

本書について

クラウドが、あらゆるものを飲み込んでいきます。

1990年、Tim Berners-Leeが最初のWebブラウザである「WorldWideWeb」をNeXT用に開発し、HTTPを提案したときから、ドキュメントを置くための理想的な場所は、手元のコンピューターではなく、サーバーに変わりました。ドキュメントをWebサーバーに置き、Webブラウザからインターネット接続して取得することで、ドキュメントの価値そのものが飛躍的に高まりました。

これが、ドキュメントがクラウドに飲み込まれた瞬間だと言えます。このときから、クラウドの膨張は、指数関数的に加速してきました。

2002年に登場した現Amazon Web Servicesは、2006年からストレージサービスのS3と、計算機そのものをネット経由で提供するコンピューティングのEC2という、2つの重要なサービスの提供を開始しました。これによって、メールサーバーやWebサーバーのマシンを買う必要がなくなりました。また、手元で使うコンピューター用のストレージも、コンピューターそのものがクラウドに飲み込まれたので、大きなディスクも、強力なPCも、買い求める必要がなくなっていました。

地球全体を覆う、海底光ファイバー網が猛烈な速度で拡張され、また、携帯電話網がデジタル通信の能力を飛躍的に高めたことで、クラウドは、あらゆるデータ、あらゆるアプリケーションを次々と飲み込んでいきます。

2010年には、AWSがEC2において、NVIDIA Fermi GPUを搭載したインスタンスタイプを追加しました。これによって、GPUを利用する3D CADやレンダラー、映像制作ソフトウェアなども、クラウドに飲み込まれました。

2016年頃からは、各種のFPGA(Field-programmable Gate Array)や、GoogleのTPU(Tensor Processing Unit)などの人工知能/機械学習向けの特別なプロセッサ、IBM QやGoogleのQuantum Computing Playgroundといった量子計算機なども、クラウドで利用可能になりました。

このように、クラウドは、あらゆるデータ、あらゆるアプリケーションを、次々と飲み込み続けてきましたが、ビデオゲームだけは、クラウドに飲み込まれることはありませんでした。

ビデオゲームは、他のさまざまなアプリケーションと比べても、毎秒60回、操作に反応して画面を更新したいという、非常に強い要求があります。この要求のために、これまでクラウドは、ゲームを飲み込むことができませんでした。

しかし、2018年現在、AWSだけをとりとってみても、地球上の18個の地点で高性能のデータセンターが利用可能で、ゲームをプレイできるほぼすべての人口

の生活位置からそれぞれ1000km以内をカバーし、極めて高速かつ低遅延な通信が可能になりました。ユーザーに十分近い位置に、クラウドサーバーを置くことが可能になったのです。光ファイバーの能力は、現在も、早いペースで増強され続けています。IPv6によって通信効率が改善し、また数年後には5Gネットワークによって、モバイル通信が現在の有線通信並みに高速化します。

筆者は、そのような高速通信の時代には、いよいよクラウドが、ビデオゲームを飲み込んでしまうのではないかと考えています。クラウドがビデオゲームを飲み込んだら、ゲームの利用者にとっては、ゲームを都度インストールしたり、アップデートする必要がなくなる上に、

- Webページを見るのと同じように、プレイしたいと思ったら、次の瞬間にプレイを始められる
- 途中でやめても、いつでも瞬間的にプレイを再開できる
- ゲームプレイの途中の状態友達に送る
- 誰かがプレイしている最中のゲームを検索して参加する

など、これまでのゲームにはなかった楽しみ方ができるようになるでしょう。

また、ゲームを作る開発者にとっては、各種のコンピューター向けにゲームを作る必要がなくなり、使い慣れたサーバーマシンで動くゲームを作れば、それを地球上の誰に対しても瞬間的に提供できるようになります。

PCやゲーム機、モバイル端末の機種ごとに移植をする必要がなくなれば、機種ごとに移植層を出力することが基本となっているゲームエンジンの設計も、変わるかもしれません。プログラミング言語やライブラリの構成にも、大きな影響を与えるでしょう。

マルチプレイゲームの実装方法についても、大きく変わることになります。従来、マルチプレイゲームを実装するためには、通信の遅延による問題が起きないように、非同期I/O、ソケット、RPCなどを駆使しながら、不正行為の余地が入り込まないように注意深くプログラミングをする必要がありましたが、その作業が不要になり、今までよりも多くの開発者がマルチプレイゲームの実装に参入できるようになるでしょう。

本書では、ネットワークにつながっているゲームの大きな魅力である、マルチプレイのクラウドゲームの実装について、実際にゲームを制作して評価をするなど、本格的に踏み込んだ解説を行っています。

本書は、クラウドが、まさにこれからゲームを飲み込むかもしれない……！というタイミングで書かれた、クラウドに飲み込まれた後のゲームのしくみと作り方についての本です。

筆者の前著『オンラインゲームを支える技術』は、2011年の時点ですでに確立していた、いわゆる「枯れた技術」を詳しく説明した本でしたが、本書は、これから起きることに焦点を当てた、かなり異なる方向性の本になりました。

読者の皆さんには、これから数年以内に起きるであろう、携帯電話の5Gネットワークをはじめとした通信技術、クラウド技術のさらなる進歩をイメージしながら、これからの時代に対応したゲーム開発を考えてみるきっかけにいただけたらうれしく思います。

2018年8月 中嶋 謙互

謝辞

本書は、すでに完成している技術ではなく、これから出現する新しい技術に関する議論に主軸を置いた野心的な書籍です。

本書におけるクラウドゲームの定義、分類、課題認識や基本的な設計思想については、筆者が2013年頃から参画していたシンラ・テクノロジーで、おもにビデオストリーム方式を採用したクラウドゲームの開発をしていたときの蓄積が基礎になっています。シンラ・テクノロジーはすでに解散してしまっていますが、Jacob Navok氏、岩崎 哲史氏、和田 洋一氏、そしてチームメイトたちには、技術的な挑戦と研究のチャンスをいただいたことに深く感謝します。また、このような野心的な企画を5年もの長い期間、消えないように温め続けてくれた編集の土井優子氏の尽力は本当に普通ではないと思い、感謝します。

最後に、休日の多くを執筆に費やしたりして子供と過ごす時間が減ったりしても、いつも機嫌良く応援してくれた妻と家族にも感謝します。

本書の構成

本書の章構成と、ブロックごとのテーマは、以下のようになっています。

第0章 [ゲーム開発者のための]レンダリング再入門

「ゲームプログラムはほとんど描画ばかりを行っている」

➡本書の内容を理解するための知識の準備をする

第1章 クラウドゲームとは何か

マルチプレイゲーム開発が変わる可能性が見えてきた(!?)

第2章 クラウドゲームのアーキテクチャ

実現したいゲーム×技術の適性を見定める

➡本書におけるクラウドゲームの範囲を押さえる

第3章 開発の道具立て

開発環境と2Dクラウドゲームライブラリとしてのmoyai

第4章 OpenGLを用いたローカルレンダリングのしくみ

moyaiの基礎部分。描画APIと内部動作

第5章 スプライトストリームのしくみ

リモートレンダリング方式概論、スプライトストリームの基本設計

第6章 スプライトストリームの通信プロトコル

TCPによる送受信、ストリームのなかみ、関数群

➡クラウドゲームの設計と実装を理解する

第7章 [クイックスタート]クラウドゲーム

多彩な軸のサンプルゲームから見えてくる要素

第8章 [本格実装]クラウドゲーム

ネットワークプログラミングなしで、MMOGの実現へ

➡実際にプレイできるゲームを動かしてクラウドゲーム開発を実体験する

本書で扱うクラウドゲームは、従来からあるビデオゲームにおける画面に描画するプログラムの部分を、クラウドを形づくるソケット通信の技術を前提として設計し直すことで可能になる、新しい設計思想を持つゲームです。

したがって、クラウドゲームを理解するためには、クラウドゲームではない、従来からあるビデオゲームがどのような技術を用いて画面を描画しているのかの基本的な考え方を理解して準備する必要があると言えます。

そのために、本書では第0章として、従来型のビデオゲームが画面をどうやって描画しているのかを、大まかに理解するための説明を行っています。第0章でゲームの描画に必要な数段階の工程を理解できれば、その工程がどのようにクラウドに展開されるのかが理解できるようになるでしょう。

第1章と第2章では、本書が扱うクラウドゲームと、その範囲を正確に定義します。現在、ゲーム産業ではまだクラウドゲームという用語自体が発展中であるため、その後の章でさらに詳しく技術を掘り下げるためには、正確な定義が必要であるためです。

第3章から第6章までで、本書で扱うクラウドゲームのタイプである「クライアントサイドレンダリング方式」を実装するゲームライブラリmoyaiの内部構成や、通信プロトコルである「スプライトストリーム」がソケット通信の技術を用いて具体的にどう実装されるのかについて、詳しく解説します。

本書では、理論と実践の両輪を目指し、第6章までの理論に加え、実際にプレイできるサンプルゲームを数種類実装し、ゲームサーバーをインターネット上に配置して、リモートでプレイしてみることができるようになっています。そのサンプルコードの使用方法や内容を、第7章と第8章で紹介します。

本書の読者対象および想定する前提知識

本書では、以下のような読者の方々を想定して解説を行っています。

- 現在、ビデオゲーム産業に関わっている幅広い方々。将来、ビデオゲーム産業に関わってみたい方
- インターネットやクラウドに関わる仕事をしている方。将来、その業界に関わる仕事をしてみたい方
- 現在主流のビデオゲーム(非クラウドゲーム)を形づくる描画技術について、大まかに短時間で把握したい方
- 現在のインターネットを支える通信プロトコルの基礎について、大まかに短時間で理解したい方

前提知識や解説レベルに関して、プログラミング部分は、ゲーム開発だけではなく、Web開発や業務アプリ開発などに関わる技術職であれば、しっかりと理解できるレベルを目指しました。

全体として、レンダリングやインターネット通信の基礎の基礎から解説しているので、技術職以外の企画職、経営者など、プログラミングの経験のない方々にも理解できる内容になっています。

合わせて、中学校～高等学校前半までの数学や物理についての知識があれば、詳しい話も含めて理解できるように進めています。

そして、モバイル、PC、ゲーム機など、さまざまな環境でのゲームプレイ経験があると、本書の内容をより深く理解できるでしょう。

動作確認環境について

本書(おもに第7章や第8章)で実装しているサンプルゲームやベンチマークプログラムは、以下の環境で動作確認を行いました。

- macOS 10.13 High Sierra
- Xcode 9.4
- CentOS 7(ゲームサーバーで使用)
- Node.js version10系(Webブラウザ版Viewerプログラムやベンチマークプログラム dummycli.jsで使用)
- Google Chrome 68

本書について.....	iii
本書の構成.....	vi
本書の読者対象および想定する前提知識.....	vii
動作確認環境について.....	vii

第0章

[ゲーム開発者のための]レンダリング再入門

「ゲームプログラムはほとんど描画ばかりを行っている」.....	1
0.1 ゲームプログラムは画面をどのように描画しているか	2
1秒間に60回、画面を書き換える——リアルタイムかつ高速に実行し続ける.....	2
ピクセル数の増加	
GPUによる描画.....	3
ネットワーク(通信)を駆使するクラウドゲーム.....	4
リモートレンダリングとローカルレンダリング/サーバーサイドレンダリングとクライアントサイドレンダリング/クラウドゲームと各種レンダリング手法	
クラウドゲームとネットワークの宿命——レスポンスと通信遅延、帯域幅、費用.....	6
0.2 レンダリングとは何か	6
レンダリングの基本の流れ——最小限の2Dレンダラー.....	7
塗りつぶしとフィルレート/PNGファイルへの出力——libpng	
2Dスプライトゲーム向けのレンダラーへの拡張.....	10
最小限の3Dレンダラー.....	10
座標変換——透視投影変換/塗りつぶし——フィルレートがポイント	
一連の描画処理と出力——最小限の3Dレンダラーと本格的なレンダラー.....	14
0.3 リアルタイムレンダリング	15
リアルタイムレンダリングと性能——ポリゴン描画性能とフィルレート.....	15
2Dゲームのポリゴン描画性能とフィルレート.....	16
0.4 CPUとGPU	18
CPUとGPUのおもな違い.....	18
並列計算/並列化できない計算などの注意/CGの計算と並列化しやすいアルゴリズムの研究	
0.5 グラフィックスライブラリ——OpenGL	21
OpenGL、基礎の基礎.....	21
各種環境とグラフィックスライブラリ	
0.6 補足:映像のエンコーディング	22
フレームバッファと圧縮.....	22
差分とフレーム間予測.....	23
Column クラウドゲームを取り巻く、これまでの状況	
——「クラウドゲーム」登場からの現在までの、長い年月.....	24
0.7 本章のまとめ	24

第1章

クラウドゲームとは何か

マルチプレイゲーム開発が変わる可能性が見えてきた(!?)

1.1	クラウドゲームとクラウドの基礎知識——クラウドゲームの定義、クラウドの実体	30
	クラウドゲームの定義	30
	クラウドコンピューティング、基礎の基礎	31
	クラウドの実体/典型的なクラウドとオンラインサービス	
	計算機資源の利用効率と新たなサービス——リモートデスクトップと3D CADの例	33
	大量のマシンを1カ所に集めると、変わるごと——利用効率や高速化と、ソフトウェア自体の設計見直し	
	クラウドゲームの事業化とゲーム産業のビジネスモデル	36
1.2	クラウドゲームでゲームの「何」が変わるか	37
	——クラウドゲームの技術的な特徴、ゲームの内容によらない変化	37
	クラウドゲームの3つの技術的な特徴	37
	技術的な特徴から生まれる多くのメリット——ユーザー視点、開発者視点	38
	クラウドゲームのおもな課題と解決の兆し——インフラ費用、通信の遅延	39
1.3	マルチプレイゲームで起きる“大”変化——マルチプレイゲームが簡単に実装できる	40
	マルチプレイゲームを、オフラインゲームと同じ方法で実装できる(!?)	40
	ファミリーコンピュータ時代から学びたい。みんなで楽しむマルチプレイゲームのルーツ	40
	『バルーンファイト』の協力プレイ/マルチプレイゲームの基本形/『マリオカート8デラックス』の異なる視点のマルチプレイ/オフラインマルチプレイ——ゲームプログラムの基本構造は同じ	
	クラウドゲームは、コントローラーのケーブルを地球の裏側まで伸ばしてくれる	43
1.4	オンラインマルチプレイとクラウドマルチプレイの対比	45
	——ゲーム実装時、ネットワークプログラミングは不要になる	45
	オンラインマルチプレイの二大実装方法のおさらい	45
	クラウドマルチプレイとP2P MMOゲームとの比較	46
	クラウドマルチプレイとC/S MMOゲームとの比較	47
	クラウドマルチプレイは富豪的な解決策(!?)	49
	ゲームエンジンのマルチプレイ機能も、根本的な解決にはならない/スプライトストリーム——中ぐらいに富豪的な方法	
	Column vGPU機能——リモートデスクトップ用のシステム	51
	クラウドマルチプレイの特徴	52
1.5	クラウドマルチプレイ最大の課題「インフラ費用」	52
	——クライアントレンダリング方式の導入	52
	インフラ費用の内訳	53
	Column 現行のクラウドゲームの料金体系	53
	マシンを保持する費用——インフラ費用①	54
	ビデオストリームとGPUの費用——GPUを用いる現在のクラウドゲームサービス	
	通信費用(インターネットに対する送信)——インフラ費用②	55
	ビデオストリームとサーバーサイドレンダリング——従来からのクラウドゲームサービス	55
	「クライアントサイドレンダリング」という発想——数十～数百人規模のマルチプレイゲームの実現	56
	2Dゲーム向けのゲームライブラリの開発——既存のエンジンでは未実現	
1.6	クライアントサイドレンダリングの基礎——ゲームの入力/処理/描画の分離	57
	オフラインゲームのプログラムの基本構造——古典からクラウドマルチプレイまで	57
	現代的なゲームプログラムの処理段階	58
	クラウドマルチプレイは、入力ガリモート	58

クライアントレンダリングで、描画する位置もリモート	
— 現代的なゲームプログラムの設計を活かして、サーバーとクライアントが連携して描画する	58
図解でわかるクライアントサイドレンダリングのプログラムの構造	
クライアントサイドレンダリングは、C/S MMOより多くの帯域とCPUを消費する	59
ゲーム内容に特化した抽象化—送信データの抽象度を高める	60
Column クライアントレンダリングの2つの方式	61
Column 描画パッチ送信方式、スプライトストリーム(スプライト情報送信方式)	61
サーバーから送信するデータの内容と処理負荷	
— クライアントサイドレンダリングと、ゲーム内容に特化した抽象化の比較	62
ビームの破片が飛び散るエフェクトの例	
スプライトストリームの概略—クライアントサイドレンダリングで用いる抽象度の高い通信のプロトコル	64
スプライトストリームで削減できるマシンを保持する費用と通信費用—サンプルゲームを用いた測定結果について	
クラウドでMMOG—クラウドマッシュマルチプレイゲーム	65
1.7 本章のまとめ	66

第2章

クラウドゲームのアーキテクチャ

実現したいゲーム×技術の適性を見定める

2.1 物理的接続構造 —1:1/1:N/N:N/M:N	68
物理的接続構造の分類—シングルプレイ用とマルチプレイ用	68
ゲームサーバーのプロセスとプレイヤーの対応関係	
1:1で見る物理的接続構造の基本—入力から出力までの流れ	69
物理的接続構造とゲームプレイ空間—図の簡略化から	70
1:1—技術的導入コストは低い、サーバーコストは高くなる	72
1:N—GPUから見ると1つの画面を描画しているが、実際にはプレイヤーは複数存在する	72
N:N—マルチプレイでゲームプレイ空間を共有する場合、ゲームプレイ空間の同期が必要	73
M:N—システム構造は複雑になるが、コスト面でメリットもある	73
サーバーマシンのCPUとメモリの節約—1:NとM:Nのみ	73
ゲームサーバーのCPU消費量の比較—1:Nのアドバンテージ① / ゲームサーバーのメモリー消費量の比較—1:Nのアドバンテージ②	
2.2 画面描画の実装方法 —サーバーサイドレンダリングとクライアントレンダリング	76
2種類の画面描画タイプ	76
図解でわかるサーバーサイドレンダリングとクライアントサイドレンダリング—GPUの位置に注目	76
サーバーサイドレンダリング / クライアントサイドレンダリング	
サーバーサイドレンダリングとクライアントサイドレンダリングの歴史	78
サーバーサイド/クライアントサイドレンダリングの使い分け—ゲーム内容とさまざまな制約	79
サーバーサイドレンダリングを使う最大の目的—高度なグラフィックスを求める	
消費電力の大きさとサーバーサイドレンダリング	
— 必要な計算処理の量を消費電力で大きめに比較する	80
ソフトウェアレンダリング—サーバーサイドレンダリングの特殊なオプション	
クライアントレンダリングの苦手分野—クラウドゲームの制約とゲームデザイン	82
サーバーサイドレンダリングとクライアントサイドレンダリングの使い分け	82
2.3 ネットワークプログラミングなしでMMOGをつくれるか	
—「1:N」と「クライアントサイドレンダリング」が持つ可能性	83
「1:N」では1つのプロセスが1つのゲームプレイ空間を保持—プロセス間通信が不要	83
プロセス間通信のおさらい—複数のプロセスで1つのゲームプレイ空間を保持	84

プロセス間通信の要不要—N:Nと1:N	
ネットワークプログラミング、非同期プログラミングの難しさ —ソケットを用いる場合のゲームサーバーに必要な機能	85
「1:N」では、ネットワークプログラミングなしでマルチプレイゲームの実現が可能である —第1段階の結論	86
さらに、MMOGの実現に向けた、「クライアントサイドレンダリング」の潜在能力—第2段階の結論	86
「ネットワークプログラミングなしでMMOGはつくれそう」である—現時点で導かれた結論	87
2.4 クラウドゲームに向いているゲーム、向いていないゲーム—遅延と経済性	87
クラウドゲームへの向き不向き—3つの観点で考える	87
① ゲームが許容できる遅延の大きさ—原因、インターネットの遅延、ゲーム内容、フレームレート	88
ビデオゲーム全般における遅延の原因/クラウドゲーム特有の遅延の原因	
最大の原因はインターネットの遅延—pingコマンドで測定しておこう	91
ゲームの内容と、低遅延への要求—「リアルタイムゲーム」「ターンベースゲーム」の大枠から	92
タイミングゲーム—低遅延への要求/入力方式/操作系統による低遅延への要求の違い—継続的入力、 ポイント&クリック	
フレームレートの範囲を考える—基本とクラウドゲームでの注意点	94
フレームレートの調整時の注意点/フレームレートの調整とゲームプレイ—同じゲームでもコンテキスト次第 で必要な条件は変わる	
② サーバー運用者にとっての経済性	96
各種のインフラ費用—大半を占める通信費用	97
1同時接続あたりの通信費用/GPU+CPUの費用—通信以外にかかるマシン費用	
Column ビデオストリームとスプライトストリームの通信帯域消費	100
③ エンドユーザーにとっての経済性	101
Column OpenGLと開発環境 moyaiで用いるグラフィックスライブラリ	102
2.5 本章のまとめ	102

第3章 開発の道具立て 開発環境と2Dクラウドゲームライブラリとしてのmoyai

3.1 開発環境の基礎知識—ゲーム開発全般からクラウドゲームまで	104
ゲーム全般の開発環境の概略—「ゲームエンジンを使う」方法から	104
ゲームエンジンを使わない方法	
クラウドゲームのために必要な開発環境の機能—既存の開発環境の状況	106
サーバーサイドレンダリングと既存の開発環境—独立ソフトウェア方式とゲーム内蔵方式/クライアントサイド レンダリングと既存の開発環境	
クラウドゲーム実装の支援はこれからの伸びしろに期待	109
3.2 moyaiの基本情報—2Dの新しいクラウドゲームライブラリ	110
moyaiとシステム構成	110
moyaiの依存関係—外部モジュール	111
プロトタイプ制作からクラウドゲーム対応の本格ライブラリへ—moyaiの特徴、機能、実装状況	111
3.3 [速習]各種ストリーム—クラウドゲームのための通信内容	113
データストリームとインプットストリーム	113
スプライトストリームとビデオストリーム—基礎の基礎	114
オーディオイベントストリームとオーディオサンプルストリーム	114
画面描画と音再生のストリームの組み合わせと、通信帯域/オーディオサンプルストリームの活用とCPU 消費の注意	

moyaiはおもにスプライトストリームとオーディオイベントストリームを想定	116
3.4 目指すゲームとライブラリ設計のための緻密な見積もり	
—MMORTS/MMORPGを例に.....	116
最初のコミット —MMORTSの要素を持つMMORPGをつくる	117
Moai SDKを参考に、GLFWとC++で実装開始	
MMORTS/MMORPGは、結果的にスプライトストリーム方式のクラウドゲーム向き (!?)	118
MMORTS/MMORPGと遅延、通信費用	
オブジェクト数を中心とした見積もりは重要 —ゲーム内容、性能、コスト.....	118
オブジェクト総数の見積もり/ゲームサーバーの1プロセスあたりの、オブジェクト数の上限—総アップデート頻度の上限	
実装言語による違い	121
ゲームの処理と処理系の速度—コンテナ、メンバー参照/オブジェクトを検索する処理の例—シェーティングゲームにおける敵キャラの当たり判定	
思考ルーチンの計算負荷	123
最適な開発環境を探して	124
つくりたいゲームと開発環境の性能 —ゲーム内容次第で適材適所	124
もし、毎フレーム動かすわけではないなら/シェーダを使って実装できるもの—ただし、GPU内部で完結している場合に限る/今回のゲームには大量のオブジェクトがある—変換のオーバーヘッドとその対応策、しかし—	
サーバーとクライアントでC++コードを共有したい	
—さらにクラウドゲームライブラリとして実現したいポイント	126
開発効率を重視し、「同じAPI」が良い—オブジェクトシステム、タスクシステム、算術API	
つくりたいゲームのために、ライブラリをつくる —クラウドゲームありきではなかったmoyai.....	127
3.5 moyaiライブラリの基本アーキテクチャ —レンダリング工程に沿った機能の分割	127
moyaiの2つの利用形態 —リモートレンダリング用、完全版.....	128
レンダリング工程別のレイヤー構成 —「スプライトシステム」と「描画機能」.....	128
ローカルレンダリングとリモートレンダリングに対応するしくみ/リモートレンダリング用(サーバー専用ライブラリ)と完全版について	
moyaiの対応状況と、現時点でできること	130
3.6 本章のまとめ	130

第4章

OpenGLを用いたローカルレンダリングのしくみ moyaiの基礎部分。描画APIと内部動作

4.1 moyaiと汎用的な論理データ —レンダリング工程の第1段階.....	132
はじめてのmoyai —min2dでローカルレンダリングの動作確認	132
moyai APIの(汎用的な)論理データの構成 —MoyaiClient、Layer、Prop2D	133
Layerの機能	134
CameraとViewportによるカリング	134
Viewportとゲーム空間における座標の関係/ Cameraの位置/ カリングの効果/ Viewportの効果的でない方/複数のViewportを併用する/ Cameraを使ってスクロールゲームをつくる	
LayerとCamera、Viewportの自由自在な組み合わせ	140
スプライトの見た目を決定する論理データ	
—Texture、TileDeck、インデックス、スケール、位置、色、模様	141
OpenGLのテクスチャ描画機能とスプライトの描画	142
テクスチャとなる画像/ PNG画像のなにかみ/ RGBAフォーマットとGPU/ OpenGLにおける画像描画の指示—glGenTextures関数	
UV座標(テクスチャ座標) —glDrawElements関数、頂点バッファ、インデックスバッファ	146
ポリゴンに色や模様をつける	

スプライトシート.....	148
格子状のスプライトシート／柔軟なスプライトシート／moyaiのスプライトシート／TileDeckを使わず直接テクスチャを貼る方法	
4.2 moyai内部のマシンに依存したポリゴンデータ ——レンダリング工程の第2段階.....	153
ポリゴンデータとバッチ化.....	153
グラフィックスハードウェア構成とレンダリング.....	153
ローカルレンダリングの流れ／データの転送と所要時間	
バッファオブジェクト——GPUへの転送回数を減らす.....	155
moyaiにおけるバッファオブジェクト——5種類／モバイル端末におけるOpenGL ES——バッファオブジェクトを使わないOpenGLプログラミングが不可能	
描画のバッチ化とglDrawElements関数.....	157
複数スプライトをバッチ化するための条件——同じテクスチャと同じシェーダを用いている場合に限る	
ドローコールの回数を減らすための設計——スプライトを「同じテクスチャ」から描画する.....	159
(C o l u m n) 予習&復習・moyaiのリモートレンダリングの基本構造.....	160
4.3 本章のまとめ	160
第5章	
スプライトストリームのしくみ	
リモートレンダリング方式概論、スプライトストリームの基本設計.....	
5.1 クラウドゲームと画面描画 ——moyaiが対応する画面描画の通信方式.....	161
クラウドゲームにおける画面描画の通信方式——7つの方式.....	161
①外部機器方式／②キーボード方式／③独立ソフトウェア方式／④ゲーム内蔵方式／⑤OpenGL LAN仮想化方式／⑥OpenGLインターネット仮想化方式／⑦抽象度の高い通信方式	
OpenGLインターネット仮想化と抽象度の高い通信(スプライトストリーム)が送信するデータ.....	167
(C o l u m n) スプライトストリーム以外の抽象度の高い方式についての試案	
——Processingストリーム.....	168
moyaiのビデオストリームとスプライトストリーム.....	169
5.2 スプライトストリームのなかみ ——抽象度の高い情報.....	170
スプライトが持つ情報.....	171
OpenGL関数の呼び出しと「OpenGL呼び出し用の情報」、moyai内部の「抽象度の高い情報」	
2Dゲームに特有の描画パターンと通信帯域への影響.....	172
大きな差が生まれる原因は「バッチ化」	
5.3 スプライトストリームの送信内容 ——min2dでmoyaiの動作を見てみよう.....	174
min2dの動作確認とログの見方——スプライトストリームなし.....	174
(C o l u m n) OpenGLの仮想化を使う別の方法——ただし、原稿執筆時点では未実験.....	175
min2dでのスプライトストリーム.....	176
スプライトストリームの3段階の送信内容.....	177
描画準備(リソース送信)の様子／スナップショット送信の様子／差分送信の様子	
スプライトストリームが向かない2Dゲーム.....	178
3Dゲームにおけるスプライトストリーム.....	179
(C o l u m n) moyaiのWebブラウザ版Viewer.....	180
5.4 スプライトストリームのサーバー／クライアント構成とレプリケーション	184
レプリケーションの基礎——スプライトストリームの特性を活かす.....	184
レプリケーションサーバー	

スプライトストリームの物理的接続構造とレプリケーション.....	185
レプリケーションサーバーの基本構成.....	185
レプリケーションツリー／レプリケーションのメリットとデメリット	
レプリケーションの構成パターン.....	187
スプライトストリームでのゲームサーバーの仕事/処理/スプライトストリームでのViewerの仕事/処理	
サーバーサイド1段レプリケーション.....	188
サーバーサイド多段レプリケーション.....	189
クライアントサイドレプリケーション.....	190
レプリケーションクライアントの4つの条件——リモートからの接続が可能、通信帯域、通信遅延、通信の安定性／レプリケーションマッチング／クライアントサイドレプリケーションのメリットとデメリット／クライアントサイドレプリケーションのセキュリティ——入力/出力、簡単な基準設定、変更の検出	
サーバー/クライアント混合型レプリケーション——レプリケーションサーバーの柔軟性を活かす.....	195
5.5 レプリケーション×伝統的なMMOG	
——スプライトストリームのレプリケーションが応用可能.....	196
伝統的なMMOGの基本構成——レプリケーションサーバーなし.....	197
サーバーサイドレプリケーション×伝統的なMMOG.....	197
ゲームサーバーからの送信量——レプリケーションサーバーを使うメリットの大小	
クライアントサイドレプリケーション×伝統的なMMOG.....	199
5.6 カリングとストリームのチェックサム	
——クライアントレプリケーションにおけるストリームの改ざんの検出.....	200
ストリームのチェックサムとカリングによるバケットの欠落.....	200
バケットのサイズと帯域消費.....	201
チェックサムを用いた抜き打ちの内容確認——改ざん防止.....	202
抜き打ちのチェックサムのメリット——柔軟に調整できる／抜き打ちのチェックサムの限界——バケットが欠損してはいないかはわからない	
Column TIMESTAMPバケットとリプレイ機能.....	204
5.7 本章のまとめ	204

第6章

スプライトストリームの通信プロトコル

TCPによる送受信、ストリームのなかみ、関数群..... 205

6.1 スプライトストリームとネットワークレイヤー——レイヤー1~4、レイヤー5~7..... 205

スプライトストリームのクラウドゲームシステムとレイヤー——本章の解説について..... 206

スプライトストリームを形づくる各層..... 207

6.2 レイヤー1~4:スプライトストリームの基礎部分
——膨大な数の機材をつなげて、高品質な通信をどのように実現するのか..... 209

インターネットとは何か——地球全体をつなぐ壮大な技術..... 209

インターネットの基礎——バケット(データグラム)..... 210

 バケット中継の経路を確認——traceroute/tracert

クラウドゲームとバケットロス..... 211

 電気信号(電波)のノイズによるバケットロス/ルーターの混雑によるバケットロス/ネットワークの輻輳

 ——ルーターの混雑が悪化/ベストエフォート——できるだけがんばる

大人数が一つの通信媒体を共有するための工夫..... 214

Ethernetフレーム——Ethernetでデータを送る..... 215

 IP断片化とIP再構成/データの大きさと送信成功率

TCPの基本..... 217

パケットの再送—TCPの送り方①／順序保証—TCPの送り方②／輻輳制御—TCPの送り方③／スロースタート(アルゴリズム)／TCPの歴史的發展と最適なアルゴリズムの選択	
UDPとデータグラムの喪失 —UDPならストールは発生しない?	221
Column RUDP	223
ストリーム指向プロトコルであるTCP —ストリーム指向とメッセージ指向	224
データの区切り—ソケットプログラミングとバッファリング	
TCPストリームへのデータの送信 —パケットロスの影響をできるだけ小さくする	225
ネットワークを流れているパケットヘッダーの中身 —Ethernet/IP/TCP	226
IPパケット／TCPパケット(セグメント)	
TCPの状態遷移	229
ソケットAPI —ブロックング、ノンブロックング、非同期API、select/epoll	232
libuv —非同期I/Oの実現	233
6.3 レイヤー5~7:スプライトストリームの送信内容	236
レイヤー1~7:スプライトストリームのデータ—moyaiのヘッダー	237
レイヤー5:スプライトストリームの接続確立と維持	237
レイヤー6:スプライトストリームのパケット構造	238
パケットの中身	238
バイナリデータ、RPCの関数ID、関数への引数データ	239
レイヤー1~6:スプライトストリームの内容自体には関わっていない	240
レイヤー7:スプライトストリームが送出されるまで	241
スプライトストリームの実例—バイナリデータを見てみよう	
スプライトストリームの関数呼び出し —描画準備、スナップショット送信、差分送信	242
TIMESTAMPの受信時刻に関する注意点	
「描画準備の送信」「スナップショット送信」「差分送信のループ」	245
(接続完了直後)描画準備の送信—scanSendAllPrerequisites関数	245
(画面全体の)スナップショット送信—scanSendAllProp2DSnapshot関数	247
(スプライトの)差分送信—スプライトの状態の変化分を送信し続ける	248
差分抽出:すべてのスプライトの変化を検知 —どのスプライトの差分を送るのかを決定する段階①	249
Prop2Dのスナップショットの構造体—PacketProp2DSnapshot構造体／PacketProp2DSnapshot構造体を用いた差分抽出	
差分スコアリング:差分スコアでソートし、大きく動いたスプライトを優先送信 —位置の同期モードと、どのスプライトの差分を送るのかを決定する段階②	252
差分スコアリング—位置の同期モード①／線形補間—位置の同期モード②／速い物体と壁へのめり込み問題—moyaiのデフォルトが差分スコアリングである理由	
可視判定:可視範囲に入っているスプライトだけを送る —どのスプライトの差分を送るのかを決定する段階③	256
Prop2Dの差分抽出と可視判定 —2つの送信モード	256
補足:カリング処理と今後の課題 —どのスプライトの差分を送るかを決定する段階を経て	257
スプライトストリームのデータ圧縮 —送信量の削減①	258
スプライトストリームのバッファリングとヘッダー圧縮 —送信量の削減②	259
ヘッダーとデータの割合／moyaiのスプライトストリームでは、NagleアルゴリズムOFF／スプライトストリームの送信頻度の変更—send_wait_msオプション／遅延とフレームレートの問題—replayerツールとTIMESTAMPの値の活用／[実測]ヘッダー消費量	
6.4 スプライトストリームとレプリケーション	263
レプリケーションサーバーにおけるカリング—送信量を削減するさらなる工夫	263
レプリケーションサーバーの実体／カリングの対象	
レプリケーションサーバーでスケールアウトできる、できない	265
HUDとスプライトストリーム、レプリケーションサーバー	
6.5 スプライトストリームの関数(抜粋) —描画準備、実際の描画、サウンド/インプットストリーム/レプリケーションサーバー制御	267
スプライトストリーム関数の命名規則	267

描画準備のための関数群.....	268
描画準備の流れ	
インプットストリームの関数.....	272
6.6 本章のまとめ	274

第7章

[クイックスタート] クラウドゲーム 多彩な軸のサンプルゲームから見えてくる要所

7.1 moyai_samplesのセットアップ	276
① 手元のMac環境でのビルド.....	276
② サンプルゲームを実行.....	278
③ macOSマシン上でのスプライトストリームの確認.....	278
④ Linuxでのビルド.....	279
⑤ Linuxでの動作確認.....	280
サンプルコードの基本的起動オプション—— <code>--ss/--vs</code>	282
min起動後の画面とviewerの状態表示.....	282
ビデオストリームとスプライトストリームの違い	
7.2 最小構成の設定を把握する ——独自の定義は15行のサンプルプログラム「min」から.....	285
min.cpp.....	285
サンプルコードのmain関数—— <code>sample_common.h</code> , <code>sample_common.cpp</code>	286
スプライトストリームの送信のための初期化——共通のオプション.....	287
7.3 性能限界を測定する ——ベンチマークプログラム「bench」.....	289
benchの基本——起動、スプライトの数をどんどん増やして動作確認.....	289
CPU消費量——CPU時間の使い道の確認 (macOSの例).....	290
通信帯域の消費量と帯域節約の検討.....	292
パケットの内容/サイズと通信量/理論値と実際の通信量、節約の可能性	
7.4 通信遅延と帯域消費を確認する ——弾幕シューティングのサンプル「dm」.....	293
dmのソースコード概説.....	293
通信遅延の確認——弾幕サンプルゲームの実行、遅延の追加.....	294
通信のRTT——通信遅延の確認のポイント/遅延を追加して動作を見る—— <code>tc</code> コマンドの <code>netem</code> フィルター/遅延の許容範囲と測定——クラウドゲームとして成り立つか	
通信帯域の確認——スプライトストリームの送信頻度.....	298
送信頻度の変更オプション/ゲーム内容に合わせて送信頻度を調整——等速直線運動と線形補間/moyaiのデフォルトの設定——アクションゲームに対応可能な最低限の頻度/— <code>linear-sync-score-thres</code> と線形補間/位置更新の基準となる累積の移動距離を調整する——線形補間が使えない自機や敵/最適を目指して、調整作業は続く	
7.5 極小の「通信量×CPU消費×コード」で見えるスプライトストリームの威力 ——リバーシのサンプル「rv」.....	302
rvの基本——操作、起動、帯域消費の確認.....	303
クラウドゲームとモバイル環境——通信の速度と料金	
rvのソースコード——100行&ネットワークプログラミング一切なしで実現できるマルチプレイ.....	304
グローバル変数定義/初期化関数 <code>reverseInit</code> の定義/更新関数	
通信量もCPU消費量もソースコード量も極小のrv ——ネットワークプログラミングなしでマルチプレイを実現する、スプライトストリームの威力.....	309
7.6 通信遅延に、シビアナ“衝撃波”を繰り出す ——一対一の対戦格闘サンプル「duel」.....	310

duelの基本	310
対戦格闘ゲームで確認する通信の遅延とクラウドゲームの限界	311
自動操作とプレイ感覚の確認／リモートサーバーを使う場合のプレイ感覚とUDP	
duelのソースコード概説——250行でつくる対戦格闘ゲーム	313
7.7 大人数同時プレイのMMOGへの第一歩	
——スクロールとDynamic Cameraを操るサンプル「scroll」	315
scrollの特徴——Dynamic CameraとDynamic Viewport	315
scrollの基本	315
scrollのソースコード	316
PCクラスの定義——画面の描画や操作をクライアントごとに保持／PCのコンストラクタの前半／PCのコンストラクタの後半	
キーボード/マウスの個別化とDynamic Camera/Dynamic Viewport——scrollのポイント	319
scrollでのキーボード処理	320
7.8 本章のまとめ	323
Column 論理データと、ポリゴンデータ/物理データの関係	324
第8章	
[本格実装]クラウドゲーム	
ネットワークプログラミングなしで、MMOGの実現へ	325
8.1 k22の開発基礎 ——「scroll」ベースの全方向シューティングゲーム	325
k22の基本情報——スプライトストリームのクラウドゲームのサンプル	326
k22のビルド	327
クラウドゲームでゼロからつくるMMOG! k22の開発の流れ——マルチプレイ化が簡単	328
開発準備:画面に描画される要素をリスト化——ゲーム内容の洗い出し、作業やコードの量の見積もり	328
k22のゲーム内容を形づくる要素のリスト	
各段階におけるk22のリビジョン	330
8.2 第1段階:シングルプレイゲームの実装	331
① moyai_samplesからソースコードをコピーする	331
main関数と全体像の準備／ゲームの初期化:gamelnit関数／ゲームのメインループ	
② ゲームのオブジェクトシステムを実装する	336
moyaiの省力化のしくみ／クラスの階層構造——省力化のしくみ①／アップデート関数の鎖構造——省力化のしくみ②／ツールで見る関数呼び出しの連鎖／サブモジュールのリビジョン／コードで見る関数呼び出しの連鎖／アップデート関数の鎖構造のまとめ	
③ 必要なキャラクターの動きと見た目を定義する	345
プレイヤーキャラクターの登場／子Prop2D——キャラクターのパーツの重ね合わせ	
④ キャラクターをキーボードで操作する	350
Padクラスと2つのメリット／キーボードから入力をキャラクターの動きに反映させる／キーの押し下げ状態をベクトルに変換するPadクラス／障害物や通れない場所の扱い	
⑤ 背景と地形を表示する	354
チャンキング	
⑥ ビーム弾を撃つ	359
⑦ 音を鳴らす	362
⑧ 敵を出現させて倒す	363
第1段階のまとめ——シングルプレイの段階で楽しめる要素をひとつとori実装する	365
8.3 第2段階:マルチプレイ化	366
マルチプレイ化の流れ	366

① gamelnit関数でRemoteHeadを初期化する部分を追加.....	366
② コールバック関数を追加.....	369
③ main.cppにaddPC、delPCなどを追加——onConnectCallback、onDisconnectCallback.....	369
④ グローバル変数 g_mouse、g_keyboardなどをPCクラスのメンバー変数に変更.....	371
k22とrvのキーボード/マウスの操作イベントの流れの違い——参加者全員でキーボードとマウスを共有するrv	
⑤ PCクラスのコンストラクタで、keyboardやmouseなどのメンバー変数を初期化.....	373
⑥ キーボード操作イベントの処理内容を変更.....	374
⑦ マウス操作イベントの処理内容を変更	
——onRemoteMouseButtonCallback、onRemoteMouseCursorCallback.....	375
⑧ PC::charPoll関数でグローバル変数ではなく、メンバー変数を使うように変更.....	375
第2段階のまとめ——桁違いに小さな作業でマルチプレイ化.....	376
B.4 第3段階:プログラムの性能測定と運用コスト予測.....	377
k22の運営のためのコストの概観.....	377
ゲームアプリにおけるインフラ費用予測——同時接続数から考える.....	378
CPU消費量——k22は何のためにCPUを消費しているか.....	378
キャラクターを動かす処理と通信のための処理/キャラクターを動かす処理とCPU消費量/通信のための処理とCPU消費量/CPU消費量の内訳を確認——「プロセスのサンプルを収集」/圧縮機能と、CPU&通信帯域の消費量	
通信帯域の測定.....	386
k22の簡易的な負荷テスト.....	387
負荷テストで見積もるCPU&通信帯域の消費量/負荷テストと実運用	
メモリー消費量.....	390
インフラ費用試算の目安の値——k22のCPU、メモリー、通信帯域の消費量.....	390
インフラ費用の試算.....	390
クラウドサービスの利用——コンテナと仮想マシン	
仮想マシンの費用——CPUとメモリーの使用料金.....	391
Column AWSの各種インスタンス.....	394
通信帯域の費用.....	395
コスト試算の結果——k22のサーバーをAWSで運用した場合.....	395
アクセス量の変動とピーク調整比.....	396
ピーク調整比を加味した、通信費用とマシン費用の予測	
第3段階のまとめ.....	397
B.5 第4段階:将来に向けた開発を見通す.....	398
通信量の削減——最も効果の大きいところから順番に変更せよ.....	398
サーバーの運用/管理を支援するツールの開発.....	401
ゲーム内容の追加/修正.....	401
第1段階から第4段階までの実装時間.....	402
B.6 思考実験——非クラウドゲームとして実装していたら、どうなっただろうか.....	403
ネットワークプログラミングなしと各種のトレードオフ——通信帯域、CPU負荷.....	403
スプライトストリームのクラウドゲーム×伝統的なMMOG.....	404
出現するものごとの比較	
スプライトストリームと典型的なオンラインマルチプレイの違い.....	407
B.7 本章のまとめ.....	408
索引.....	409