

東京大学
情報理工学系研究科 電子情報学専攻
修士論文

マルチコンテキスト没入環境のための
Windowing System の実現に関する研究
The design and implementation of a windowing system
for multi-context immersive environments.

48-216413

木内 陽大
Akihiro Kiuchi

指導教員 江崎 浩 教授

2023年1月

概要

日本語の要約.

Abstract

English abstract.

目次

| | | |
|--------------|----------------------------|----|
| 第 1 章 | 緒言 | 1 |
| 1.1 | 背景 | 1 |
| 1.2 | 問題点 | 2 |
| 1.3 | 本研究の目的 | 3 |
| 1.4 | 本論文の構成 | 3 |
| 第 2 章 | 提案するシステムの概要 | 4 |
| 2.1 | はじめに | 4 |
| 2.2 | Windowing System | 4 |
| 2.3 | 類似研究・技術 | 5 |
| 2.4 | 本論文での結論 | 7 |
| 第 3 章 | レンダリングプロトコル | 10 |
| 3.1 | 3D コンピュータグラフィックス | 10 |
| 3.2 | 関連技術・研究 | 12 |
| 3.3 | 提案手法 | 15 |
| 3.4 | 評価：レンダリング性能 | 19 |
| 3.5 | 評価：レンダリングの自由度 | 21 |
| 3.6 | まとめ | 25 |
| 第 4 章 | 2D アプリケーションの利用 | 26 |
| 4.1 | 関連技術・研究 | 26 |
| 4.2 | 提案手法 | 27 |
| 4.3 | 評価 | 29 |
| 4.4 | まとめ | 29 |
| 第 5 章 | 結言 | 32 |
| 5.1 | レンダリングプロトコル | 32 |
| 5.2 | 2D アプリケーションの利用 | 32 |
| 5.3 | 今後の展望・展開 | 33 |

vi 目次

| | |
|------|----|
| 発表文献 | 34 |
| 参考文献 | 35 |

第1章

緒言

1.1 背景

Virtual Reality (VR) や Augmented Reality (AR) などの没入環境の社会実装は昨今急速に進んでおり、比較的安価なコンシューマ向けの Head Mounted Display (HMD) の登場により、多くの人が没入環境を体験、利用できるようになった。没入環境の利用シーンは年々多様化しており、コンシューマ向けの VR/AR マーケットではゲームなどエンターテイメント向けの利用が目立つが、VRChat^{*1} や Horizon World^{*2} といったコミュニティに重きを置いた利用も活発になり、コンピュータネットワーク上の新たな世界を指すメタバースという言葉が流っている。また没入環境の職業支援や、職業訓練への応用も進んでおり、医療 [1] や航空宇宙 [2]、軍事 [3]、農業 [4] などさまざまな分野での研究が活発で、実際に産業界での導入も進んでいることが IDC のレポート [5] からわかる。

ここで述べておきたいことは、昨今の没入環境の利用シーンは多様化しており、様々なコンテキストにおいて没入環境の利用が期待されているという点である。ここでのコンテキストとは、没入環境の内外に関わらず、ユーザがどのような目的を持ってどのような作業をしようとしているかを指し、それは利用時の状況（休日で趣味に時間を使いたいのか、平日で仕事をしたいのかなど）やユーザの職業などの複雑な組み合わせのもとに生起され、個人ごとの差異があり、個人のなかでもその時々ごとに連続的に変わってゆくものであるとする。ここで連続的と表現しているのは、ユーザがもつコンテキストは料理をする、勉強をする、などと離散的に切り替わってゆくものではなく、料理をしながら動画を見る、料理をしながら一度動画を止めてタイマーを開始する、一度料理をやめてメールを確認する、と連続的に変化するものであることに注意したいからである。

^{*1} VRChat Inc. “VRChat” <https://hello.vrchat.com/> (accessed on 24 Jan, 2023)

^{*2} Facebook Technologies, LLC. “Horizon Worlds” <https://www.oculus.com/horizon-worlds/> (accessed on 24 Jan, 2023)

1.2 問題点

これから現在の没入環境のパラダイムの問題点を考える前に、これと比較するため2次元のデスクトップ環境のパラダイムを考察する。2次元のデスクトップ環境では、1つのスクリーン空間に複数のアプリケーションを立ち上げて使うことができる。アプリケーションは随時開発が可能で、様々な開発元のアプリケーションをダウンロードしたり、アプリケーション自分で作成したりして、他の開発元のアプリケーションと同時に組み合わせて利用できる。アプリケーションの起動や停止も随時可能であるため、ユーザは自分に必要なアプリケーションをダウンロードしたり、開発したりしておき、時々のコンテキストに合わせて必要なアプリケーションを起動して、不要なアプリケーションを停止するということを行っている。これによつて1つのスクリーン空間を柔軟に変化させ、様々なコンテキストに適応させて用いている（図1.1）。

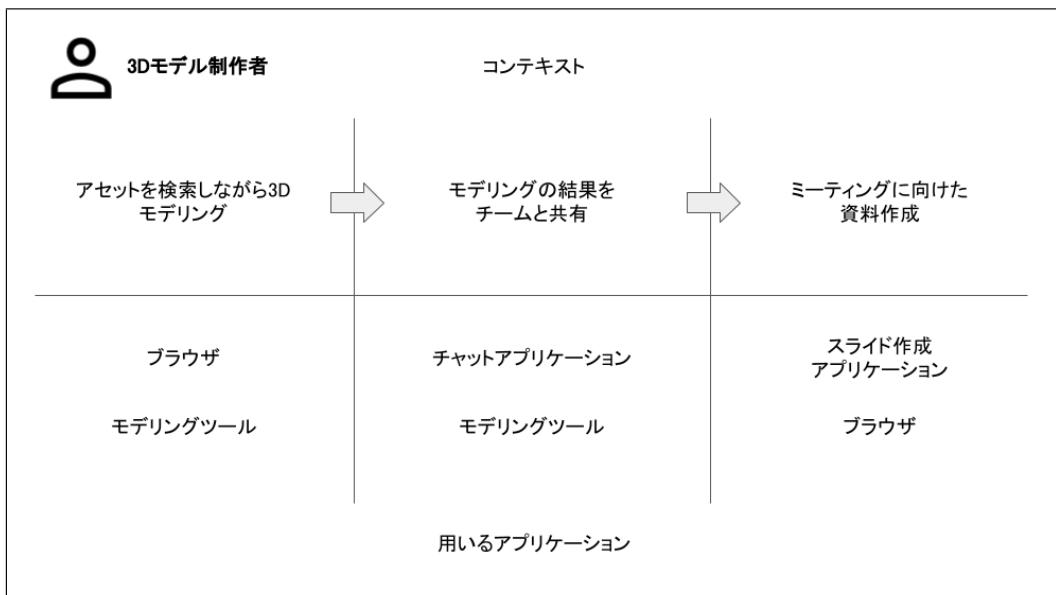


図1.1. 2次元のデスクトップ環境のユーザがコンテキストを変化させてゆく例。3Dモデルの制作者を例に、コンテキストが上段の左から右のように変化していくとき、3Dモデル制作者は下段のようにそのコンテキストの変化に対応して、アプリケーションを開じたり、新しく開いたりすることでスクリーン空間を柔軟に適応させることができる。

以上の点から2次元のデスクトップ環境でのスクリーン空間は不特定のコンテキストに対応可能な、マルチコンテキストな空間であるといえる。

比較して現在の没入環境のパラダイムでは、基本的に1つのアプリケーションがユーザの視野全体を支配している。そのためユーザがコンテキストを切り替えるときは、アプリケーションを別のものに切り替え、全く異なる世界に移動するような形となる（図1.2）。このため、没入環境での3D空間はその空間を支配する1つのアプリケーションによって、会議をす

るといった特定のコンテキストのために設計されているシングルコンテキストな空間となっている。没入環境が今後ユーザの普段の生活に溶け込み、継続的に生活や仕事をサポートするようになるためには、ユーザの連続的に変化し続けるコンテキストに柔軟に対応可能な、新しいパラダイムが必要である。

Google Earth VR^{*3}VRChat^{*4}

図 1.2. 没入環境でコンテキストを切り替える例。地図を見るというコンテキストから大人数でゲームをするというコンテキストへの変化を考える。この時ユーザは左図のような空間全体を支配した地図アプリケーションの中において、このアプリケーションを停止し、一度ホームの空間に戻る。そこから次は右図のようなソーシャルアプリケーションを起動し、そのアプリケーションの世界に入るという流れになる。このように現在の没入環境ではシングルコンテキストな 3D 空間の間を、非連続的に切り替えることしかできない。

1.3 本研究の目的

本研究の目的はマルチコンテキストな没入環境の実現を目指し、その新しいパラダイムの可能性と必要な基盤システムを詳細に検討することである。その検討が机上の空論とならないために、本研究では実際に動くシステムを実装しており、現状のハードウェア的な制約や、コンピューティングリソースの制約からくる限界にも目を向ける。

1.4 本論文の構成

本論文ではマルチコンテキストな没入環境を実現するため特に基礎となる、基盤システム全体の設計方針（第 2 章）・3D アプリケーションのレンダリング手法（第 3 章）・既存の 2D アプリケーションの利用手法（第 4 章）それぞれについて既存技術や類似研究と比較しながら本システムで提案する手法を示し、その評価と今後の課題・まとめを述べる。最後に第 5 章でマルチコンテキストな没入環境を実現するという観点から全体をまとめる。

^{*3} Adapted from Google. “Google Earth VR” <https://arvr.google.com/earth/> (accessed on 19 Dec, 2022)

^{*4} Adapted from VRChat Inc. “PRESS KIT - VRChat” <https://hello.vrchat.com/press> (accessed on 19 Dec, 2022)

第 2 章

提案するシステムの概要

2.1 はじめに

本章では次章以降の研究課題解決を述べるにあたり、システムの全体設計とそれを採用した理由を既存研究と比較しつつまとめる。

1.2 で述べたように、現在の没入環境のパラダイムでは、2 次元のデスクトップ環境とは異なり、基本的に 1 つのアプリケーションがユーザの環境全体を支配するため、マルチコンテキストな没入環境は実現できていない。このため没入環境でも複数のアプリケーションを同時に利用できるようにすることが、マルチコンテキストな没入環境を実現する基本的な要件であると考えられ、本論文で提案するシステムでも複数の 3D アプリケーションが没入環境で同時に利用できることを基礎的なシステム要件の 1 つとする。ここでの没入環境の 3D アプリケーションとは、ユーザの視野全体を支配するようなものではなく、机や時計といった空間の一部を占有するような新しいパラダイムのアプリケーションである。

次節以降ではまず、2.2 で 2 次元のデスクトップ環境でマルチコンテキストな空間を作り出している重要な技術要素である Windowing System について、その特徴をマルチコンテキストという視点からまとめる。次に 2.3 で、関連する研究や他の技術要素について触れ、最後に 2.4 で本論文の結論として採用したシステムの概要と、システムが提供する機能のスコープを定義する。

2.2 Windowing System

Windowing System とは、2 次元のデスクトップ環境で複数のアプリケーションを 1 つのスクリーン空間で同時に用いるための仕組みである。さまざまなオペレーティングシステムでの実装があり、オープンソースな実装としては X.Org^{*1} による X Window System[6] や Wayland^{*2} などがある。Windowing System は 2 次元のデスクトップ環境において、ブラウザやメールクライアントなど複数のアプリケーションを同時に利用することを可能にしている

^{*1} X.Org Foundation. “X.Org” <https://www.x.org/wiki/> (accessed on 24 Jan, 2023)

^{*2} Wayland “Wayland” <https://wayland.freedesktop.org/> (accessed on 24 Jan, 2023)

が、アプリケーション間の競合なくこれを実現するための様々な工夫がある。

まず、Windowing System では、サーバ・クライアントモデルを採用しており、複数のアプリケーション（クライアント）と、それらとコミュニケーションを行ってウィンドウをスクリーンに実際に表示するコンポジッタ（サーバ）から構成される。アプリケーションはそれぞれプロセスとして起動でき、Inter-Process Communication (IPC) によってコンポジッタとコミュニケーションをとる。このためアプリケーションどうしはオペレーティングシステムによってメモリ空間の分割や、CPU のスケジューリング、セキュリティ上の分離などを行ってもらえ、基本的なアプリケーションどうしの競合を防いでいる。また、2 次元のデスクトップ環境では 1 つのスクリーン空間上に複数のアプリケーションがウィンドウと呼ばれる長方形領域が重なる形で表示される。ウィンドウの位置や大きさは基本的にアプリケーション側で決めるのではなく、ユーザが調整できるため、アプリケーションの空間的な競合をユーザ自身が最小限にできる。さらに、ユーザはマウスなどのポインティングデバイスでスクリーン上のカーソルを操作し、カーソルを介してアプリケーションとのインタラクションを行う。入力に関するこのプロトコルはカーソルが重なっているウィンドウのみにカーソルのイベントが伝達されるという制約ゆえに、1 つのポインティングデバイスの入力イベントを適切にアプリケーションに割り振っており、ユーザの意図しないアプリケーションが入力を受け取ったり、複数のアプリケーションが入力を同時に受け取ってしまうといった、入力の競合を最小限にしている。Windowing System ではこういった競合を防ぐ工夫によって、それぞれ全く無関係のアプリケーションをどのように組み合わせて用いても、ある程度使いやすくデスクトップ環境を利用できるようにしている。マルチコンテキストな空間を実現するためには、市場にある様々なアプリケーションの中から、ユーザが自由に選んで、その時々のコンテキストに合わせて恣意的にアプリケーションを組み合わせられる必要があるため、このアプリケーション間の競合を防ぐ仕組みは、マルチコンテキストな没入環境を考えるうえでも重要である。

2.3 類似研究・技術

初めてマルチコンテキストな没入環境を実現するシステムが提案されたのは 1997 年の Tsao らによる CRYSTAL[7] である。CRYSTAL は X Window System に強い影響を受けており、サーバ・クライアントモデルに近い、MasterSynchronizer とそれぞれ機能を持ったモジュールからなるシステム（図 2.1）を提案している。ここではモジュールがアプリケーションと近い意味を持つが、アプリケーションだけではなく、ヘッドトラッキングなど、ハードウェアから提供されるデータもモジュールとして抽象化されている。単一マシン向けであり、モジュールと MasterSynchronizer とのコミュニケーションは共有メモリとセマフォを用いた独自のプロトコルで効率的に行っている。アプリケーションには 2 次元デスクトップ環境におけるウィンドウのメタファーとして 3D Crystal（図 2.2）と呼ばれる直方体領域が割り当てられており、このようなメタファーは後述する motorcar[8] や studierstube[9] などでも用いられている。

Schmalstieg らは AR の分野で、複数アプリケーションを表示することに加え、複数ユーザ・ホストマシンでの利用を想定したシステムである、Studierstube[9] を提案した。Studierstube

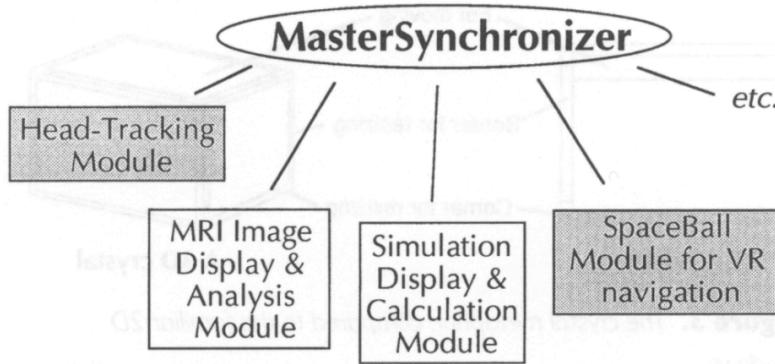


図 2.1. CRYSTAL のシステム概要図. MasterSynchronizer とそれぞれ機能を持ったモジュールからなる. 影のついた箇所はハードウェアコンポーネントである. (Adapted from Tsao and Lumsden 1997[7])

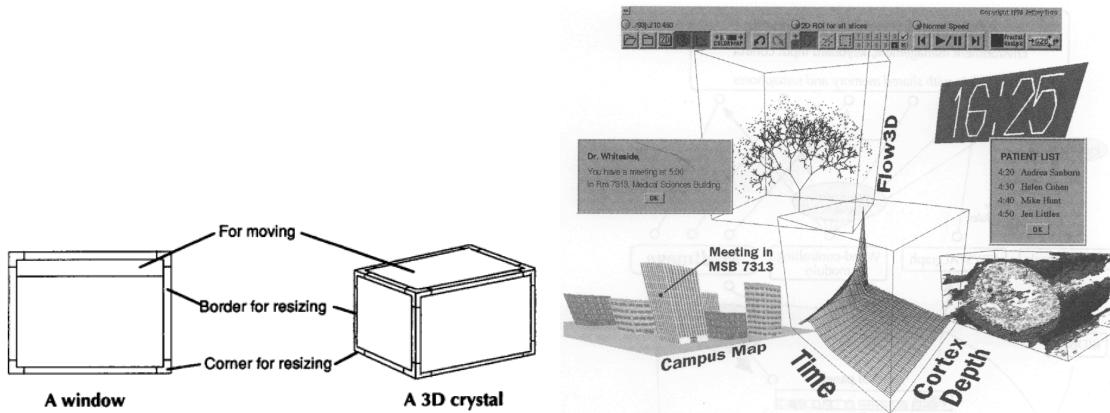


図 2.2. CRYSTAL で提案された 3D crystal の図. 左図はウィンドウとのメタファーを説明したもの. 右図は実際に用いられる状況を想定した図. (Both adapted from Tsao and Lumsden 1997[7])

では、システムに接続した複数のユーザがアプリケーションの状態や、現実空間での位置を共有しながら操作する仕組みを提案している。加えて、同時にアプリケーションの状態は共有しながら、その現実空間の位置はユーザごとに変えることも可能にすることで、遠隔地からのコラボレーションや、一箇所に集まりきれないほどの大人数でのコラボレーションにも対応できるようなマルチローケルなシステムであることも特徴である。また、Studierstube のアプリケーションはプロセスとして起動するのではなく、動的ライブラリとしてシステムにロードされ、システムのプロセスの中で呼び出されることも特徴である。

Reiling の提案した motorcar[8] では Linux 上で動く Windowing System である Wayland のプロトコルを拡張する形で 3D の Windowing System を提案した。Wayland は Linux で

広く使われていた X Window System の、Linux カーネルやハードウェアの進化と共に浮き出てきたパフォーマンス上の欠点などを解消する形で生まれた Windowing System である。Wayland では基本的な 2 次元の Windowing System を構成するのに必要なプロトコルが定義されているのに加え、自由にプロトコルを拡張できる仕様になっている。motorcar では、Wayland のこの特徴を用いて拡張したプロトコルに沿ってアプリケーションから 3D オブジェクトを表示するための情報を受け取り、複数のアプリケーションから受け取った情報を合成して、ヘッドマウントディスプレイ（HMD）に出力している。既存の 2 次元デスクトップの Windowing System の拡張であるため、Studierstube とは違い、シングルホストマシンにおいて複数の 3D アプリケーションを利用できるシステムである。また、既存の 2D Wayland アプリケーションとコミュニケーションを取ることで、既存の 2D アプリケーションを没入環境に 3D アプリケーションと一緒に表示できる。

近年の産業的な成果に目を向けても、マルチコンテキストに対応しようとする動きがみえる。特に既存の 2D のアプリケーションを没入環境に表示することで、没入環境のマルチコンテキスト化を図る例は多い。2 次元の PC の画面を没入環境で使えるようにするものとしては、Virtual Desktop^{*3} のような専用の VR アプリケーションや、Meta Horizon Workrooms^{*4} や Spatial^{*5} といった会議やイベント用の VR アプリケーションの中で PC 画面を投影するものなどがある。また、Microsoft の提供する Mixed Reality HMD である HoloLens^{*6} や Meta 社の提供する Meta Quest^{*7} では特定の形式の 2D アプリケーションを没入環境で複数起動できる仕組みがある。ただし、これらは没入環境自体をマルチコンテキスト化しているわけではなく、没入環境にマルチコンテキストな 2D デスクトップ環境を持ち込んでいると捉えられる。

さらに、VR や AR のシステムの標準化におけるデファクトスタンダードである OpenXR^{*8} では、Overlay と呼ばれる機能が定義されており、メインのユーザの視野全体を支配するアプリケーションに重ねて、他のコンテンツを表示できる仕組みを提供しており、VR ゲームをプレイしながら、ソーシャルメディアを時々チェックするといった、マルチコンテキストな没入環境の使い方を支援している。

2.4 本論文での結論

本論文での結論としては、Wayland のプロトコル拡張の機能を用いる Reiling の手法をシステム構成のベースとした（図 2.3）。システムの構成要素は、3D アプリケーションと Wayland 上で動作する既存の 2D アプリケーション、入力装置であるキーボードとマウス、没入環境

^{*3} Virtual Desktop. Inc. “Virtual Desktop” <https://www.vrdesktop.net/> (accessed 25 Dec, 2022)

^{*4} Meta. “Meta Horizon Workrooms” <https://www.meta.com/jp/work/workrooms/> (accessed 25 Dec, 2022)

^{*5} Spatial Systems, Inc. “Spatial” <https://www.spatial.io/> (accessed 25 Dec, 2022)

^{*6} Microsoft “HoloLens” <https://www.microsoft.com/ja-jp/hololens> (accessed 25 Dec, 2022)

^{*7} Meta “Meta Quest” <https://www.meta.com/jp/quest/> (accessed 25 Dec, 2022)

^{*8} Khronos group “OpenXR” <https://www.khronos.org/openxr/> (accessed 25 Dec, 2022)

8 第2章 提案するシステムの概要

を提示する HMD, それらとコミュニケーションをとって制御する 3D コンポジッタである。3D コンポジッタはマウスやキーボードの入力を適切にアプリケーションに分配し, 2D/3D アプリケーションは没入環境に描画したい内容を 3D コンポジッタに伝える。3D コンポジッタは複数のアプリケーションから受け取った情報を合成して HMD に表示することで, ユーザに複数アプリケーションが利用可能な没入環境を提示できる。

本論文が扱うのは 3D アプリケーションとコンポジッタとの間のやり取りのプロトコル, また 3D コンポジッタの設計・実装の部分である。2D アプリケーションとコンポジッタとの間のプロトコルは既存のものが存在しており, 本論文のスコープの範囲ではないが, 他の部分の設計において重要な要素であり, 詳しくは 4 章で扱う。

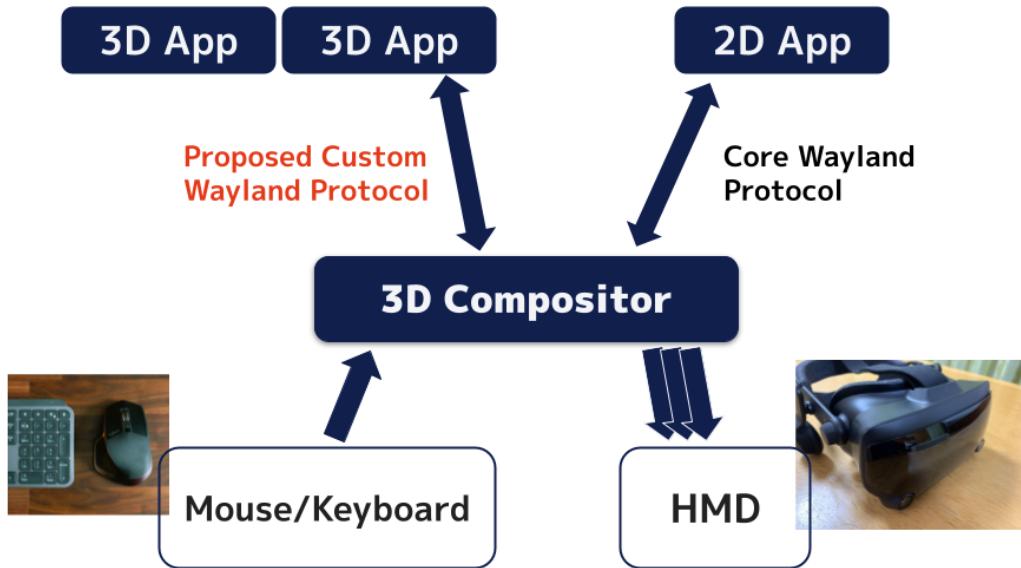


図 2.3. 提案するシステムの概要図。

類似研究・技術と比較してこの設計を選択した理由は以下の通りである。

- マルチコンテキストな没入環境を社会実装していくうえで大きな課題となるのは, 初期の 3D アプリケーションの少なさである。ユーザの視野全体を覆うというパラダイムのアプリケーションから, 空間の一部を占有するアプリケーションへと変えようとしているため, 既存の 3D アプリケーションを少し改変して対応できるわけではない。しかし豊富に存在する既存の 2D アプリケーションを没入環境で使うことができれば, 3D アプリケーションは, 例えば 3D モデルを見るだけといった, 補助的な役割から担っていくことができる。このため既存の 2D アプリケーションを利用できるようにすることは大変重要であると考える。CRYSTAL や Studierstube, OpenXR の Overlay などは 2D アプリケーションを表示する仕組みとは全く異なる仕組みを用いているため, 既存の 2D アプリケーションを利用することは不可能, または制限がかかることになる。一

方、本論文が採用したシステムでは、2D アプリケーションと直接やり取りするコンポジッタを自由に設計し直せるため、没入環境で 2D アプリケーションを最大限利用できる。その詳細は 4 章で述べる。

- 本システムは単一ホストマシンでの利用のみをスコープとしており、Studierstube のような複数のホストマシンでの多人数の利用に関してはスコープに入れていない。どのように多人数でのコラボレーションを実現するかは、提供するサービスの特性と深く結びつき、システムで規定することは現実的でないと考えられるからである。例えば、ユーザの間で同期されるべきデータについても、3D モデルを共同編集するようなアプリケーションであれば、メッシュやテクスチャのデータを同期しなければいけないが、同時にある動画を見るアプリケーションであれば、同期するべきデータは動画の ID と再生時間のみで良い可能性がある。その他にも同期の頻度やタイミング、操作や閲覧の権限管理をするなどは個別のアプリケーションによって変わりうる。また、サーバなどインフラやその他の機器を必要とする場合もありえる。このため個別の 3D アプリケーション提供者が自身のビジネスとしてサーバなどを介して、サービスとして多人数コラボレーションを実現するのが現実的だと考えられる。
- 1.1 で述べたように、没入環境を用いた 3D アプリケーションは様々な分野で応用されており、単に没入環境に PC の画面を投影するシステムでは没入環境を最大限利用できていない。

第3章

レンダリングプロトコル

3D アプリケーションとコンポジッタとの間のプロトコルの中でも特に重要であるのが、アプリケーションの表示に関するレンダリングプロトコルである。ここでも様々なアプリケーションを組み合わせて用いたときに、他のアプリケーションへの影響（競合）を最小限にしてレンダリングができるようにする必要がある。ここでレンダリングプロトコルに求められるものとして以下の指標を決めた。

- 複数のアプリケーションが別々に提示してきたレンダリングの情報を合成して、1つの没入環境に矛盾なくレンダリングできる。
- 質の悪いアプリケーションが他のアプリケーションや環境全体に影響を及ぼさない。複数のアプリケーションを利用できる環境下では、質の悪いアプリケーションがあり、フレームレートにレンダリングが追いつかなかったり、ハングしてしまう可能性がある。このようなときにシステム全体への影響を最小限にする必要がある。
- レンダリングの効率がよい。
- レンダリングの自由度が高く、様々な表現が可能である。

3.1 3D コンピュータグラフィックス

この節では以降の説明のために最小限必要な、3D レンダリングにおけるレンダリングパイプラインの概要を述べる。ただしレンダリングの手法は様々であり、ここでは代表的な手法についてのみ述べている。

3D オブジェクトのレンダリングは基本的に、3D 空間で定義されたオブジェクトを、ある視点から見た時の 2 次元平面でのピクセルマップに落とし込む行為である。このプロセスは大きく、Geometry Processing と Rasterization の 2 つのフェーズに分けることができる。

Geometry Processing

3D オブジェクトの形は多くの場合、メッシュによって表現され。メッシュは三角形などの Primitive の集まりとして表現される。レンダリングパイプラインではこの Primitive ごとに、それが Screen のどのピクセルを何色で塗りつぶすかを計算する。Geometry Processing のフェーズ（図 3.1）ではこの Primitive を構成する頂点の座標（Object Space）を行列計算によって、ある視点から投影したときの座標（Clip Space）へ変換するフェーズである。Primitive の頂点はまず、Model Matrix をかけることで、World Space のどこに配置するかが決定される。さらにこれに、視点の位置や向き、視野角などから取得される View Projection Matrix を乗算することで、頂点は Clip Space に配置される。View Projection Matrix は視点の位置と向きを表す View Matrix と、視野角などから計算される投影の変換を表す Projection Matrix との乗算で表される。

これらの操作は Vertex Shader によって制御される。OpenGL^{*1} のシェーダ言語である GLSL における簡単な Vertex Shader の例をソースコード 3.1 に示した。ここで model, view_projection は Uniform 変数などと呼ばれ、その値はレンダリングパイプラインを実行するときに指定できる。

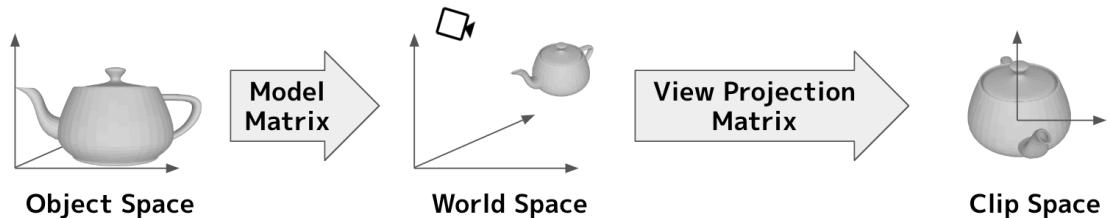


図 3.1. Geometry Processing.

ソースコード 3.1. Vertex Shader の例

```

1 uniform mat4 model;
2 uniform mat4 view_projection;
3 layout(location = 0) in vec4 position;
4
5 void main() {
6     gl_Position = view_projection * model * position;
7 }
```

^{*1} Khronos Group "OpenGL" <https://www.opengl.org/> (accessed 27 Dec, 2022)

Rasterization

Clip Space に変換された Primitive は Rasterization のプロセスによって、スクリーン上のどのピクセルをどの色で塗るかといった情報にまで落とし込む。このピクセルごとの情報はフラグメントと呼ばれ、一般的にフラグメントには色の情報と、そのフラグメントに投影された Primitive が視点からどれくらい離れているかを指す深度情報が含まれる。この Rasterization のフェーズでは Fragment Shader やテクスチャなどをプログラマが指定する。

Framebuffer と Depth Testing

Framebuffer とはレンダリング結果を格納するバッファである。Framebuffer はピクセルごとの色の情報を保持する Color buffer と深度の情報を保持する Depth buffer を含むことが多く、Rasterization によって得られたフラグメントは Framebuffer の Color buffer と Depth buffer に格納される。Rasterization によって Primitive がフラグメントに変換され、そのフラグメントが次々と Framebuffer に書きこまれていくが、このとき Depth Testing を用いると、そのフラグメントの深度情報と書き込み先の Framebuffer が持つ深度情報を比較して、フラグメントが手前にあるときだけ、Framebuffer に書き込むことができる。これによって、Primitive の処理順に関係なく、重なっている Primitive の前後関係をレンダリングに反映させることができる。

レンダリングパイプライン

今まで述べたプロセスはレンダリングパイプラインとして、GPU で行われる。ただしレンダリングパイプラインを実行するために必要なデータはプログラマによって CPU とメインメモリで作成され、OpenGL などの API を通して GPU に送られるのが一般的である。レンダリングパイプラインへの入力の例としては以下のものがある。

- メッシュに関するデータ：頂点の位置情報やそれに付随するデータなど。
- テクスチャ
- Model Matrix や View Matrix, Projection Matrix の値
- Vertex / Fragment Shader (ソースコードの文字列などの形式)

3.2 関連技術・研究

本システムはそれぞれ別プロセスのアプリケーションがレンダリングしたい情報を持っており、結果的に 1 つのレンダリング結果を導くという点で Parallel Rendering の分野と類似性がある。Molnar らは Parallel Rendering の手法を “sort-first”, “sort-middle”, “sort-last” の大きく 3 つに大別した [10] (図 3.2)。3.1 で述べたとおり、3D レンダリングは Primitive

がどのピクセルに影響を与えるかを計算するタスクであり、Primitive をスクリーンへと sort (選別・分配) するタスクであると捉えられる。Parallel Renderingにおいてこの sort のタスクでは、プロセッサ間で Primitive やフラグメントに関するデータを再配置する必要がある。Molnar らはこの再配置のタイミングに注目しており、“sort-first” は Primitive を Geometry Processing の際に再配置する手法、“sort-middle” は Geometry Processing 後に Primitive を再配置する手法、そして “sort-last” は Rasterization の際にフラグメントなどを再配置する手法である。

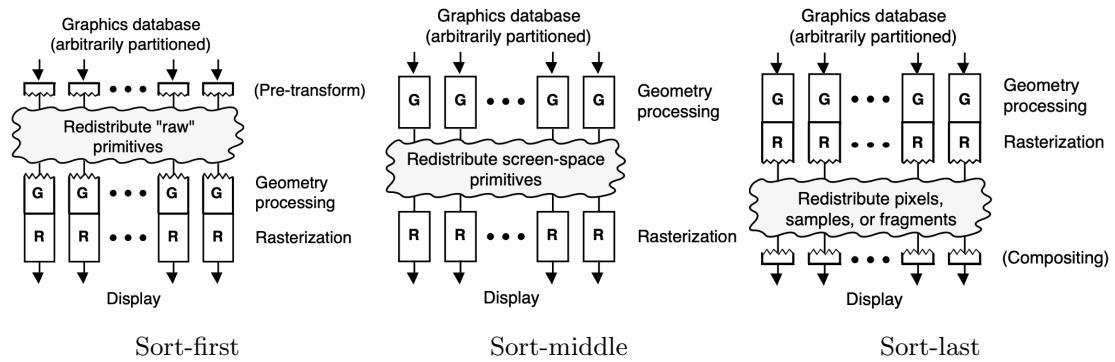


図 3.2. Molnar らによって分類された Parallel Rendering の 3 つの手法。 (Each figure is adapted from Molnar, et al. 1994[10])

本研究では、複数のアプリケーションがコンポジッタに対してレンダリングに必要な情報を伝達して、それを用いてコンポジッタ側が最終的なレンダリングを行うため、Molnar らの分類において、データの分配のタイミングまでをアプリケーションが担い、分配後をコンポジッタが担うと捉えられる。ただし Molnar らの設定とは大きな違いがあり、Molnar らの論文ではスクリーンはいくつかの領域に分割され、その領域ごとにプロセッサが割り当てられてレンダリングを行う設定であったため、どのプロセッサに Primitive やフラグメントを再配置するかを知るために、それらがスクリーンのどの領域に影響を与えるものであるか計算できている必要があった。一方本研究の設定では、アプリケーションがデータを作成しコンポジッタで合成するためにデータの再配置（アプリケーションからコンポジッタへの描画情報の伝達）をおこなう。このため、Primitive やフラグメントがどの領域に sort されるかといった計算は不要であるが、Molnar らの論文での洞察や検証は本研究にも応用できる部分が多い。

Reiling の提案した motorcar[8] は “sort-last” に近い手法(図 3.3)をとっている。motorcar ではそれぞれのアプリケーションがレンダリングパイプラインを実行し、生成したフレームバッファをできるだけ効率のよい方法でコンポジッタに送り、Depth Testing によってそれぞれのアプリケーションのフレームバッファどうしを合成している。motorcar の手法の特徴は以下のとおりである。

矛盾のないレンダリングについて

Depth Testing によって、複数のアプリケーションを矛盾なく合成できる。

質の悪いアプリケーションについて

質の悪いアプリケーションがフリーズしてしまった場合などは、オブジェクトが消えてしまうか、視点の情報を更新できず、描画の整合性が失われてしまう。

レンダリングの効率について

複数のアプリケーションが GPU を共有するため、プロセス間で GPU を奪い合い、コンテキストスイッチにオーバヘッドがある。また、Molnar らが指摘しているとおり、“sort-last” の手法ではコンポジッタに対して毎フレーム全てのフラグメントの情報を送る必要があり、データの転送量が多くなってしまう。Reiling の手法では単一ホストマシン内でのデータ転送であるため、アプリケーションとコンポジッタで共有したバッファを用いることで時間がかかる GPU メモリへの読み書きを最小限に抑え、伝達効率を上げているが、Reiling 自身が言及されているとおり、バッファへの書き込みや読み出しの部分でオーバヘッドが存在する。

レンダリングの自由度について

アプリケーションがレンダリングパイプラインを全て実行できるため、描画の自由度が高い。ただし、他のアプリケーションのオブジェクトの形などの情報は取得できないため、他のアプリケーションの影を落とすといったことはできない。

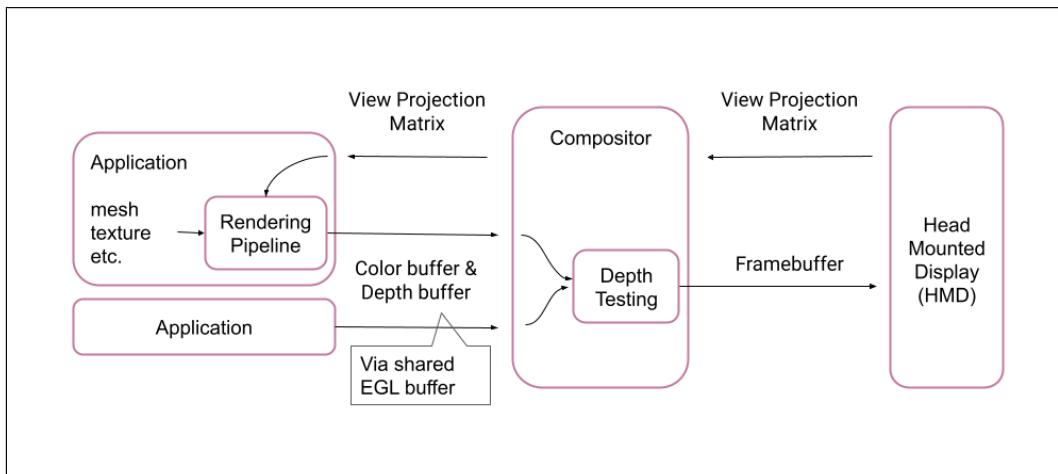


図 3.3. motorcar[8] でのレンダリング手法の概略図。

Peuhkurinen らは motorcar の手法におけるコンテキストスイッチのオーバヘッドを指摘し、アプリケーションとコンポジッタとの間で共有したシンググラフを用いる手法を提案した[11]。Peuhkurinen らはモバイル端末を用いた AR 環境で実験をしているため、没入環境ではないが、端末の位置や傾きをもとに、複数のアプリケーションを AR 環境にレンダリングしている点で、レンダリングに関しては本研究と同じ問題に取り組んでいる。Peuhkurinen らの手法（図 3.4）ではコンポジッタ側で 1 つのシンググラフを持ち、アプリケーションが IPC によってそのシンググラフに変更を加える。コンポジッタでは毎フレームそのシンググラフを走

査し、そのタイミングで取得したデバイスの位置や傾きを使ってシーンをレンダリングする。Peuhkurinen らの手法の特徴は以下のとおりである。

矛盾のないレンダリングについて

システムで共有するシンググラフからレンダリングすることで複数のアプリケーションを矛盾なくレンダリングできる。

質の悪いアプリケーションについて

質の悪いアプリケーションがフリーズした場合でも、シンググラフの更新が途絶えるだけで、フレームごとにデバイスの位置や傾きを更新して、矛盾なくシーンをレンダリングできる。

レンダリングの効率について

レンダリングは全てコンポジッタ側で行われるため、コンテキストスイッチのオーバヘッドがなく効率的である。

レンダリングの自由度について

シンググラフの設計によってレンダリングできるものが限られてしまい、レンダリングの自由度は著しく落ちる。特に Peuhkurinen らが提案したシンググラフを構成する要素は以下の 5 つであり、より高度なレンダリングを実現する提案までは至らなかった。

1. **Mesh** 位置を表す (x,y,z) と頂点の法線ベクトルの (x,y,z) , UV マッピングのための (u,v) の情報を持つ。
2. **Texture** 8bit ずつの RGBA コンポーネントで構成される。
3. **Transformation Matrix** アプリケーションのローカル座標系における Mesh の平行移動、回転、スケールを表す 4×4 の変換行列。
4. **Node** Mesh, Texture, Transformation Matrix から構成され、コンポジッタによって描画される単位。
5. **Application Volume** アプリケーションが Node を描画できる直方体領域。

3.3 提案手法

motorcar の手法では、レンダリングの自由度は高いが、レンダリングの効率や質の悪いアプリケーションが存在する場合において問題があった。一方共有シンググラフを用いた Peuhkurinen らの手法では、レンダリング効率や、質の悪いアプリケーションに関しては有効だが、レンダリングの自由度が制限されていた。本システムでは “sort-first” に近い手法を提案し、レンダリングの効率や質の悪いアプリケーションの問題を解決しつつ、Peuhkurinen らの手法よりレンダリングの自由度を高くする。

本システムでのレンダリング手法の基本的な方針は、シンググラフといった抽象化を行うと

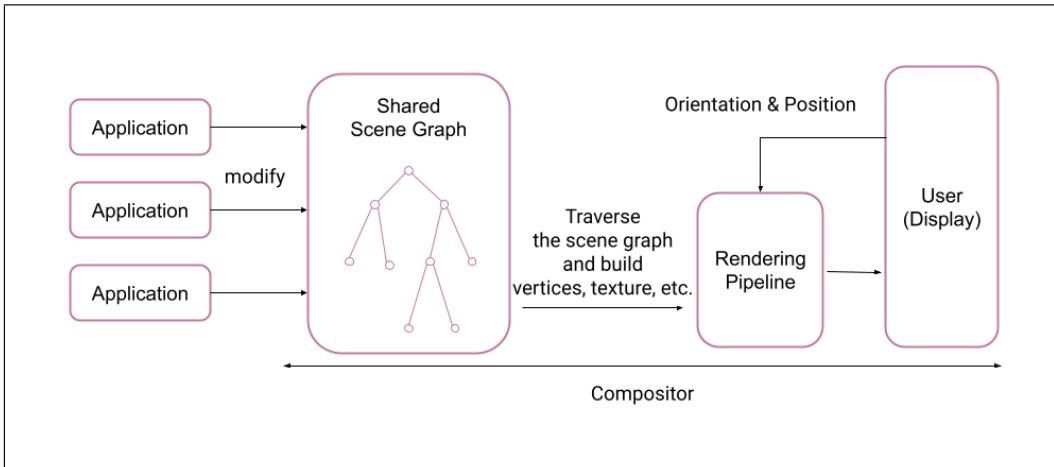


図 3.4. Peuhkurinen ら [11] の提案するシステムのレンダリング手法の概略図.

その分レンダリングの自由度が落ちてしまうため、できるだけアプリケーションがレンダリングパイプラインをそのまま使うことができるようになることである。そこで本システムでは広く使われているグラフィックスライブラリである OpenGL をベースとしたレンダリングプロトコルを作成した。今回は特に OpenGL ES 3.2 ^{*2} をもとにしている。

OpenGL では以下のようなオブジェクトが定義されており（一部）、これらのオブジェクトといくつかのパラメータを指定してレンダリングパイプラインを実行することで、指定した Framebuffer にレンダリング結果が格納される。

- **Buffer Object** 頂点データや頂点の配列順のデータなどを格納する汎用的なバッファ
- **Shader Objects** Vertex Shader や Fragment Shader などを格納するオブジェクト
- **Program Objects** レンダリングパイプラインの各ステージで用いられる Shader Object をひとまとめにするオブジェクト
- **Texture Objects** テクスチャを格納するオブジェクト
- **Sampler Objects** テクスチャのサンプリングに関するパラメータの集合を保持するオブジェクト
- **Vertex Array Objects** Vertex Shader で用いる頂点情報がどのバッファのどこに、どのような形式で存在するかを指定するオブジェクト

本レンダリングシステム（図 3.5）ではまず、アプリケーションが IPC を用いて上記の 6 つのオブジェクト（Rendering Resource）をコンポジッタ側で作成するプロトコルを用意した。また、レンダリングパイプライン実行時に指定するオブジェクトとパラメータをまとめた Rendering Unit というオブジェクトを新たに定義した。Rendering Unit は 1 回のレンダリングパイプラインの実行に相当する。そして、アプリケーションが Rendering Unit をコンポ

^{*2} Khronos Group "OpenGL ES Version 3.2" https://registry.khronos.org/OpenGL/specs/es/3.2/es_spec_3.2.pdf (accessed 29 Dec, 2022)

ジッタ側に作成し、Rendering Unit に対して Rendering Resource の紐付けと、パラメータの指定をできるようにした。Rendering Resource は複数の Rendering Unit に対して紐づけることが可能である。また、Rendering Unit に対して指定可能なパラメータには、シェーダから利用可能な Uniform 変数の値などを含む。

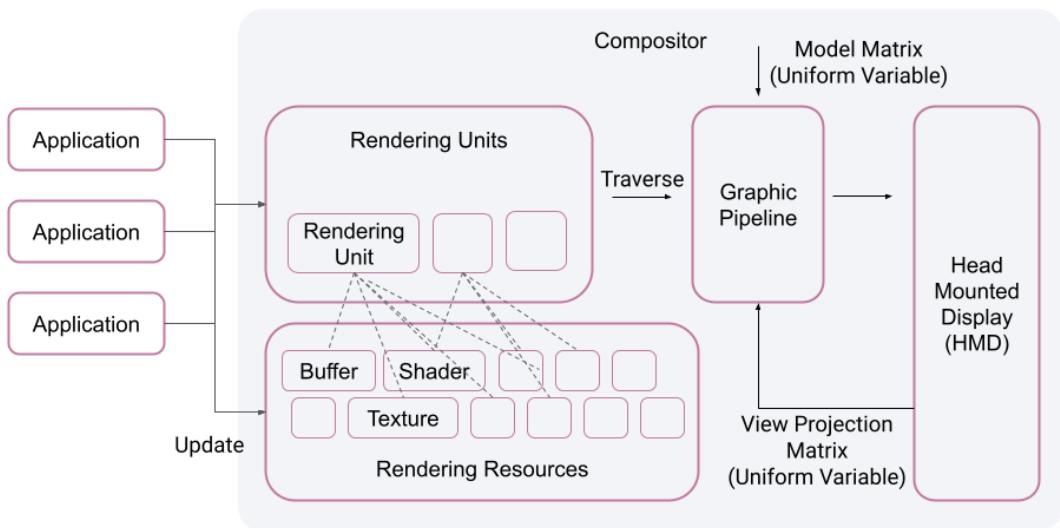


図 3.5. 本システムのレンダリング手法の概要図。

上記のようなシステムにすることで、アプリケーションは頂点データの形式、バッファへの格納方法、テクスチャの形式、そしてシェーダを自由に設定、記述でき、レンダリングの自由度を高く保つことができた。また、レンダリングパイプラインの実行自体は全てコンポジッタ側で行われるため、GPU のコンテキストスイッチが少なく効率的である。一般に Texture や Buffer, Shader などのサイズの大きいオブジェクトはその生成やフォーマットの修正、転送のコストも無視できないが、本レンダリングシステムではアプリケーションは共有メモリを介してそれらのオブジェクトをコンポジッタと共有し、コンポジッタはそのデータを再形成することなくそのまま GPU 上のメモリに転送できるため、転送や再形成のためにオーバヘッドがかかることはない。最後に、質の悪いアプリケーションに対応するための工夫を述べる。

2.2 で述べたとおり、マルチコンテキストな没入環境を実現するためには、アプリケーション間の競合を防ぎ、適切に分離することが大切である。そのため質の悪いアプリケーションがフリーズしたり、動作が遅くなった場合でも、そのアプリケーションをレンダリングする際の視点情報が更新される必要がある。そうでなければ、アプリケーションのレンダリング情報が途絶えた場合に、そのアプリケーションのレンダリングをやめるか、最後のレンダリング結果を表示し続けることになる。前者の場合はチラつきが発生し、後者の場合は現在の視点と異なる視点でレンダリングした結果を表示するため、整合性が失われ、酔いの原因になりうる。いずれにせよ、1 つのアプリケーションが環境全体にあたえる影響が大きい。視点情報が更新さ

18 第3章 レンダリングプロトコル

れ続ければ、フリーズしたアプリケーションがレンダリングするオブジェクトは没入環境の中では静止するが、描画の整合性が失われることではなく、他のアプリケーションは通常通り利用できる。

また、本システムではアプリケーションの位置をコンポジッタ側が決定する。これはユーザがアプリケーションを好きな位置に配置し、アプリケーションどうしの空間的な競合をユーザが最小限にできることを保証するためであるが、フリーズしたり、動作が遅いアプリケーションでもその配置を変更できるようにする必要がある。そのため質の悪いアプリケーションがフリーズしたり、動作が遅くなったりした場合でも、アプリケーションの配置を決める Model Matrix が更新可能である必要がある。

以上の議論から、レンダリングする際に Model Matrix と視点の情報である View Matrix, Projection Matrix をいかに適用するかが重要である。3.1 で述べたように、通常 Model Matrix や View Matrix, Projection Matrix は Uniform 変数として用い、シェーダで頂点に対して適用することが多い。一般的な既存の VR アプリケーション（図 3.6）ではソースコード 3.1 のような Vertex Shader と、その Vertex Shader で用いられている Model Matrix や View Projection Matrix を示す Uniform 変数に入れる値を Graphics Pipeline に伝えることで、頂点に Model Matrix と View Projection Matrix を適用させている。

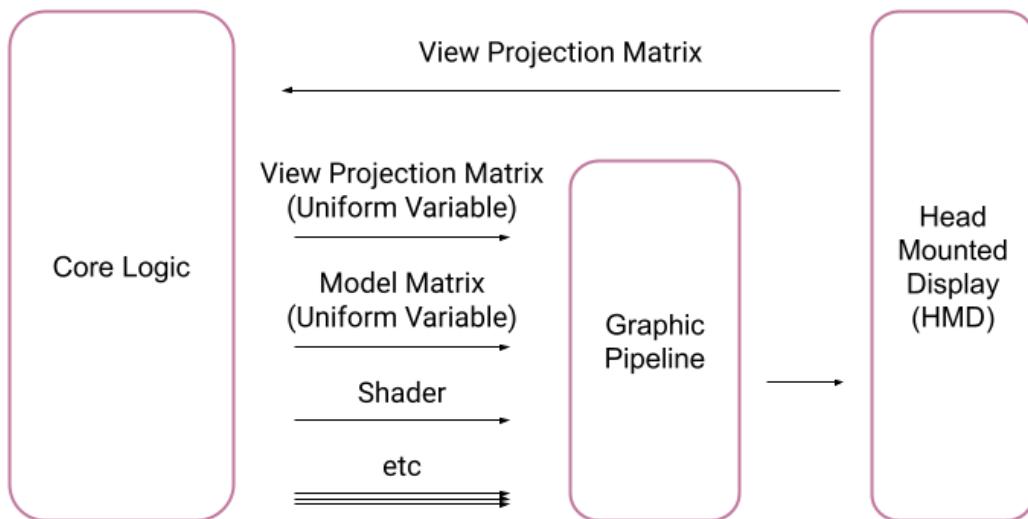


図 3.6. 没入環境での一般的なアプリケーションでの Mode View Projection Matrix の適応のさせ方の例。

本レンダリングシステムでは（図 3.5），Model Matrix はコンポジッタがアプリケーションごとに管理しているものを、View Matrix, Projection Matrix は HMD から取得したものをアプリケーションへ渡すことなく Graphics Pipeline 実行時に適用する。そのため本システムではシェーダで用いる Uniform 変数のいくつかを予約しており、例えば “zModel” という

名前の Uniform 変数には Model Matrix の値を, “zView” という名前の Uniform 変数には View Matrix の値を, “zViewProjection” という Uniform 変数には View Projection Matrix の値をコンポジッタ側で設定することとしている。アプリケーションはソースコード 3.2 のような Vertex Shader を記述することで、Model Matrix や View Projection Matrix の実際の値を知らないまま、用いることができる。

ソースコード 3.2. 本システムでの Vertex Shader の例

```

1 uniform mat4 zModel;
2 uniform mat4 zViewProjection;
3 layout(location = 0) in vec4 position;
4
5 void main() {
6     gl_Position = zViewProjection * zModel * position;
7 }
```

このように本システムでは、Model Matrix, View Matrix, Projection Matrix を表す Uniform 変数の値をコンポジッタ側が、それ以外のレンダリングパイプラインの実行に必要なオブジェクトやパラメータをアプリケーション側が提供し、コンポジッタ側でレンダリングパイプラインを実行する。これにより、レンダリングに関するオーバヘッドが小さく、レンダリングの自由度を高く保ち、かつ質の悪いアプリケーションに対して堅牢なレンダリングシステムを実現した。

3.4 評価：レンダリング性能

本節では、レンダリング性能に関して絶対的な評価をした。

3.4.1 実装・環境

プログラミング言語は C と C++ を用いた。アプリケーションとコンポジッタ間の IPC には Wayland を用いた。HMD への出力、HMD からの View Matrix, Projection Matrix の取得には SteamVR^{*3} ランタイムの OpenVR^{*4} を用いた。その他の環境に関しては表 3.1 に示す。また、今回の実装では FPS の上限は 90fps となっている。

3.4.2 実験の設定

本実験ではレンダリングの性能を確かめるため、いくつかのタイプの 3D アプリケーションを複数実行し、システム全体の FPS の変化を測定した。実験に用いたアプリケーションは以

^{*3} Valve Corporation “SteamVR” <https://store.steampowered.com/steamvr> (accessed on 24 Jan, 2023)

^{*4} Valve Corporation “OpenVR” <https://partner.steamgames.com/doc/features/steamvr/openvr> (accessed on 24 Jan, 2023)

| | |
|--------|---|
| OS | Ubuntu 20.04.4 LTS |
| Kernel | 5.13.0-41-generic |
| CPU | Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz |
| GPU | GeForce RTX 2070 SUPER |
| Memory | 32GB |
| HMD | Valve Index |

表 3.1. レンダリングに関する実験環境

以下の3つである。

Box App

Box App (図 3.7) は立方体を表示する、頂点の少ない簡単なアプリケーションである。ただし一面にテクスチャが貼られており、アプリケーションは毎フレームテクチャを更新する。テクスチャのサイズは 256x256 で 1 ピクセルあたりビット数は 32bit である。

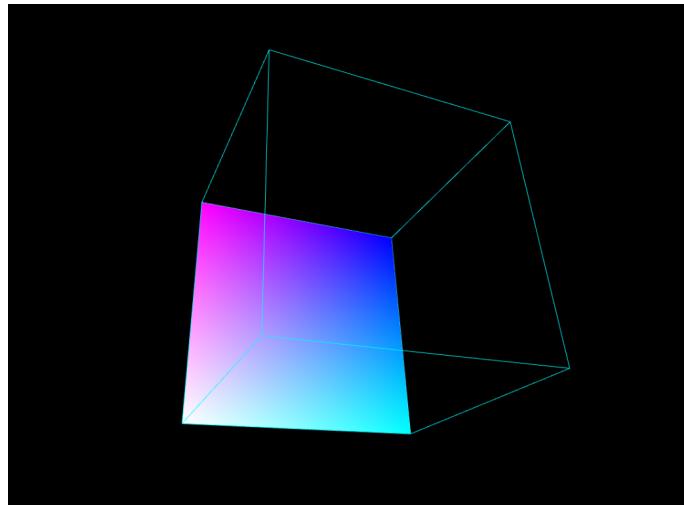


図 3.7. Box App. 頂点が少なく、テクスチャの更新がある。

STL App

STL App (図 3.8) は STL^{*5} 形式のファイルを表示するアプリケーションである。比較的頂点数が多いが、データの更新がない。本実験で用いた STL ファイルは頂点数が 324,000 個のものである。

^{*5} Wikipedia. “STL(file format)” [https://en.wikipedia.org/wiki/STL_\(file_format\)](https://en.wikipedia.org/wiki/STL_(file_format)) (accessed 17 Jan, 2023)



図 3.8. STL App. 頂点が多いが、データの更新がない。

STL App with rotation

STL App with rotation は STL App に加え、STL オブジェクトをヨー方向に回転させる。アプリケーションは回転量を表す Uniform 変数 (float, 32bit) を毎フレーム更新し、頂点シェーダで頂点の回転処理を記述している。

3.4.3 結果

上記の三種類のアプリケーションそれぞれについて、同時に起動するアプリケーションの数を増やしたときの FPS の変化を図 3.9 に示す。

いずれのタイプのアプリケーションでも 40 個ほどまでは 90fps 近くを保っており、本レンダリング手法が十分実用に耐えうることが示された。また、STL App や STL App with rotation のように、頂点データ数が多くても時間あたりに更新するデータが少ないアプリケーションであれば、さらに多くのアプリケーションをレンダリングできることがわかった。

STL App と STL App with rotation において、アプリケーション数が 90 個近くなったときに急激に FPS が落ち込むのは、頂点数が多く、レンダリングパイプラインの処理が 1 フレーム内に収まりきらなくなり、バッファのスワップが次のフレームタイミングで行われるようになつたためである。一方 Box App では比較的連続的に FPS が落ちるが、これはコンポジッタ側でアプリケーションから共有されたテクスチャを GPU へ送る部分に時間がかかるており、コンポジッタのメインループがビジー状態になっていることが原因である。

3.5 評価：レンダリングの自由度

この章では様々なレンダリング手法が本レンダリングシステムで適応できることを示し、レンダリングの自由度が高いことを示す。

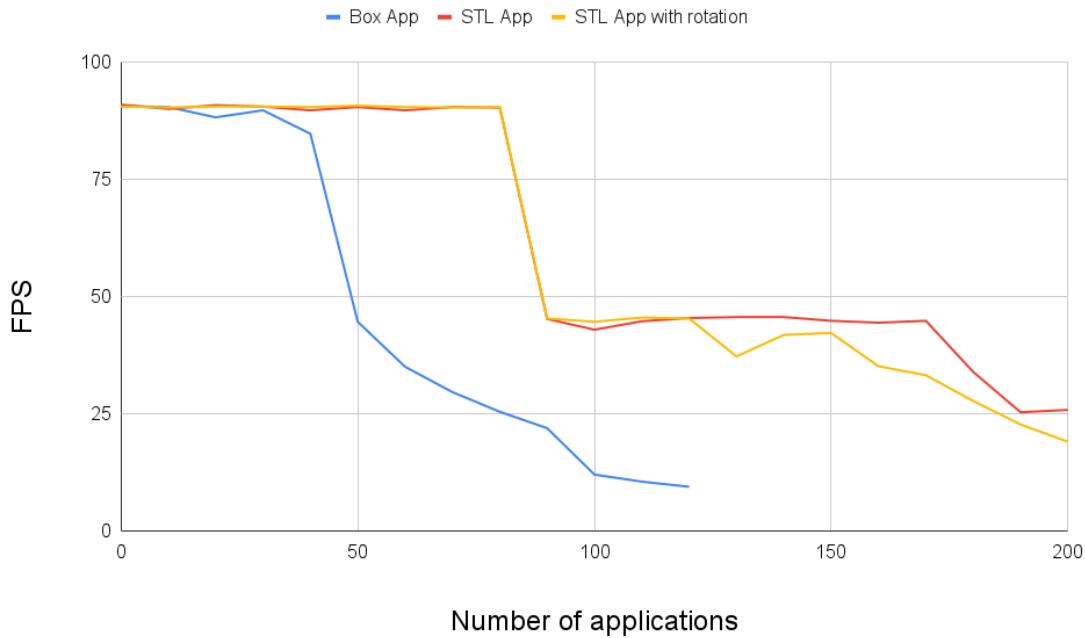


図 3.9. レンダリング性能測定実験の結果.

3.5.1 Fragment Shader での Ray Marching を用いたメタボール

Ray Marching はメッシュを用いずに Ray を走査することでレンダリングを行う技法の一つであり、数学的な幾何構造の可視化 [12] に用いられたり、より高度なレンダリング手法 [13][14] の基礎として用いられる。

メタボール [15] とはメタボールどうしが近づくことで、一定の規則に沿ってそれらのメタボールが融合し、ひとかたまりとなる球状のオブジェクトである。

ここでは Fragment Shader 内での Ray Marching の一種である Sphere Tracing[16] を実装し、本システムでメタボールを表現可能であることを確認した（図 3.10）。

3.5.2 Geometry Shader を用いた頂点の追加と変換

3.1 では主なシェーダとして Vertex Shader と Fragment Shader を説明したが、OpenGL ではその他にも Geometry Shader や Tessellation Shader などがあり、これらはレンダリングの効率化 [17] や、より高度な表現 [18] のために用いられることがある。

本システムでは OpenGL の API を踏襲しているため、これらのシェーダを利用してレンダリングを行うこともできる。ここでは本レンダリングシステムで Geometry Shader を用いて、メッシュにポリゴンを追加したり、頂点を編集したりできることを確かめた（図 3.11）。

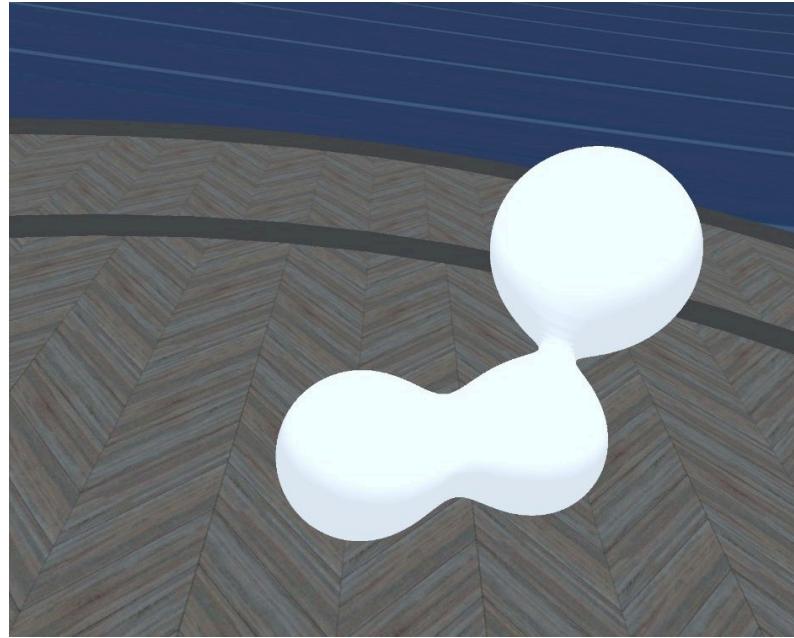


図 3.10. 本レンダリングシステムで Sphere Tracing を用いてメタボールをレンダリングする様子.

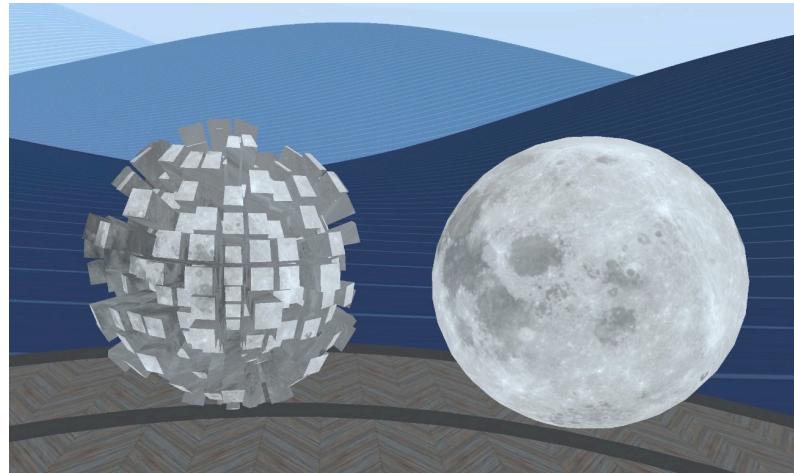


図 3.11. 図右の月のオブジェクトに対して, Geometry Shader を適応し, 球体内部のポリゴンの追加と, 頂点の編集を行い, 図左のオブジェクトをレンダリングする様子.

3.5.3 カラーマップ以外のテクスチャの利用

テクスチャマッピングは、メッシュ表面のベースカラーを表現する以外にも多くの利用方法がある。バンプマップ [19] ではテクスチャにメッシュの表面からの高さの変位をエンコードしている。面の法線ベクトルの微妙な変化を計算して、シェーディングに利用することで細やかな表面の表現が可能になる。また、テクスチャにメッシュ表面の法線ベクトルをエンコード

24 第3章 レンダリングプロトコル

ディングした法線マップを用いたレンダリング手法 [20] や、テクスチャにメッシュ表面の光沢の度合いをエンコーディングするスペキュラマップなどは一般によく用いられる。

本システムではテクスチャの利用の仕方も特定の方法に限定しないため、これらのようなテクスチャの利用方法も実現できる。ここでは本レンダリングシステムでバンプマップを用ることで、月面の細やかな法線の揺らぎを計算し、月面の凹凸による陰を表現できることを確認した。

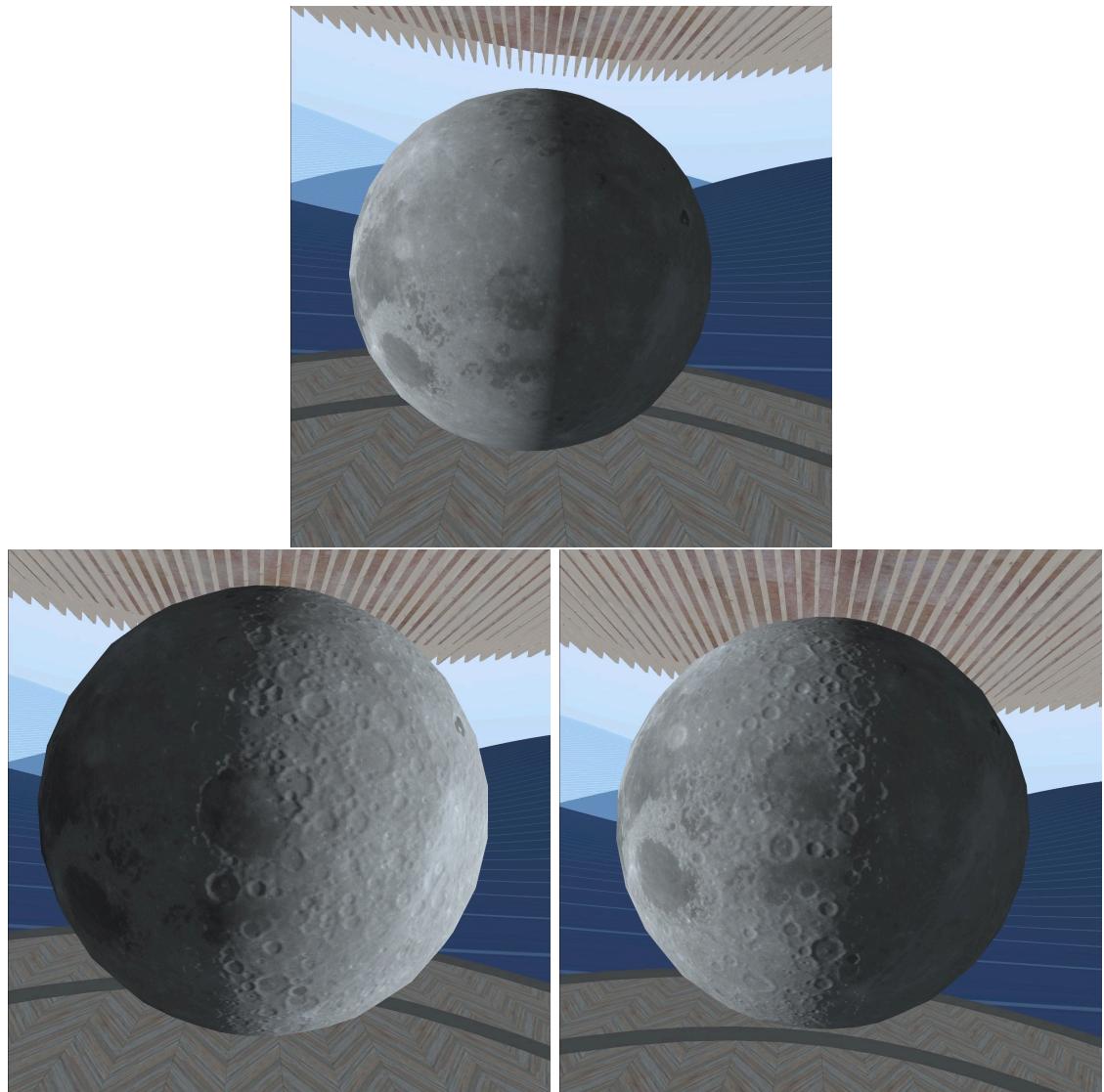


図 3.12. 本レンダリングシステムでバンプマップを適用し、月面の凹凸をレンダリングに反映させた様子。図上はバンプマップを適用しない場合。図下の左右はバンプマップを適用し、光源の位置を変化させた様子。

以上の3つのレンダリング手法はレンダリングパイプラインの機能を用いた様々なレンダリング手法のごく一部であり、これらをシーニングラフを定義する方法では定義しきれない。本レンダリングシステムは、シーニングラフという抽象化を行うのではなく、レンダリングパイプラ

インへの入力をそのままアプリケーションが制御できるようにしたことで、レンダリングの自由度が飛躍的に向上したことを確かめることができた。

3.6 まとめ

本システムでは没入環境でのウィンドウシステムにおいて、複数のアプリケーションが別々に提示してきたレンダリングの情報を合成して1つの没入環境に矛盾なくレンダリングする方法を検討した。OpenGLで定義されているオブジェクトと、一度のレンダリングパイプラインの実行に相当するRendering Unitをプロトコルとして定義することで、効率的かつ、自由度の高いレンダリングを可能にした。また、HMDなどから取得されるModel View Projection MatrixはUniform変数としてコンポジッタ側でレンダリングパイプラインに渡すことで、質の悪いアプリケーションに対してもロバストなシステムとなった。

本レンダリングシステムの限界としてはRendering Unitが一度のレンダリングパイプラインの実行に相当しており、マルチパスなレンダリング手法に対応できていないことが挙げられる。これを実現するには、Rendering Unitの出力を他のRendering Unitの入力に用いるといったレンダリングプロトコルの拡張を考える必要があるが、これは今後の課題としたい。

また、頂点のデータ構造などもそれぞれのアプリケーションで定義することになるため、他のアプリケーションの影を落とすといった、アプリケーションどうしのレンダリングの影響の及ぼし合いは困難である。影を落とすという点では、メッシュの形状などを別途のプロトコルで定義したり、Physically Based Rendering[21]に即しているglTF2.0^{*6}を踏襲したレンダリングプロトコルを定義するといったことが考えられるが、これも今後の研究課題としたい。

さらに、本レンダリングシステムの課題としてはアルファブレンディングが困難な点がある。アルファブレンディングは通常視点より遠いオブジェクトからレンダリングすることで達成するが、本システムではアプリケーションが直接視点情報を知り得ないため、オブジェクトを視点から遠い順にソートすることが難しい。Order Independent Transparencyの技法を用いることも考えられるがマルチパスが必要であったり、レンダリングコストが高かったりするため、ここにも研究の余地がある。

^{*6} The Khronos 3D Format Working Group. “glTF 2.0 Specification” <https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html> (accessed 23 Jan, 2023)

第 4 章

2D アプリケーションの利用

2.4 で述べたように、マルチコンテキストな没入環境を社会実装していくうえで、初期の 3D アプリケーションの少なさは大きな課題となる。世の中にはすでに有用で、洗練された 2D アプリケーションが数多く存在しており、アプリケーションそのものだけではなく、ライブラリやデザインシステム、設計パターンなど多くの知見と資産がある。既存の 2D アプリケーションがうまく利用できることは、マルチコンテキストな没入環境の発展において不可欠であり、本システムの重要な設計方針の 1 つである。

また、1.2 や 2.2 で言及したとおり、既存の 2D Windowing System 上で動くアプリケーションは、他のアプリケーションと連携してマルチコンテキストな 2D デスクトップ環境を実現しており、マルチコンテキストな没入環境においても、他の 2D アプリケーションや 3D アプリケーションと連携して利用されることが期待できる。

4.1 関連技術・研究

2D のウィンドウやディスプレイを没入環境に投影する取り組みはすでに多い。

2.3 で言及した Meta Horizon Workrooms や Virtual Desktop, Spatial といった商用アプリケーションは、Microsoft Windows^{*1} や macOS^{*2} といったプロプライエタリなオペレーティングシステムのウィンドウやディスプレイを没入環境に投影できる（図 4.1）。これらのアプリケーションはオペレーティングシステムが提供するディスプレイやウィンドウ単位での画面共有の仕組みを用いて、それを VR 空間に投影している。しかしこの画面共有を用いた手法では、実際の物理ディスプレイの数やサイズ、ウィンドウの配置に制約をうけたり、本来 Windowing System が提供するより高度な仕組みを使うことができない。本来 Windowing System が提供するより高度な仕組みの例としては、以下のようなものがある。

- ウィンドウのコンテンツ（ピクセルマップ）の中で変更があった領域だけを取得し、効

^{*1} Microsoft. “Meet Windows 11” <https://www.microsoft.com/en-us/windows/windows-11> (accessed 18 Jan, 2023)

^{*2} Apple. “macOS Ventura” <https://www.apple.com/macos/ventura/> (accessed 18 Jan, 2023)

率的に画面の変更を反映する仕組み

- アプリケーション側のアニメーションなどに用いられるリフレッシュレートを制御する仕組み
- ポップアップを表示する位置を制御する仕組み

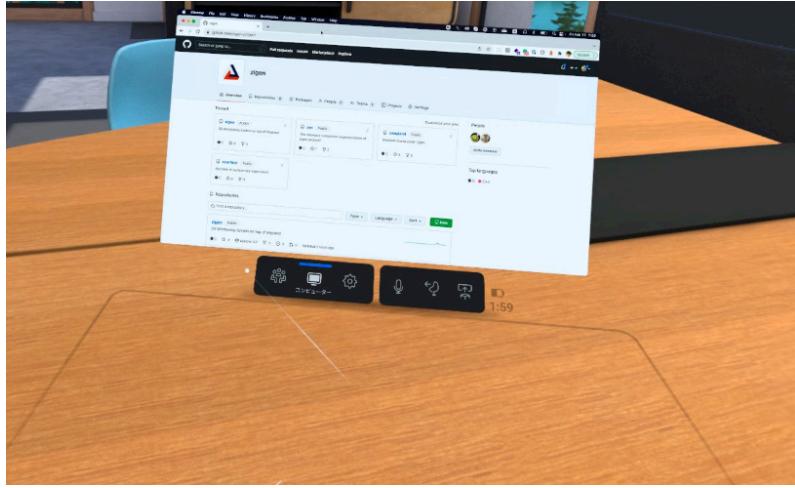


図 4.1. Meta Horizon Workrooms での PC 画面の共有機能.

これに対して、Reiling[8] が提案した没入環境向けのコンポジッタでは、既存の 2D Windowing System のプロトコルも実装することで、没入環境で既存の 2D アプリケーションが利用できるようにした（図 4.2）。

この手法では、コンポジッタが 2D アプリケーションと直接やり取りするため、既存の 2D アプリケーションを改変しないという制約のもとで、2D アプリケーションの表示や操作に関して最大限の自由度がある。

しかし、Reiling の提案では、3D アプリケーションを 2D アプリケーションと連携して使うための工夫については言及がなかった。3D アプリケーションと 2D アプリケーションとを連携して使うためには 3D アプリケーションのプロトコルの設計も工夫する必要がある。

4.2 提案手法

本システムでは、3D アプリケーションに対する操作に関するプロトコルを、既存の 2D の Windowing System と変換可能な形に設計することで、2D アプリケーションを 3D アプリケーションとより連携して使えるようにした。

2D の Windowing System ではカーソルを介してそれぞれのアプリケーションに操作を加えるが、本システムでは始点と方向ベクトルを持つ Ray を介してアプリケーションに操作を与えることにした。Ray による操作は没入環境では一般的だが、特に本システムでは Ray を定義することで、Ray とウィンドウが交差した位置にカーソルを定義でき、これを用いて Ray の操作をカーソルの操作に変換できるため重要である。

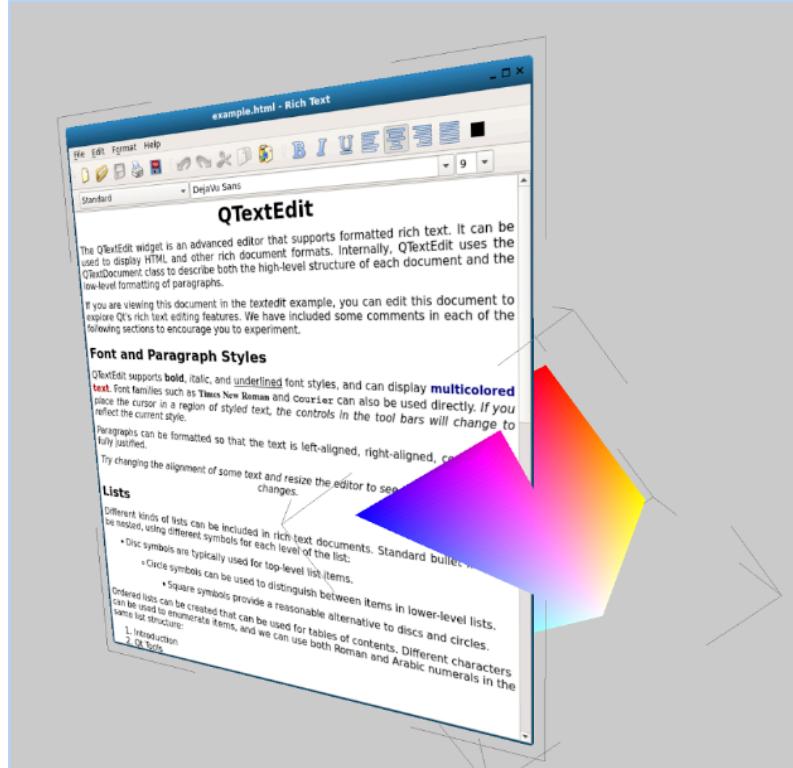


図 4.2. motorcar[8] での既存の 2D アプリケーションの表示. (Adapted from Reiling. “Defense Slides” (Google Slide) https://docs.google.com/presentation/d/1svgGMxxbfmcHy_KuS5Q9hah8PQOsXqvjBK0oMIzW24Y/edit?usp=sharing (accessed 18 Jan, 2023))

本システムでは 2D の Windowing System の操作に関するプロトコルのうち、カーソルに関する部分だけを Ray に置き換えたプロトコルを 3D アプリケーションの操作のために定義した。そして、3D コンポジッタと既存の 2D アプリケーションとの間に Ray の動きをウィンドウ上のカーソルの動きに変換するコンポーネントを入れた（図 4.3）。このコンポーネントの主な役割は没入環境の 3D 空間上にウィンドウを表す長方形の領域を定義し、既存の 2D アプリケーションから受け取った描画内容をその長方形領域に描画すると同時に、Ray とその長方形領域とが重なった位置にカーソルを定義して、Ray の操作をカーソルの操作に変換してアプリケーションに伝えることである。

これによって、2D アプリケーションと 3D アプリケーションを同じデバイスで、同じ Ray を介して操作できるようになる。また、本システムではドラッグ & ドロップに関するプロトコルも Ray を用いて再定義している。2D Windowing System におけるドラッグ & ドロップのプロトコルはイベント（コンポジッタからアプリケーションへの通信）とリクエスト（アプリケーションからコンポジッタへの通信）の複雑な組み合わせで実現されており、ドラッグ & ドロップの開始のリクエスト、データを送る側がどのような形式のデータで送ることができるかの候補（mime-type）を提示するリクエスト、送る側のアクションの意図（データのコ

ピーなのか、移動なのか)を提示するリクエスト、ドラッグ中のカーソルの位置を知らせるイベント、など20種類以上存在する。これらを全て3Dアプリケーション用に定義し直すことで、本システムでは既存の2Dアプリケーションと3Dアプリケーション間でのドラッグ&ドロップを実現しており、これはまさに本システムでしか実現できない特徴である。

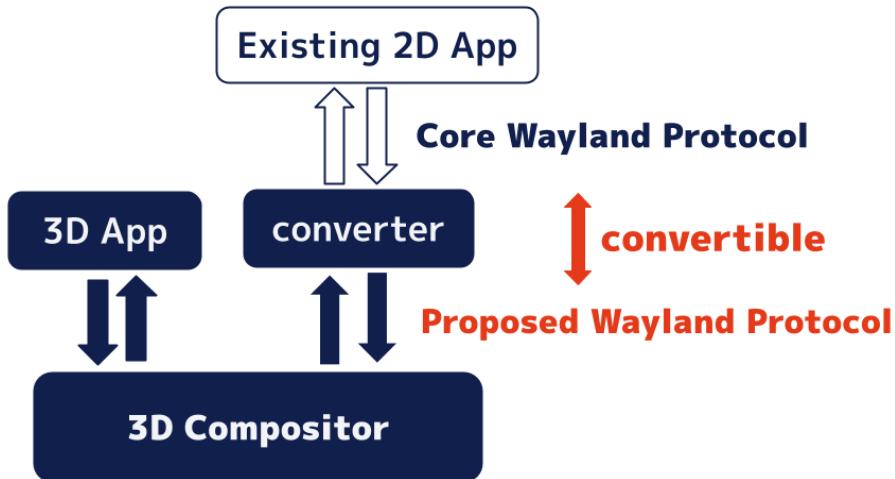


図4.3. 本システムでの既存の2Dアプリケーションを利用する仕組み。

4.3 評価

本システムにおいて2Dアプリケーションに関する設計が有効であることを確かめるために、サンプルのアプリケーションを用意した(図4.4)。1つは改変のない既存の2Dブラウザ(Google Chrome^{*3} 図4.4右側)、もう1つは地球などの天体を表示する3Dアプリケーション(図4.4左側)である。また今回はRayの操作にはマウスを用い、マウスの2次元の移動量をRayの方向ベクトルの変化にマッピングした。

まず、このRayを用いてブラウザに対してカーソルの移動、クリック、スクロールの基本的な操作が可能であることを確かめた。また天体のアプリケーションに本システムで定義したドラッグ&ドロップのプロトコルを実装し、ブラウザからのドラッグ&ドロップによって天体のテクスチャデータを更新できることを確かめた(図4.5)。

4.4 まとめ

既存手法では2Dアプリケーションを3Dアプリケーションと同時に没入環境に提示することは実現していたが、3Dアプリケーションと2Dアプリケーションとの協調の可能性までは

^{*3} Google. “Google Chrome” <https://www.google.com/chrome/> (accessed 18 Jan, 2023)

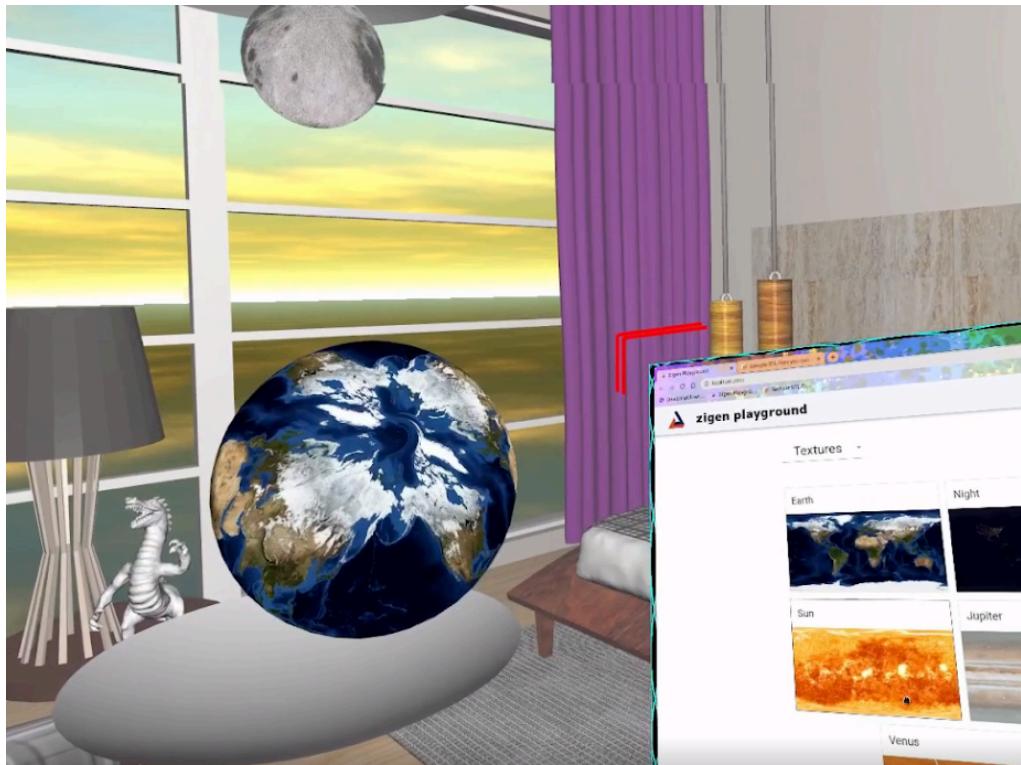


図 4.4. サンプルの 3D アプリケーションと Google Chrome

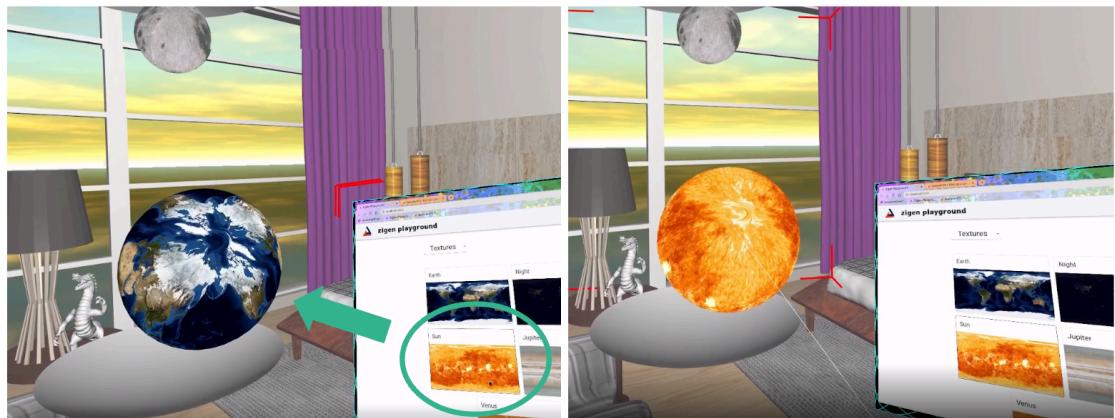


図 4.5. 3D アプリケーションと 2D アプリケーションとの間でドラッグ & ドロップする様子。左図がドラッグ & ドロップの前で、ブラウザから太陽のテクスチャをドラッグ & ドロップすることで、右図のように地球のテクスチャを太陽に変化させることができた。

検討されていなかった。本システムでは 3D アプリケーションの操作に関するプロトコルを 2D アプリケーションの操作に関するプロトコルと変換可能な形に、丁寧に設計することで、2D アプリケーションと 3D アプリケーションとの間のドラッグ & ドロップのような協調が可能であることを示した。

本システムでは Ray を用いて 2D・3D 両方のアプリケーションを操作するが、どのようなハードウェアでこの Ray を操作するかまでは規定しなかった。これは様々なハードウェアがシステムと連結できるようにするために、2D アプリケーションと 3D アプリケーションの両方に対して操作しやすいデバイスに関しては研究の余地があり、今後の研究課題としたい。

第5章

結言

最後に本研究がマルチコンテキストな没入環境の実現にどのように貢献したかを考察し、まとめとする。

5.1 レンダリングプロトコル

本システムで提案したレンダリングプロトコルは実験で示したとおり、実用に十分な性能を発揮した。レンダリングの自由度に関しては、アプリケーション間でレンダリングに影響を及ぼし合うことやマルチパスを用いたレンダリングは実現できなかったが、リッチな見た目を要求しない、仕事や生活をアシストするシンプルなアプリケーションを作るには十分であり、複数アプリケーションによるマルチコンテキストな没入環境を実現する第一歩としては十分に現実的なレンダリングの自由度を得たと考える。ただし、環境の落ち着きや、安心感を表現したりすることは重要であり、よりリッチなアプリケーションを実現可能にするためにも、レンダリングプロトコルは今後も改善の余地を研究する必要がある。

5.2 2D アプリケーションの利用

既存手法における2Dアプリケーションの利用では、画面共有の仕組みの上に作られていたことや、2Dアプリケーションと3Dアプリケーションとが同時に存在する場合を考慮していなかったために、通常のディスプレイで2Dアプリケーションを利用する場合と比べて制約があった。マルチコンテキストな没入環境という新しいパラダイムを実現していくうえで難しい点は、そのパラダイム上で動く3Dアプリケーションが最初に少なく、ユーザを惹きつけないということである。2Dアプリケーションをうまく使えるようにすることは、特にこの最初期において重要であり、本研究成果はマルチコンテキストな没入環境の実現に大きく寄与したと考える。

5.3 今後の展望・展開

本研究ではマルチコンテキストな没入環境の実現に必要な Windowing System におけるレンダリングプロトコルと 2D アプリケーションの利用に関して、提案と評価をした。しかし、マルチコンテキストな没入環境のための Windowing System にはまだ多くの研究の余地がある。例えば、複数のユーザがコラボレーションを図る場合は、どのようにアプリケーションの状態や位置を共有するのか、どこまで Windowing System としてサポートするべきかといった課題がある。また複合現実を考えた場合は、複数のアプリケーションが同じ現実の物体に対して何らかのレンダリングを行おうとして競合が発生したり、複数のアプリケーションがそもそも現実のオブジェクトをどのように理解するべきか、といった課題がある。本研究がこれらの課題を洗い出し、解決していくうえでのベースラインとなることを期待する。

また、本研究が目指すマルチコンテキストな没入環境では、様々なベンダーのアプリケーションが同時に起動して没入環境を作り上げてゆくという性質、また、それぞれのアプリケーションは空間全体ではなく、特定の機能だけを提供すればよいという性質から、アプリケーション自体の多様性や、その開発元の多様性は増してゆくと考える。本研究が発展し、没入環境がより参画しやすい、開かれた市場となることを期待する。

発表文献

- (1) Akihiro Kiuchi, Taishi Eguchi, Jun Rekimoto, Manabu Tsukada, and Hiroshi Esaki.
Zigen: A windowing system enabling multitasking among 3d and 2d applications
in immersive environments. In ACM SIGGRAPH 2022 Posters, SIGGRAPH '22,
New York, NY, USA, 2022. Association for Computing Machinery.

参考文献

- [1] Anthony G Gallagher, E Matt Ritter, Howard Champion, Gerald Higgins, Marvin P Fried, Gerald Moses, C Daniel Smith, and Richard M Satava. Virtual reality simulation for the operating room: proficiency-based training as a paradigm shift in surgical skills trainings. *Ann Surg*, Vol. 241, No. 2, pp. 364–372, February 2005.
- [2] Kristoffer B. Borgen, Timothy D. Ropp, and William T. Weldon. Assessment of augmented reality technology's impact on speed of learning and task performance in aeronautical engineering technology education. *The International Journal of Aerospace Psychology*, Vol. 31, No. 3, pp. 219–229, 2021.
- [3] Yun-Chieh Fan and Chih-Yu Wen. A virtual reality soldier simulator with body area networks for team training. *Sensors*, Vol. 19, No. 3, 2019.
- [4] Hailin Li. Analysis of virtual reality technology applications in agriculture. In Daoliang Li, editor, *Computer And Computing Technologies In Agriculture, Volume I*, pp. 133–139, Boston, MA, 2008. Springer US.
- [5] IDC. Ar/vr spending in asia/pacific* to reach 14.8 billion, driven by remote meetings, training, and collaboration, says idc. <https://www.idc.com/getdoc.jsp?containerId=prAP49932422>, (accessed Dec 17, 2022).
- [6] Robert W. Scheifler and Jim Gettys. The x window system. *ACM Trans. Graph.*, Vol. 5, No. 2, p. 79–109, apr 1986.
- [7] Jeffrey Tsao and Charles J. Lumsden. CRYSTAL: Building Multicontext Virtual Environments. *Presence: Teleoperators and Virtual Environments*, Vol. 6, No. 1, pp. 57–72, 02 1997.
- [8] Forrest F. Reiling. Toward general purpose 3d user interfaces: Extending windowing systems to three dimensions, 2014.
- [9] Dieter Schmalstieg, Anton Fuhrmann, Gerd Hesina, Zsolt Szalavári, L. Miguel Encarnaçāo, Michael Gervautz, and Werner Purgathofer. The Studierstube Augmented Reality Project. *Presence: Teleoperators and Virtual Environments*, Vol. 11, No. 1, pp. 33–54, 02 2002.
- [10] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, Vol. 14, No. 4, pp. 23–32,

1994.

- [11] Antti Peuhkurinen and Tommi Mikkonen. Mixed reality application paradigm for multiple simultaneous 3d applications. In *Proceedings of the 16th International Conference on Mobile and Ubiquitous Multimedia*, MUM '17, p. 133–141, New York, NY, USA, 2017. Association for Computing Machinery.
- [12] Rémi Coulon, Elisabetta A. Matsumoto, Henry Segerman, and Steve J. Trettel. Ray-marching thurston geometries. *Experimental Mathematics*, Vol. 31, No. 4, pp. 1197–1277, 2022.
- [13] Bo Zhang and Kyoungsu Oh. Interactive indirect illumination using mipmap-based ray marching and local means replaced denoising. In *Proceedings of the 25th ACM Symposium on Virtual Reality Software and Technology*, VRST '19, New York, NY, USA, 2019. Association for Computing Machinery.
- [14] Kun Zhou, Zhong Ren, Stephen Lin, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Real-time smoke rendering using compensated ray marching. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, New York, NY, USA, 2008. Association for Computing Machinery.
- [15] Tomoyuki Nishita and Eihachiro Nakamae. A method for displaying metaballs by using bézier clipping. *Computer Graphics Forum*, Vol. 13, No. 3, pp. 271–280, 1994.
- [16] John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, Vol. 12, No. 10, pp. 527–545, 1996.
- [17] Francois De Sorbier, Vincent Nozick, and Hideo Saito. Multi-view rendering using gpu for 3-d displays. *GSTF International Journal on Computing*, 07 2010.
- [18] Balázs Hajagos, László Szécsi, and Balázs Csébfalvi. Fast silhouette and crease edge synthesis with geometry shaders. In *Proceedings of the 28th Spring Conference on Computer Graphics*, SCCG '12, p. 71–76, New York, NY, USA, 2012. Association for Computing Machinery.
- [19] James F. Blinn. Simulation of wrinkled surfaces. *SIGGRAPH Comput. Graph.*, Vol. 12, No. 3, p. 286–292, aug 1978.
- [20] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-preserving simplification. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, p. 115–122, New York, NY, USA, 1998. Association for Computing Machinery.
- [21] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

謝辞

本論文の執筆にあたり、多くの方から多くのご指導やご支援、ご協力をいただきました、心より感謝申し上げます。

指導教員である江崎浩教授には、研究活動において多面的なご指導をいただきました。また、自分の好きな研究を自由に続けてこられたのは江崎先生の懐の深さのおかげだと思っております。心より感謝申しあげます。

落合秀也准教授は私が学部の頃から、研究活動の勝手がわからない私が研究を進めていくうえでいつも気さくに相談に乗ってくださり、暖かくご指導くださいました。心より感謝申しあげます。

塙田学准教授には合宿などの発表の場でのご指導や、学会への提出論文に対するご指導に加え、私の進路についても親身に相談に乗っていただきました。心より感謝申しあげます。

研究内容に限らず様々なことを教えていただいた研究室の先輩方、切磋琢磨し、刺激を与えてくれた同期、後輩一同に心より感謝申し上げます。また岩井愛映子秘書、高橋富美秘書には研究生活の様々な面でバックアップをしていただきました。心より感謝申しあげます。

首藤一幸教授をはじめ、未踏 IT 人材発掘・育成事業の PM・関係者の方々には本論文的具体的な実装である XR 向け Windowing System の実現に関して貴重なご助言と人脈の紹介を頂きました。心より感謝申しあげます。加えて稻見昌彦教授には、国際学会に出席したときにも準備段階や現地で色々と面倒を見ていただきました。勝手がわからない私が国際学会を楽しめたのは稻見先生のおかげです。心より感謝申しあげます。

構想時点から一緒に XR 向け Windowing System の研究開発をしてきた、江口大志氏、加えて同じ構想を共有し賛同してくれた今の開発メンバーである、伴玲吾氏、わたすけ氏、劉弘毅氏に心より感謝申しあげます。

最後にこれまでの学生生活をあらゆる面から支えてくださった家族と恋人、友人、全ての皆様に深く感謝申しあげます。

