樹, 藤代一成: マルチスレッド設計によるインタラクティブなグルーヴ可視化, 画像電子学会誌, Vol. 46, No. 1(2017), pp. 160–164.

[115] 早川雄登, 藤代一成, ほか: 直接操作による流体挙動の制御, 第 *78* 回全国大会講演論文集, Vol. 2016, No. 1(2016), pp. 185–186.

[116] 全国高等専門学校第 30 回プログラミングコンテスト実行委員会: 第 30 回都城大会（2019）競技部門パンフレット. https://www.procon.gr.jp/wp-content/uploads/2019/10/bc1804a6f66928358a6e44afd046c92a.pdf.

[117] 全国高等専門学校第 32 回プログラミングコンテスト実行委員会: 第 32 回秋田大会（2021）パンフレット. https://www.procon.gr.jp/?p=77939.

[118] 竹渕瑛一, 梶並知記, 徳弘一路, 速水治夫, ほか: 螺旋と極座標による音高系列の表示方法の提案, マルチメディア, 分散協調とモバイルシンポジウム *2017* 論文集, Vol. 2017(2017), pp. 1484–1487.

[119] 日本ソフトウェア科学会: 「ソフトウェア論文」について. https://www.jssst.or.jp/edit/detail/software_papers.html, (also at Internet Archive https://web.archive.org/web/20201101045802/https://www.jssst.or.jp/edit/detail/software_papers.html).

略語

(アルファベット順)

API　Application Programming Interface
DSL　Domain-Specific Language
GUI　Graphical User Interface
HCI　Human-Computer Interaction
HID　Human Interface Device
IDE　Integrated Development Environment
MIDI　Musical Instrument Digital Interface
MSDF　Multi-channel Signed Distance Field
RAII　Resource Acquisition Is Initialization
SDF　Signed Distance Field
SDK　Software Development Kit
SFINAE　Substitution Failure Is Not An Error
STL　Standard Template Library
ZWJ　Zero Width Joiner

著者発表リスト

本研究に関連する発表

論文誌

1. 鈴木遼, 長幾朗. Siv3D: インタラクティブアプリケーションのための C++ フレームワーク, コンピュータ ソフトウェア 34(4) pp. 17-38, 2017.（査読有り）（**ソフトウェア論文賞**）
2. Suzuki, Ryo, Takuto Takahashi, and Hiroshi G. Okuno. Development of a robotic pet using sound source localization with the hark robot audition system, Journal of Robotics and Mechatronics 29.1: pp. 146-153, 2017.（査読有り）

国際会議論文

3. Suzuki, Ryo, and Ikuro Choh. Using Emoji as Image Resources in Educational Programming Tools, International Conference on Human-Computer Interaction. Springer, Cham, pp. 325-331, 2020.（査読有り）
4. Suzuki, Ryo, Takuto Takahashi, Kenta Masuda, and Ikuro Choh. Implementing Node-Link Interface into a Block-Based Visual Programming Language, International Conference on Human-Computer Interaction. Springer, Cham, pp. 455-465, 2018.（査読有り）

解説記事

5. 鈴木遼. 音や画像で遊ぼう-インタラクティブアプリケーションための C++ フレームワーク「Siv3D」, 情報処理 58(6) pp. 474-480, 2017.（依頼有り）

口頭発表

6. 鈴木遼, 上田和紀, 坂井滋和. 情報可視化やインタラクションのためのライブラリ Siv3D の機能強化と C++17, C++20 への対応, 日本ソフトウェア科学会 第 38 回大会, 2021.（**高橋奨励賞**, **優秀発表賞**, **学生奨励賞**）
7. 鈴木遼, 松村哲郎, 安藤弘晃. ゲーム開発者のための C++11〜C++20, 将来の C++ の展望, CEDEC 2020, 2020.（査読有り）
8. 鈴木遼, 長幾朗. C++14 における名前付き引数の実装, 情報処理学会 第 79 回全国大会講演論文集 2017.1: pp. 201-202, 2017.
9. 鈴木遼. 音や画像で遊ぼう, 日本ソフトウェア科学会 第 33 回大会企画 FTD 2016, 2016.（招待有り）
10. 鈴木遼, 長幾朗. ゲームとインタラクティブメディアのプログラミングを容易にする C++ ライブラリの開発, 情報処理学会 第 77 回全国大会講演論文集 2015.1: pp. 97-98, 2015.（**学生奨励賞**）
11. 鈴木遼. メディアアート制作の敷居を下げる C++ プログラミングライブラリ, CEDEC 2014, 2014.（査読有り）

本研究に関連しない発表

書籍

12. 鈴木遼. 冒険で学ぶ はじめてのプログラミング, 技術評論社, 2018.

口頭発表

13. 鈴木遼, 曹暘. 視線情報を利用したテキストエリアの選択. 情報処理学会 第 78 回全国大会講演論文集, 2016.1: pp. 383-384, 2016.