〔非公開〕

ＴＲ－Ｍ－００１８

動画像を用いた仮想環境の生成表示法

山　口　　徹　也
Tetsuya　YAMAGUCHI

１９９７．３．３１

ＡＴＲ知能映像通信研究所

動画像を用いた
仮想環境の生成表示法

A Study of Creation and Displaying Scenes in Virtual Environment by Using Live Movie Images

平成９年３月３１日

ATR 知能映像通信研究所 第１研究室
大阪大学工学部 学外実習生
山口 徹也 [Tetsuya Yamaguchi]

動画像を用いた仮想環境の生成表示法

山口 徹也

内容梗概

仮想環境を構築する際に、背景シーンを含めて全てを３次元モデル化し、３次元コンピュータグラフィックスを用いて生成すると、リアリティが損なわれるばかりでなく、膨大な計算処理も必要となる。リアリティの向上のための計算コストが増大すれば、実時間でのシーン生成が困難となり、観察者との対話的な関係を実現することが困難となる。従って、リアルタイムで、仮想環境内にリアリティの高いシーンを生成するために、仮想環境内の背景シーンを生成する際、実写の２次元動画像を積極的に利用することが有効である。しかし、実写の動画像をそのまま仮想環境の中にはめ込むだけでは、視点の移動に伴う運動視差が実現できず、リアリティの欠如が問題となる。そこで視点に追従して動画像の表示を制御する手法の検討が必要となる。本研究では、その手法の基本的機能である、動画像利用によるリアリティ向上と観察者の視点位置を考慮することによる運動視差の効果を実現するシステムを構築し、手法の有用性についての検討を行った。このシステムは、表示用ディスプレイに対して平行な視点移動と、表示用ディスプレイに対する前後の視点移動に対応可能である。

今回の検討で、観察者の視点の変化に追従して実写の２次元動画像を切り換えて表示する手法は、運動視差の効果を実現可能とし、リアルで効率的な仮想環境の構築に有用である見通しを得た。

キーワード

仮想環境、 複合現実、実写動画像、３次元コンピュータグラフィックス

目 次

1 研究の背景

1.1 3次元CGによるシーン生成手法の問題点

近年、急速に発展をとげてきているVR(Virtual Reality)であるが、VR研究の発展、多様化に従って、これまでに大きく取り上げられなかった問題も深刻なものになりつつある。

VRシステムを構築する際に、その仮想環境内のシーンをどう生成するかという点についての問題である。従来はCGによる3次元モデリング・レンダリングを行って、仮想環境内のシーンを生成するのが代表的な手法であった。この3次元モデル化によるシーン生成手法はモデルベーストレンダリング (Model Based Rendering,MBR) と呼ばれるものである。計算機上に3次元データという形式でモデリングされた現実世界にレンダリングを行い、可視化するこの手法は、システム開発構想時の仮想環境内のシーンに対するビジョン、イメージといったものを、ある程度、可視化・具現化することが可能であった。

3次元CGを用いた手法では、モデリングさえできれば、現実には存在しないシーンを可視化することが可能となる。また、仮想環境内で、観察者にインタラクティブに対応して、形状を変える必要があるようなオブジェクトはモデリングデータを必要に応じて処理し、それをレンダリングすれば、容易に所望の形状が得られる。したがって、CGは種々のシミュレーションや仮想環境のシーンを構築する際に利用され、映画の特殊撮影にも頻繁に登場する。

しかし、仮想環境のシーンを構築する際に、全シーンをこのMBRに基づく3次元CGによる手法で生成しようとすると、シーンのリアリティが損なわれるばかりでなく、膨大な計算コストが必要となり、リアリティを向上させようとすると、シーン生成のための計算処理が増大し、システムのリアルタイム性が実現困難になる。

仮想環境内の全てのシーンを3次元モデル化して生成する場合、写真的なリアリティを得ようとすると、現実世界の対象を形状・3次元の位置関係などを忠実にモデリングし、視点から遠い物体をかすむように表示する処理などを行うことが必要となる。形状の確定している人工物に対しては比較的容易に、正確なモデリングを行なえるが、無機物や生物などの自然物体は、確定的にモデリングすればするほど人工的に感じられ、実世界より鮮明に見えすぎてしまい、自然さを失うため、フォトリアリスティックな画像を生成するのは容易ではない。近年では優れたモデリングソフトウェアが数々登場し、ある程度簡易にこの問題を解消することが可能となってきている。またフラクタル理論 [1] による自然物体のフォトリアリスティックな表現の研究なども盛んになってきている。しかし、リアリティ獲得のための計算コストが増大するため、写真的なリアリティをもつ画像をリアルタイムで生成可能とする手法は開発されていない。

つまり、視覚的リアリティ欠如の問題をある程度解決できても、生成された3次元モデリングデータを処理し、レンダリングを行なうには膨大な計算が必要となり、実時間で観察者とインタラクティブ (対話的) に対応することが困難になり、時間的リアリティの欠

如の問題が新たに発生する。リアルタイム性実現困難の問題は、近年、発展・進歩が顕著な画像レンダリング専用のGWSなどのハードウェア機器を導入すれば、ある程度、解消される問題ではある。また複数台のGWSに個々に処理を割り当てネットワークで接続を図ることも実用的であるし、実際に、規模の大きなシステムでは、ネットワークを介して接続された複数台のGWSでの負荷分散が当然となっている。しかし、システム自体の規模がさらに大きくなる場合、例えば、複数の人の位置・身体の動作・顔の表情などを検出してそれを反映したモデルを仮想環境内に実時間で再現する場合や、ネットワークを介し、複数の遠隔地で仮想環境を共有して共同作業をする場合などを想定すると、システムのリアルタイム性、つまり、システムと観察者との対話的関係を実現困難とする仮想環境内のシーン生成のための計算処理の増大の問題は深刻なものとなる。

1.2　2次元実写動画像利用の提案

1.1節で述べた、3次元ＣＧによるシーン生成手法のもつリアリティ欠如と計算処理増大によるシステムの実時間性実現困難の問題を解決し、少ない計算コストでをリアリティのある仮想環境を生成するために、実写の2次元画像をなるべくそのまま使って、3次元の空間的広がりを表現する手法について提案する。

2　視点に追従した動画像切り出し表示法

2.1　2次元実写画像の利用の際の要件

前章では、従来の3次元CGを用いたシーン生成手法の問題点について述べたが、問題解決のために、2次元実写画像を利用が有用であると提案した。

本章では、従来の実写画像を用いたシーン生成手法であるクロマキー合成とIBR(Image Based Rendering)の考察を行いながら、2次元実写画像利用の要件として、動画像の利用と視点位置・方向の変化を考慮することの意義・重要性について考察する。

2.1.1　動画像を利用することの意義

動画像というリアリティを持つもので仮想環境を覆うことにより、比較的簡易に実写2次元画像のリアリティを仮想環境内に反映させることができ、仮想環境内のリアリティの飛躍的な向上が期待されるが、3次元ＣＧを用いて、移動物体を生成するには、その姿勢や移動量を数値化しなくてはならない。これは、形状自体のモデリングと同様に、車両や飛行機などの人工物体の場合は比較的容易に行えるが、自然物体の場合には困難となる。例えば、流れ落ちる瀧の水流や、飛び回る鳥の群などの動きなどは、ＣＧによるシーン生成では、フォグ処理やパーティクルなどの処理を行わなければリアリティを十分得ることはできず、計算処理が増大して、実時間で、リアリティのあるシーンの生成は実現できない。従って、動画像の利用は、リアリティ向上の効果を最も効率的に得ることができる実写2次元画像の利用であると言える。

2.1.2　視点位置の変化に対応することの重要性

クロマキー合成では、動画像を利用可能なので、少ない処理でリアリティの向上が実現されるが、一方で、観察者の視点位置の変化が考慮されておらず、リアリティの欠如が起こる。われわれが日常生活のなかで現実世界を観察している場合、視点の位置を変化させると、視界に入る全シーンもそれに応じて変化する。これが運動視差の効果であるが、クロマキー合成の場合、この効果が得られず、リアリティの欠如につながる。

IBRの手法では視点位置・方向の変化を考慮し、それに応じたシーンの画像を表示することで、3次元の空間的広がりを表現可能である。ただし、画像の補間を行わない手法では、シーンの切り換えが離散的になり、補間を行う手法では、シーンの切り換えが、スムーズである、という違いがあるが、どちらの手法でも、前述のように動画像を利用することができず、現実世界を忠実に再現することはできない。

2.2　視点に追従した動画像切り出し表示法

2.1節のような検討を行った結果、動画像を利用して、運動視差の効果も得られる手法の検討が必要となる。

そこで、観察者の視点位置を考慮して動画像の表示を切り換える手法の一つとして、視点に追従した動画像切り出し表示法の提案をする。

この手法では、リアリティ向上のため、動画像を利用するが、２次元動画像をそのまま仮想環境の背景にはめ込むのではなく、検出された観察者の視点の位置情報に基づいて動画像の切り出し表示処理を行なう。

以下、この切り出し表示処理について述べる。

まず、撮影アングルの変化しない動画像 (図 2.1) を用意する。

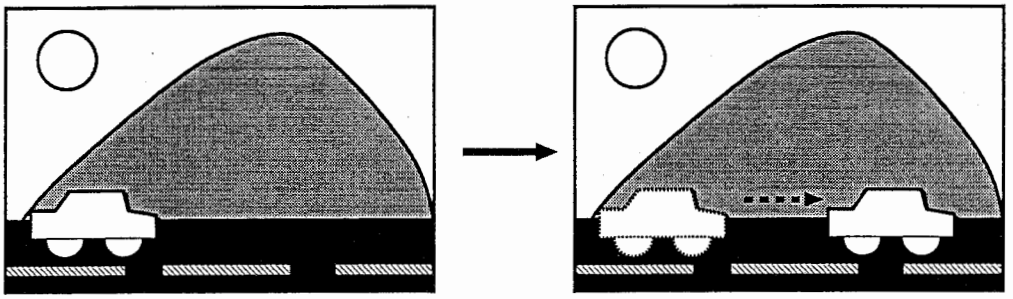

図 2.1: 撮影アングルが固定の動画像

撮影された動画像の１フレームを仮想環境全体とみなし、視点位置の変化に従って対応した範囲に window 処理を行い、切り出して表示するのである (図 2.2) が、より狭い領域を切り出して、それを表示すれば、ズームインの画像が得られ、逆により広い領域を切り出せば、ズームアウトの画像が得られる。この手法が優れている点は、リアリティの高い実写動画像を視点情報を元に処理し、視点の位置に対応した領域を少ない計算コストで表示することが可能であり、従って、リアリティ向上の効果と、運動視差の効果がリアルタイムで得られることである。従来手法との比較を下表にまとめる。

		3DCG	クロマキー合成	IBR	切り出し表示法
リアリティ	動画像利用	○	○	×	○
	運動視差実現	○	×	○	○
計算量		極めて大	小	中	小

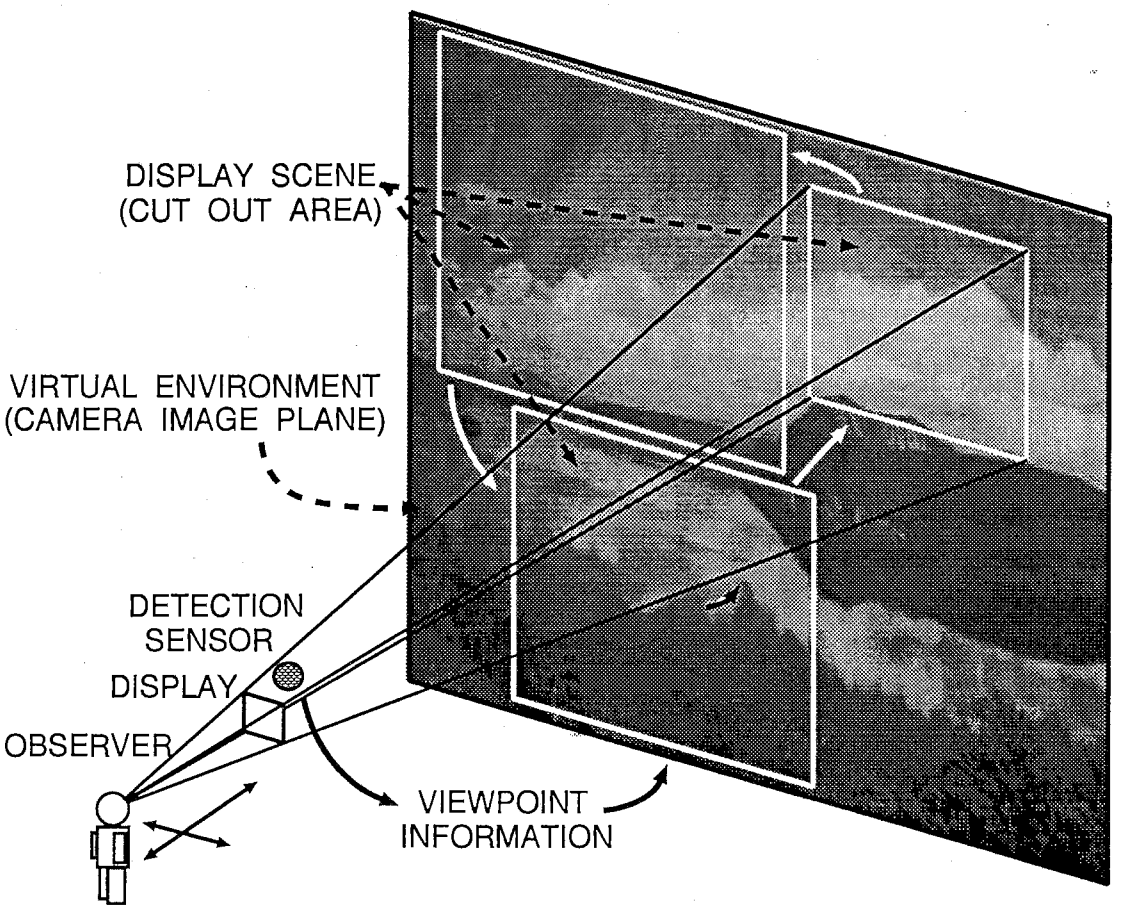

図 2.2: 切り出し表示処理

3　視点追従型仮想環境生成表示システム

提案した、視点に追従して実写動画像の表示を切り換える手法の効果、有用性を確認するために、その基本的機能を実現する、視点に追従した背景動画像切り出し表示法を用いたシステムを構築した。

システムの構成を図 3.1に示す。

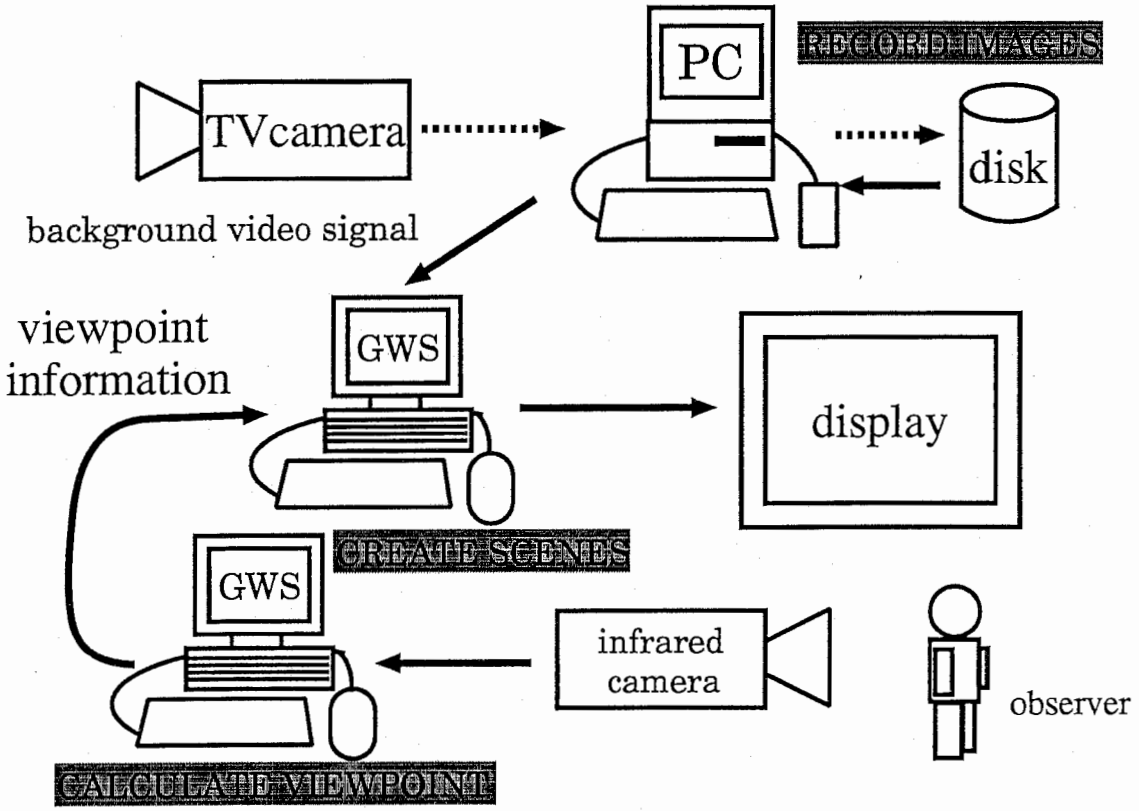

図 3.1: システムの構成

3.1　構築の構想

実写動画像の効率的な利用箇所として、モデリングが困難で、観察者と対話的な関係をもって大きく変化せずに、より写真的なリアリティを必要とする部分に利用するのが適当である。つまり、全シーンのなかの背景的役割を果たす箇所に組み込むことで、より効果的に利用できる。従って、図 3.2 のような仮想環境のシーンを構築することとした。これは、観察者と対話的関係をもって形状を変化させる必要のあるキャラクタモデルと、その背後に広がる背景シーンから構成されている。キャラクタモデルは３次元ＣＧによってモデリング・レンダリングを行い、背景シーンは動画像を組み込んで、観察者の視点に追従した切り出し処理を行って生成することとした。

表示を切り換える際に、背景シーンを切り換えるのと同様に、キャラクタモデルの移動、

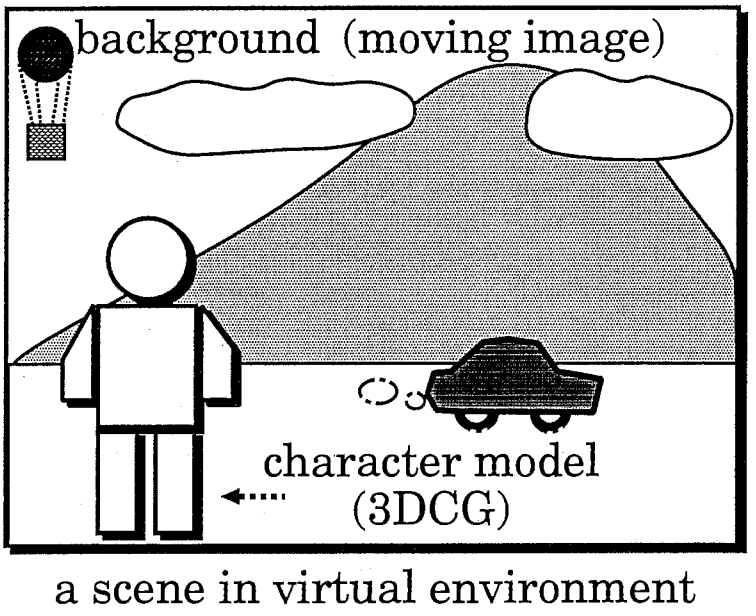

図 3.2: 背景シーンに実写動画像を利用

拡大縮小を行う必要があるが、背景シーンと同移動量、同倍率で処理すると、背景シーンの奥行きが表現されず、違和感が生じる。このため、自然で違和感のないキャラクタモデルと背景シーンの移動量比、倍率比を検討する必要がある。

視点位置の検出は、仮想環境の全シーンを表示するディスプレイに対して平行平面内の2次元の移動と、ディスプレイまでの距離に対応する必要がある。ディスプレイに平行な面上の視点の位置とディスプレイまでの距離から切り出す範囲を決定するためにである。

3.2 システムの概要

3.2.1 システムの構成

PC(Personal Computer) の機種は DEC 社の Alpha を用いた。OS は WindowsNT である。この汎用 PC に動画像用のキャプチャボード (DIGITAL 社、Perception) を実装し、動画像の蓄積に使用した。GWS(Graphics Workstation) の機種は Silicon Graphics 社の ONYX infinite reality を用いた。GWS 上での動画像取り込み、及び切り出し window 処理は C、C++を使用してプログラムで制御した。ライブラリは OpenGL、Silicon Graphics Video Library を使用した。

3.2.2 処理の流れ

まず、一台のテレビカメラで、撮影アングルを固定し、電車、車などの移動物体を含むシーンを撮影する。次に得られた動画像を PC のキャプチャボードを介して動画像蓄積用のハードディスクに記録しておき (図 3.1 点線)、記録した動画像を転送する。PC はハード

ディスクに保存されている動画像を呼出し、ビデオ信号としてをGWSのキャプチャボードに出力する。GWSはこのビデオ信号となった動画像をリアルタイムで、1フレーム毎にRGBのテクスチャイメージに変換する。一方、観察者の視点の位置(観察者の立ち位置)情報を赤外線カメラにより検出する。赤外線カメラの位置と仮想環境内のシーン表示用のディスプレイは同じ位置にあり、ディスプレイからの視点位置情報を検出する。PCから転送し、テクスチャイメージに変換した動画像に対して、検出した視点位置情報を元に、視点に追従した背景シーンの切り出し表示処理をGWS上で行なう。即ち、図3.3に示すようにGWS上に転送されてくる動画像の各フレーム(640x480画素のサイズ)から、検出された視点位置の変化に応じて様々な範囲を切り出すwindow処理を行う。最後にGWSで、3次元モデル化されたキャラクタモデル(本システムでは怪獣)をレンダリングし、切り出し処理を行った背景動画像と合成して表示する。

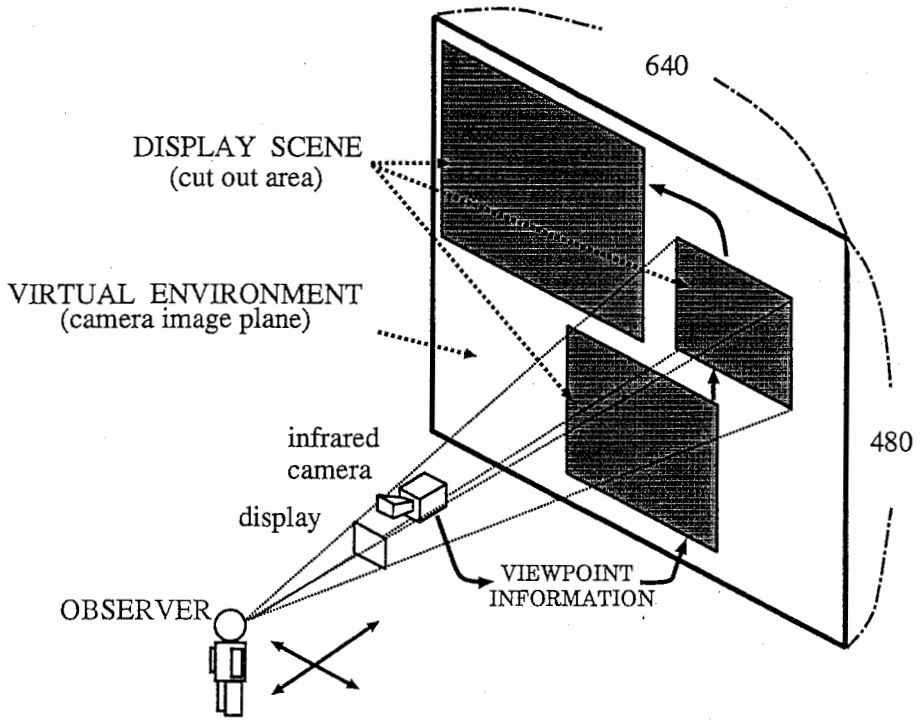

図 3.3: 処理の流れ

3.3 視点に追従した背景画像切り出し表示処理

運動視差の効果を得るための、観察者の視点位置に追従した背景動画像切り出し表示処理の実現について述べる。

3.3.1 視点位置検出

視点情報は赤外線カメラを使って検出される [2]。この手法は、赤外線カメラで人間の体温と室温の差を利用して、人物像を背景より抽出し、実時間で人物の全身像の動きを検出する [2]。検出のための処理はシーン合成処理とは別の計算機上で行なわれ、23 フレーム/秒の処理速度を実現している。

ただし、この検出は 1 台のカメラで実現されているため、観察者から表示用ディスプレイ (検出用カメラ)、つまり仮想環境までの距離が検出されていない。一台の検出カメラで観察者-カメラ間の距離という 3 次元情報を検出するために、動き検出の際に検出・算出されている人物の輪郭・身体の長さ等を利用した。人物の前後の移動は赤外線カメラの画像中の人物像の縮尺、長さや面積に影響を与える。つまり近づけば、身体全体は大きく映り、遠のけば、身体は小さく映る (図 3.4)。

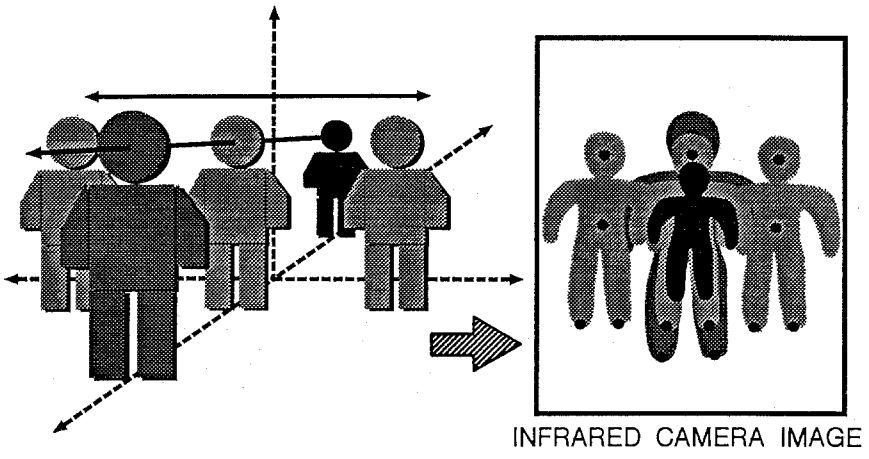

図 3.4: 観察者の移動と赤外線カメラの映像の関係

観察者までの距離を検出するには、具体的に次のような情報を利用する手法が考えられた。

- 頭部の面積の変化を利用。
- 頭部の長さの変化を利用。
- 身体の面積の変化を利用。
- 身体の長さ (身長等) の変化を利用。

リアルタイムで動いている人物の情報を検出しているため、面積にしても長さにしても頭部という身体の小さな部分の情報は、不安定になる。また、面積の情報はディスプレイに対する身体の角度に対しても急激に変化してしまう。そこで、安定性の高い身体の長さ情

報を利用して、検出カメラと観察者間の距離を算出した。つまり、頭部の高さがイニシャライズの際の値からどれだけ変化しているかを計算し、検出カメラと観察者間の距離を算出した。しかし、この手法では、前後移動を行なったときと、縦方向の平行移動、つまり、その場でしゃがんだり、立ち上がったりしたときの明確な区別ができず、観察者の視点位置を正しく検出することができない。そこで、重心から足先の長さ情報に留意し、観察者の体制を判定する。重心から足先の長さが直前の状態から大きく変化した場合には、観察者は前後移動を行わずに、その場でしゃがんだり、立ち上がったりしたと判定し、逆に、重心から足先までの距離が直前の状態から緩やかに変化した場合、観察者は前後移動したと判定する。前後移動か縦の平行移動かを区別する閾値は、実験を繰り返し、決定した。視点位置判定のアルゴリズムを図3.5に示す。ズーム処理は、直前が立っている状態で、かつ、今現在も立っている状態の場合に限り、実行する。

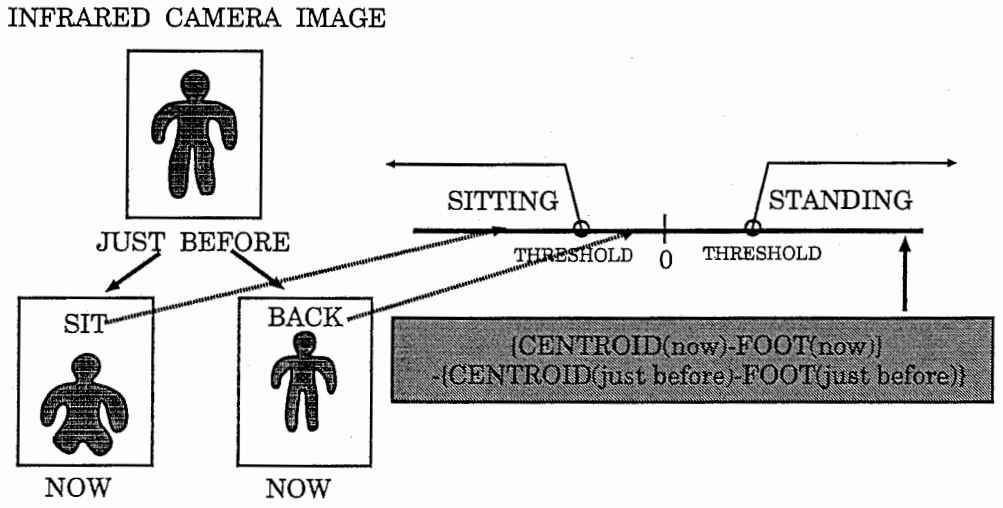

図 3.5: 視点位置判定のアルゴリズム

3.3.2 切り出し表示処理

赤外線カメラで検出した観察者の視点情報に基づいて、PCからGWSのキャプチャボードを介して得られた動画像のテクスチャイメージを切り換える処理については、図3.3で示されるように送られてくる動画像の1フレームを1枚の大きなバックグラウンド (仮想環境全体) と考え、視点位置に対応する範囲を切り出して表示することにより実現した。まず、視点位置はディスプレイに平行な平面内に拘束されるので、視点がディスプレイに平行で観察者を含む一平面のどこにあるのかという情報を用いて、切り出す領域の中心点を確定する。次に、観察者が表示用のディスプレイからどれだけ離れているかを3.3節の手法で検出・計算し、切り出す領域のサイズ (画素数) を拡大縮小して最終的な切り出し領域を確定する。この流れを図3.6に示す。切り出した領域のテクスチャイメージをマッ

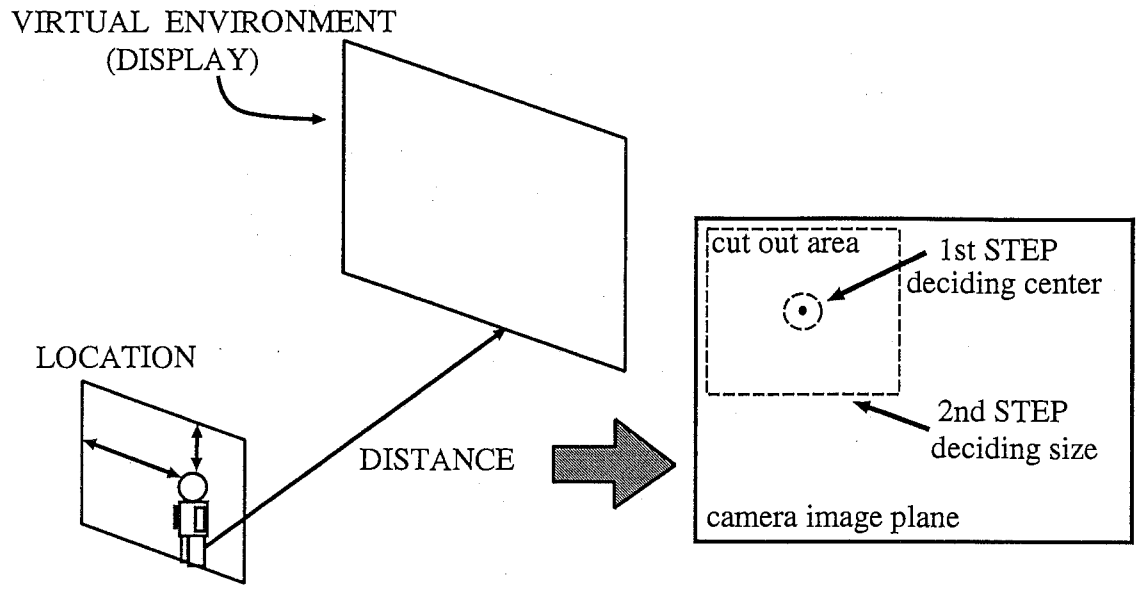

図 3.6: 表示領域確定のアルゴリズム

ピングする際に、ピクセルの補間処理を行う必要があり、補間するピクセルの中心に最も近い 4 つの周辺要素の平均を取って値を算出した。

3.3.3 背景シーンの奥行き感の実現

視点位置の変化に追従した表示画面の切り換え処理を行う際に、3 次元モデル化されたキャラクタと背景シーンを同倍率で拡大縮小処理すると、背景シーンとキャラクタの間の距離感が表現されず、キャラクタと背景シーンが同位置にあるように感じられ、リアリティが欠如してしまう。そこで、背景シーンとキャラクタの拡大縮小倍率を変える必要がある。

今、観察者、投影面、キャラクタ、背景シーンが図 3.7のような位置関係になっている。図中で、F は観察者から投影面までの距離、つまり焦点距離であり、D 1は観察者からキャラクタ、D 2は観察者から背景シーンまでの距離を表す。視点が前後に移動することにより、F、D1、D2 は変化する。

このような位置関係にあるとき、投影面に投影されるキャラクタは実際の D1/F 倍の長さに縮小され、背景シーンは D2/F 倍に縮小される [3]。従って、投影面上のキャラクタと背景シーンの、もとの長さからの拡大縮小倍率の比は D1：D2 となる。D1：D2 の倍率比で拡大縮小の処理を行うことで、手前にあるキャラクタの拡大縮小の倍率を背景シーンのそれよりも大きく変化させ、背景シーンの奥行き感を実現した。

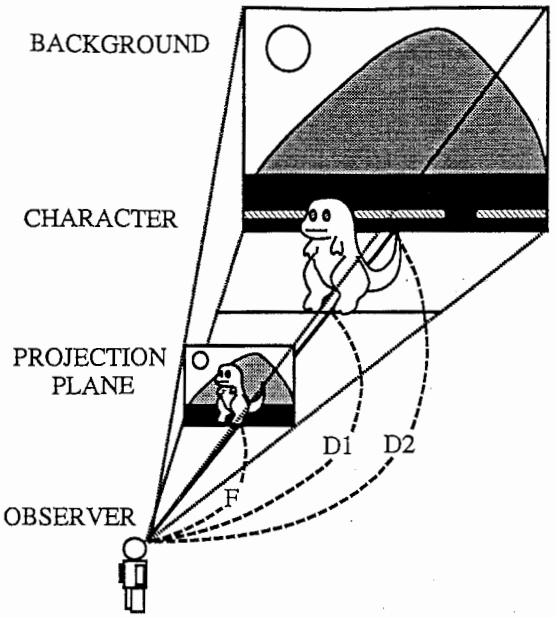

図 3.7: 背景シーンと 3 次元キャラクタの位置関係

3.4 実験結果

システム構築の結果、次に示すような視点位置の変化に対応した自然な画面の切り換えを実行可能とし、リアリティの向上と運動視差の効果を十分に得ることができた。

図 3.8は初期画面である。図 3.9は、キャラクタモデルが右方向に移動し、その動きに合わせて観察者の視点位置が右方向に移動した場合の表示シーンである。図 3.10は、キャラクタモデルの左方向の移動に合わせた視点の左方向への移動があった場合の表示シーンである。図 3.9、図 3.10における、観察者の視点位置の変化の様子を図 3.11に示す。つまり、仮想環境内のキャラクタモデルの移動に追従して観察者が視点位置を変える。次に視点位置の変化に対応して、背景シーンの切り換えが実行される。

観察者が表示用ディスプレイに近付いた場合の表示シーンを図 3.12に、遠ざかった場合の表示シーンを図 3.13に示す。

図 3.8: 生成された仮想環境 (1) 初期画面

図 3.9: 生成された仮想環境 (2) 視点位置がキャラクタモデルの右方向への移動に追従

図 3.10: 生成された仮想環境 (3) 視点位置がキャラクタモデルの左方向への移動に追従

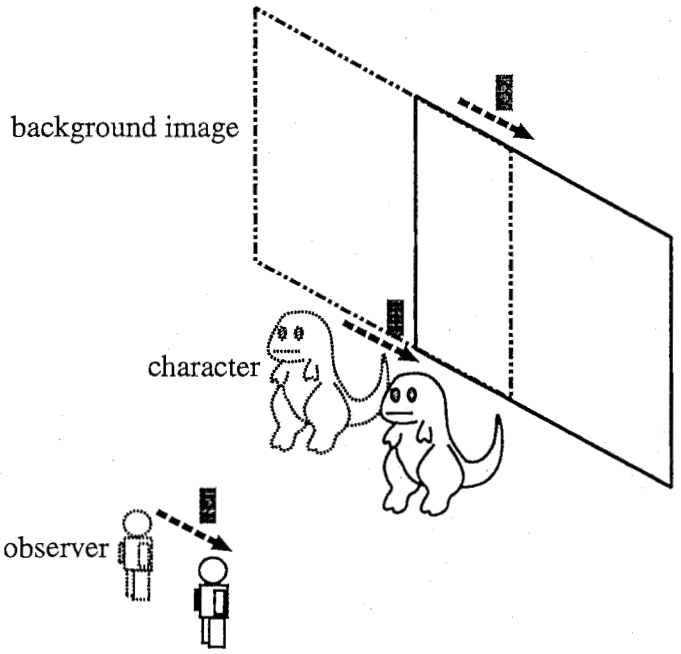

図 3.11: 視点位置の変化

図 3.12: 生成された仮想環境 (4) ズームイン

図 3.13: 生成された仮想環境 (5) ズームアウト

4　結論・今後の展開

4.1　結論

本稿では、3次元モデルによる仮想環境構築の際のリアリティの欠如、計算処理増大の問題を解決するために、実写の2次元画像、その中でも特に動画像を用いて、シーン生成を行う手法について検討してきた。構築したシステムの評価結果から、視点に追従した背景動画像切り出し表示処理のリアリティ向上、運同視差の効果獲得という基本機能を確認し、動画像を観察者の視点に追従して切り換え表示することは、よりリアリティのある仮想環境のシーン生成を効率的に、実時間で行うことが可能で、従来の仮想環境内の全シーンを3次元モデルで生成する手法がもつ、リアリティ欠如と計算処理増大の問題を解決するのに有効な手法であることが確認された。

4.2　今後の展開

本稿では、実写の2次元画像、その中でも特に動画像を利用して仮想環境を生成・表示する手法の検討を行ってきた。しかし、構築したシステムには、観察者の視点の自由度に制限があり、多少の画質の劣化も生じた。今後は、さらに本内容の研究・検討を進める予定である。現段階での構想について述べる。

4.2.1　視点・視線情報をもとに動画像の表示を切り換える

構築したシステムでは、視点に追従した動画像の切り換え表示の基本的効果を確認するために、リアルタイム性を重視した、計算処理の少ない切り出し処理による手法で実現し、画像を補間・再生成する処理を行わず、視点方向の自由度の制限が大きくなった。切り出し表示処理では、例えば、回転を含むようなアングル切り換えの際、画像を再生成することは困難である。視点方向、視線方向の自由度の制限は、切り出し処理のみでは生成不可能なアングルの画像を補間しなくては解決できない。実写の動画像から、物体の3次元構造を抽出するなどの、任意の視点からのシーンを生成するような処理をリアルタイムで実現するアルゴリズムを開発する必要がある。また、回転を含むアングルの画像を複数台のカメラを同期を取りながら撮影し、視点方向に対応したアングルの動画像から、視点位置に応じて切り出すwindow処理を行うことも視点自由度向上のための実現可能な手法の一つであると考えられる。

視点の自由度を向上させるためには、動画像表示の切り換え以外にも、観察者の視点位置・方向の検出に関しても検討する必要がある。構築したシステムでは、観察者の身体に何の装置も装着せずに、一台の検出用カメラで、ディスプレイに平行な面上の視点の位置とディスプレイまでの距離の検出を実現した。しかし、前述の、回転する視点変化を検出するには、複数台の検出装置が必要となる。また、観察者の頭部に磁気センサなどの検出装置を装着させ、視点位置を検出する手法も詳細な視点検出を可能にする手法の一つであ

る。さらに厳密に視点・視線の検出について考えると、臨場感通信会議 [4] で実現されている眼球の向きの検出 [5]、つまり黒目の向きの検出も考慮する必要が出てくる。

切り出し処理を行った際に、元の撮影された画像よりも、画質の劣化が生じた。システムの評価結果では、リアリティの欠如が大きく生じる程の劣化ではなかったが、大画面表示ディスプレイで、拡大表示を行わなくてはならない場合などには問題となるだろう。この画質劣化の問題は、高精細カメラや、複数台カメラの同期によって撮影した高精細、多画素、広視野の動画像を利用することで解決できる。

4.2.2 動画像からの情報抽出

本稿では、観察者の視点位置をもとに動画像の表示アングルを切り換えることを中心に検討をすすめてきたが、フレーム間の差分などから、動画像中から移動物体の動き情報や固定物体の 3 次元位置情報などを抽出し、その情報をもとに、表示アングルを切り換える試みも行いたい。

動き情報抽出の有効性に関しては、例えば、一般の撮影技法の中に、移動物体の動きと並進運動しながら撮影を行うドリーがあるが、観察者が仮想環境内の動画像を見ているとき、仮に飛び回る鳥の動きに並進したアングルで仮想環境を眺めてみたいと希望したならば、それに実時間でインタラクティブに対応して、鳥の動きに追従したアングルのシーンを生成・表示することが可能となる。このような処理を実現することによって、さらに視点の自由度の向上が期待でき、仮想環境のシーン構築に有効的に動画像を利用できる。

4.2.3 仮想自然環境での遠隔協調作業

以上のようなシステムの改良、種々の処理の実現を行い、動画像を用いたシーン生成手法を適用して、遠隔地の参加者が、ネットワークを介して仮想自然環境を共有し、その中での協調作業を可能とするシステムの構築を試みる。このシステムの確立によって、例えば、建築物を新設する際に、既存の建物との調和の確認や、新築する建物によって、周りの環境 (日当たりなど) がどのように変化するかといったシミュレーションを、遠隔地で協調的に行うことが可能となる。また、システムの付加価値として、建設地の現場の映像をリアルタイムでキャプチャリングすれば、人通りの多さなどのマーケティングを実施することも可能となる。

参考文献

[1] 桑原他:“フラクタルを用いた階層的な樹木形状表現による3次元樹木画像の高速生成方法,” 電子情報通信学会論文誌 D-II Vol.J78-D-II No.7 pp.1091-1104 (1995)

[2] 海老原他:“赤外線カメラを用いた実時間人物全身像動き検出,” 信学会ソサイエティ大会, p.212, '96.9

[3] 出口:“画像と空間 コンピュータビジョンの幾何学,” 昭晃堂

[4] 岸野:“ヒューマンコミュニケーション-臨場感通信-,” テレビジョン学会誌, 46,6, pp.698-702(1992)

[5] J.Ohya:“Virtual Space Teleconferencing:Real-Time Reproduction of 3D Human Images,” Jornal of Visual Communication and Image Representation, Vol.6, No.1, March, pp.1-25, 1995

付録 A

プログラム

A.1 概要

付録として、以下に、本研究で作成したプログラムを記載する。

このプログラムは第 1 研究室の [Virtual KABUKI system] に組み込まれて実行される。[Virtual KABUKI system] を起動し、背景メニューの中から"SIRIUS" を選択すると、背景シーンに実写ビデオのソースが合成される。観察者の前後移動に対応したズームイン、ズームアウトのシーン切り換えを行うためには、[Virtual KABUKI system] 起動の際、人物像検出プログラム"Mosura2" を起動する必要がある。

実写動画像を利用して背景シーンを生成する部分のソースプログラム、及び実行ファイルは、miris18:/home/tetsuya/DINO-DEMO/ に格納している。

また、"Mosura2" も同様に、miris18:/home/tetsuya/DINO-DEMO/ に格納している。

A.2 ソースプログラム

A.2.1 ビデオソースからテクスチャを生成する関数群 “vid2tex.c”

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <getopt.h>
#include <device.h>
#ifdef SIRIUS
#include <vl/vl.h>
#include <vl/dev_sirius.h>
#endif
#include <GL/glx.h>
#include <GL/glu.h>

GLfloat texture_matrix[4][4] = {
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 1
};

int  g_videomode=0;
int  g_cofrmnum=50;
int  g_cg2blend=0;
int  g_presno=0;
int  g_preno=1;
int  g_pretake=1;

extern struct SEND_DT g_socd;
extern int g_texcount;

#ifdef SIRIUS
/*
```

```
 * VL variables
 */
VLControlValue   g_size, g_timing;
VLControlValue   g_format;
VLControlValue   g_cap_type;
VLServer         g_svr;
VLPath           g_path;
VLNode           g_src;
VLNode           g_drn;

GLXVideoSourceSGIX g_glxVideoSource;
#endif

extern int        scriptS[20][20];
extern int        g_cur_bgdata[3];
extern int        g_stereo;
extern int        g_width, g_height;
extern GLfloat    g_alpha;
extern GLfloat    g_aplusminus;
extern GLfloat    g_cgalpha;
extern GLfloat    g_cgaplusminus;
extern Display   *fl_display;
extern Window     g_win;
extern GLXContext g_ctx;

extern struct INITDATA inidt;

/*
 * function prototypes
 */
void initVideo(int);
void initGfx();
void ready_video();
int  CreateTexture(void);
void DisplayNewFrame(void);
```

```
void cleanup(void);
void ProcessVideoEvents(void);
void UpdateTimingFormat(void);

void initVideo(int packingFormat)
{
#ifdef SIRIUS
    VLControlValue tex;

    /* open the server */
    if (!(g_svr = vlOpenVideo(""))) {
        fprintf(stderr, "couldn't open video\n");
        exit(1);
    }

    /* Get the Video source */
    g_src = vlGetNode(g_svr, VL_SRC, VL_VIDEO, VL_ANY);
    /* Get the Texture drain */
    g_drn = vlGetNode(g_svr, VL_DRN, VL_TEXTURE, 0);

    /* Create path   */
    g_path = vlCreatePath(g_svr, VL_ANY, g_src, g_drn);
    if (g_path < 0) {
        vlPerror("vlCreatePath");
        exit(1);
    }

    /* setup path */
    if (vlSetupPaths(g_svr, (VLPathList)&g_path,
1, VL_SHARE, VL_SHARE) < 0) {
        vlPerror("vlSetupPaths");
        exit(1);
    }
    /* select the appropriate events */
    if (vlSelectEvents(g_svr, g_path, VLStreamPreemptedMask |
```

```
                        VLControlChangedMask ) < 0) {
        vlPerror("Select Events");
        exit(1);
    }
    g_cap_type.intVal = VL_CAPTURE_NONINTERLEAVED;
/*
 g_cap_type.intVal = VL_CAPTURE_INTERLEAVED;*/
    if (vlSetControl(g_svr, g_path,
g_drn, VL_CAP_TYPE, &g_cap_type) <0) {
        vlPerror("VlSetControl");
        exit(1);
    }
    UpdateTimingFormat();
    /* Set the Texture packing mode */

    switch(packingFormat){
      case 0 :  tex.intVal = SIR_PACK_RGBA_6; break;
      case 1 :  tex.intVal = SIR_PACK_RGB_8; break;
      case 2 :  tex.intVal = SIR_PACK_RGBA_12; break;
      case 3 :  tex.intVal = SIR_PACK_YYYY_12; break;
      default : fprintf(stderr,"Invalid texture...\n"); exit(1);
    }

    if (vlSetControl(g_svr, g_path, g_drn, VL_PACKING, &tex) <0) {
        vlPerror("VlSetControl");
        exit(1);
    }
#endif
}

void initGfx()
{
#ifdef SIRIUS

g_glxVideoSource = glXCreateGLXVideoSourceSGIX(fl_display,
```

```
                        DefaultScreen(fl_display), g_svr,
                        g_path, VL_TEXTURE, g_drn);
if (g_glxVideoSource == None) {
fprintf(stderr, "can't create video source\n");
exit(1);
}

    glMatrixMode(GL_TEXTURE);
    glLoadMatrixf((GLfloat *) texture_matrix);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0, 0, -0.5);
#endif
}

void
ready_video()
{
#ifdef SIRIUS
    CreateTexture();
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glColor4f(1, 1, 1, 1);

    {
        const char *gl_extensions;
        int has_interlace;

        gl_extensions = glGetString(GL_EXTENSIONS);

if (!gl_extensions)
  {
    fprintf(stderr, "can't get GL_EXTENSIONS\n");
  }
```

```
        has_interlace = strstr((const char *) gl_extensions,
                               "GL_SGIX_interlace") != NULL;

        glEnable(GL_INTERLACE_SGIX);

if (!has_interlace
            || !glIsEnabled(GL_INTERLACE_SGIX))
  {
            fprintf(stderr, "can't enable interlace extension\n");
            fprintf(stderr,
    "This program will only run on InfiniteReality\n");
            exit(EXIT_FAILURE);
  }
      }
#endif
  }

int CreateTexture()
{
#ifdef SIRIUS
  GLubyte fakeTexture[1024][1024][3];
  int t_width, t_height;
  int vheight = g_size.xyVal.y/2;   /*check*/
  float  s_scale, t_scale, s_fraction, t_fraction;

    if (g_timing.intVal == VL_TIMING_525_SQ_PIX ||
        g_timing.intVal == VL_TIMING_525_CCIR601) {
        t_width = 1024;
        t_height = 512;
    }
    else {
        t_width = 1024;
        t_height = 1024;
    }
```

```
    s_scale = ( g_size.xyVal.x - 1) /(float) t_width;
    t_scale = vheight/(float) t_height;

    /* Sirius always transfers 768 pixels */
    s_fraction = g_size.xyVal.x/t_width;
    t_fraction = vheight/(float) t_height;

  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB5_EXT, t_width, t_height, 0,
       GL_RGB, GL_UNSIGNED_BYTE, fakeTexture);
  texture_matrix[0][0] =  s_scale;
  texture_matrix[1][1] = -t_scale;
  texture_matrix[3][1] =  t_scale;
#endif
}

/*
 * DisplayFrame: Display a new frame.
 * This routine is
 * called at 30 Hz for 525 timing and 25Hz for 625 timing.
 */
static int      count = 0;

void DisplayNewFrame()
{
#ifdef SIRIUS

  float  s_x, s_y;
  int i;
  s_x = 1.1;
  s_y = 1.1;

  glMatrixMode(GL_TEXTURE);
  glPushMatrix();
```

```
  glDisable(GL_LIGHTING);

  glClear(GL_COLOR_BUFFER_BIT);

  /*if (count++ & 1){
    glCopyTexSubImage2DEXT(GL_TEXTURE_2D, 0, 0, 0, 0, 0,
    g_size.xyVal.x, g_size.xyVal.y/2);
    } else
    glCopyTexSubImage2DEXT(GL_TEXTURE_2D, 0, 0, 1, 0, 0,
    g_size.xyVal.x, g_size.xyVal.y/2);
    */

  /* Load the texture from Sirius to Texture memory */

  glDisable(GL_INTERLACE_SGIX);

  for(i=0;i<2;i++)
    {
      if (count++ & 1)
glCopyTexSubImage2DEXT(GL_TEXTURE_2D, 0, 0, 0, 0, 0,
       g_size.xyVal.x, g_size.xyVal.y/2);
      else
glCopyTexSubImage2DEXT(GL_TEXTURE_2D, 0, 0, 1, 0, 0,
       g_size.xyVal.x, g_size.xyVal.y/2);
    }


  glMatrixMode(GL_TEXTURE);
  glLoadMatrixf((GLfloat*)texture_matrix);

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  glEnable(GL_TEXTURE_2D);
```

```
#endif
}

void cleanup()
{
#ifdef SIRIUS
  vlEndTransfer(g_svr, g_path);
  vlDestroyPath(g_svr, g_path);
  vlCloseVideo(g_svr);
  exit(0);
#endif
}

void ProcessVideoEvents()
{
#ifdef SIRIUS
  VLEvent ev;

  if (vlCheckEvent(g_svr, VLControlChangedMask|
   VLStreamPreemptedMask, &ev) == -1) {
    return;
  }
  switch(ev.reason) {
  case VLStreamPreempted:
    cleanup();
    break;
  case VLControlChanged:
    switch(ev.vlcontrolchanged.type) {
    case VL_TIMING:
    case VL_SIZE:
    case VL_PACKING:
      UpdateTimingFormat();
      printf("got VL_PACKING event\n");
      break;
    default:
```

```
        break;
      }
      break;
    default:
      break;
    }
#endif
}

void UpdateTimingFormat()
{
#ifdef SIRIUS
    /* Get the timing from input source */
    if (vlGetControl(g_svr, g_path, g_src, VL_TIMING, &g_timing) < 0) {
        vlPerror("vlGetControl");
        exit(1);
    }

    /* Set texture drain's timing to input source */
    if (vlSetControl(g_svr, g_path, g_drn, VL_TIMING, &g_timing) < 0) {
        vlPerror("vlSetControl");
        exit(1);
    }

    if (vlGetControl(g_svr, g_path, g_src, VL_SIZE, &g_size) < 0) {
        vlPerror("vlSetupPaths");
        exit(1);
    }
    if (vlGetControl(g_svr, g_path, g_drn, VL_SIZE, &g_size) < 0) {
        vlPerror("vlSetupPaths");
        exit(1);
    }
#endif
}
```

```
int startTrans()
{
#ifdef SIRIUS
vlBeginTransfer(g_svr, g_path, 0, NULL);
return 0;
#endif
}

int endTrans()
{
#ifdef SIRIUS
    vlEndTransfer(g_svr, g_path);
return 0;
#endif
}

int unset_blending()
{
#ifdef SIRIUS
if (!(g_videomode & 0x0808080)) {
glDisable(GL_BLEND);
return 0;
}
return 1;
#endif
}

int to_cgmode(int fade_flag)
{
#ifdef SIRIUS
if (g_videomode == 0) return;

if (!fade_flag) {
g_videomode = 0;
cgmode();
```

```
}
else {
if (g_videomode == 1) {
g_videomode = 0x0800080;
/*123
g_cgalpha = 0.0;
g_aplusminus = 1.0 / (float)g_cofrmnum;
*/
}
else {
g_videomode = 0x0800000;
/*123
g_alpha = 1.0;
g_aplusminus = -(1.0 / (float)g_cofrmnum);
*/
}
}
#endif
}

int cgmode()
{
endTrans();
glDisable(GL_INTERLACE_SGIX);
glDisable(GL_TEXTURE_2D);
/*glDeleteLists(1, 31);*/
/*read_texture(&inidt, 1);*/
/*123
use_simple_light_calc(g_lightnum);*/
}

int to_videomode(int fade_flag)
{
#ifdef SIRIUS
if (g_videomode == 1) return;
```

```
#if 1
glClearColor(0.0, 0.0, 0.0, 1.0);
glClearDepth(1.);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glDisable(GL_TEXTURE_2D);
#endif

if (!fade_flag) {
glDisable(GL_LIGHTING);
if (!g_videomode) {
ready_video();
startTrans();
}
g_videomode = 0x1;
}
else {
/*123
g_cgalpha = 1.0;
g_aplusminus = -(1.0/(float)g_cofrmnum);
*/
glDisable(GL_LIGHTING);
if (!g_videomode) {
ready_video();
startTrans();
}
g_videomode = (g_videomode)? 0x00081: 0x0800081;
   }
#endif
}
```

A.2.2 実写動画像と CG キャラクタの合成処理 "Demo.cc"

```
#include <GL/gl.h>
#include <GL/glx.h>
#include "Demo.h"

extern "C" {
  void initGfx(void);
  void ProcessVideoEvents(void);
  void DisplayNewFrame(void);
  void ready_video(void);
  void cleanup(void);
  int CreateTexture(void);
  int startTrans();
  int cgmode(void);
  int to_videomode(int);
  endTrans();
}

void
Demo::initialize(char *bserver, char *hserver, char *model_name)
{
  _win.open();
  _win.parent(this);

  backtexture.mapRead("../demo/back2.rgb");
  backtexture.set();
  backtexture.unset();
  cout << "../demo/back2.rgb read" << endl;

  body1.initialize(bserver, model_name);
#ifdef PARALLEL
  body2.initialize(bserver, model_name);
#endif
```

```
#ifndef TANTAI
  head1.initialize(hserver);
#ifdef PARALLEL
  head2.initialize(hserver);
#endif
#endif
}
//
void
Demo::action(int no)
{
  int i;
  extern void HumanDataClear(HumanData *);

  if (bodyMenu.calib) {
    calibHumanData();
    bodyMenu.calib = 0;
  }

  if (body1.avrg != bodyMenu.avrg) {
    body1.avrg = bodyMenu.avrg;
    body2.avrg = bodyMenu.avrg;
    free(body1.hdbuf);
    free(body2.hdbuf);
    body1.hdbuf = (HumanData *)malloc(sizeof(HumanData)*body1.avrg);
    body2.hdbuf = (HumanData *)malloc(sizeof(HumanData)*body2.avrg);
    for (i = 0; i < body1.avrg; i++) {
      HumanDataClear(&body1.hdbuf[i]);
    }
    for (i = 0; i < body2.avrg; i++) {
      HumanDataClear(&body2.hdbuf[i]);
    }
  }

#ifdef PARALELL
```

```
  if (!bodyMenu.finder) {
    memcpy(&body2.fd, &body1.fd, sizeof(HumanData));

  }
#endif

  switch (no) {
  case 1:
    body1.action();
#ifndef TANTAI
    head1.action();
#endif
    break;
  case 2:
    body2.action();
#ifndef TANTAI
    head2.action();
#endif
    break;
  }
}
//
void
Demo::draw(int no)
{
  gettimeofday(&tv1, &tz);
  drawWorld();

  glPushMatrix();
#if 1
  glScalef(sf*scale, sf*scale, sf*scale);
  movey = -30;
  glTranslatef(movex, movey, 0.0);
  glRotatef(rota, 0.0, 1.0, 0.0);
  glRotatef(elea, 1.0, 0.0, 0.0);
```

```
#endif

  glPushMatrix();
  switch (no) {
  case 1:
    body1.draw();
#ifndef TANTAI
    head1.draw();
#endif
    break;
  case 2:
    body2.draw();
#ifndef TANTAI
    head2.draw();
#endif
    break;
  }
  glPopMatrix();

  glPopMatrix();
  glXSwapBuffers(fl_display, _win.win());
  glFlush();
  gettimeofday(&tv2, &tz);

  t = (tv2.tv_sec*1000.0 + tv2.tv_usec/1000.0) -
    (tv1.tv_sec*1000.0 + tv1.tv_usec/1000.0);
  printf("========================\n");
  printf("time:%f(msec)\n", t);
  printf("%f(frame/sec)\n",1/t*1000);
}
//

int count = 0;

void
```

```
Demo::drawWorld(void)
{
  int       i, j;
  float       v[3];
  float       tv[3];
  //GLubyte scrn[3] =  { 0x000000, 0x2e5800, 0x445eef };
  static long scrn[3] =  { 0x000000, 0x2e5800, 0x445eef };
  float   px;
  float   py;
  float dframe1;
  float dframe2;
  float d1;
  float d2;
  float mat1[16], mat2[16];
  const float range = 60.0;
  const float range_s = 30.0;

#ifdef SIRIUS
  extern GLXVideoSourceSGIX g_glxVideoSource;

#endif
  glDisable(GL_LIGHTING);
  if (menu->back[0] < 0.0) {
    glClearDepth(1.);
    glClear(GL_DEPTH_BUFFER_BIT);
    glGetFloatv(GL_PROJECTION_MATRIX, mat1);
    glGetFloatv(GL_MODELVIEW_MATRIX, mat2);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.01, 0.99, 0.01, 0.99);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

   if (menu->back[0] == -1.0) {
      for ( i = 0; i < 5; i++ ) {
```

```
for ( j = 0; j < 3; j++ ) {
  // glColor4ubv(&(scrn[j]));
  glBegin(GL_POLYGON);
  v[0] = (1.0 / 15) * (i * 3 + j);    v[1] = 0.0;
  glVertex2fv(v);
  v[0] = (1.0 / 15) * (i * 3 + j +1);
  glVertex2fv(v);
  v[1] = 1.0;
  glVertex2fv(v);
  v[0] = (1.0 / 15) * (i * 3 + j);
  glVertex2fv(v);
  glEnd();
  count =0;
}
      }
    } else if (menu->back[0] == -2.0) {
#if 0
      cerr << "backtexture.index:" << backtexture.index() << endl;
#endif
      glColor3ub(255,  255,  255);
      backtexture.set();

      px = body1.G_dx;
      if (px < -range)  px = -range;
      if (px >  range)  px = range;
      glBegin(GL_POLYGON);
      tv[0] = 0.15 + (px / range) * 0.15;
      tv[1] = 0.0;
      v[0] = 0.0; v[1] = 0.0;
      glTexCoord2fv(tv);
      glVertex2fv(v);
      tv[0] = 0.85 + (px / range) * 0.15;
      tv[1] = 0.0;
      v[0] = 1.0; v[1] = 0.0;
      glTexCoord2fv(tv);
```

```
        glVertex2fv(v);
        tv[0] = 0.85 + (px / range) * 0.15;
        tv[1] = 1.0;
        v[0] = 1.0; v[1] = 1.0;
        glTexCoord2fv(tv);
        glVertex2fv(v);
        tv[0] = 0.15 + (px / range) * 0.15;
        tv[1] = 1.0;
        v[0] = 0.0; v[1] = 1.0;
        glTexCoord2fv(tv);
        glVertex2fv(v);
        glEnd();

        backtexture.unset();
        count = 0;
      }else if (menu->back[0] == -3.0) {
#ifdef SIRIUS
        if (count==0)
{
  count =1;
  glXMakeCurrentReadSGI(fl_display, _win.win(), g_glxVideoSource,
_win.context());
}
#endif
        glColor3ub(255,  255,  255);
        ready_video();
        startTrans();
        ProcessVideoEvents();
        DisplayNewFrame();

        ////set "px" for translating of background////
        px = body1.G_dx;
        if (px < -range)  px = -range;
        if (px >  range)  px = range;
```

```
      ////determine sit or stand////
      prephy = phy;
      prepcy = pcy;
      prepfy = pfy;
      phy = (float)body1.phy;              //head position
      pcy = body1.pcy;                     //centroid position
      pfy = (float)body1.pfy;              //foot position
      //dframe1 = phy-pcy-prephy+prepcy;  // from head to centroid
      dframe2 = pcy-pfy-prepcy+prepfy;     // from centroid to foot

      if(dframe2 >3.0){
fflag = STD;
      }else if(dframe2 <-3.0){
fflag = SIT;
      }else{
if (fflag == STD){
  py = phy - d_head;              //update "py" for zooming
        }
      }

      if (py < -range_s)  py = -range_s;
      if (py > range_s)  py = range_s;

      ////create background////
      glBegin(GL_POLYGON);
      tv[0] = 0.15 + (px / range) * 0.10+(py / range_s) * 0.10;
      tv[1] = 0.15+(py / range_s) * 0.10;
      v[0] = 0.0; v[1] = 0.0;
      glTexCoord2fv(tv);
      glVertex2fv(v);
      tv[0] = 0.85 + (px / range) * 0.10-(py / range_s) * 0.10;
      tv[1] = 0.15+(py / range_s) * 0.10;
      v[0] = 1.0; v[1] = 0.0;
      glTexCoord2fv(tv);
      glVertex2fv(v);
```

```
        tv[0] = 0.85 + (px / range) * 0.10-(py / range_s) * 0.10;
        tv[1] = 0.85-(py / range_s) * 0.10;
        v[0] = 1.0; v[1] = 1.0;
        glTexCoord2fv(tv);
        glVertex2fv(v);
        tv[0] = 0.15 + (px / range) * 0.10+(py / range_s) * 0.10;
        tv[1] = 0.85-(py / range_s) * 0.10;
        v[0] = 0.0; v[1] = 1.0;
        glTexCoord2fv(tv);
        glVertex2fv(v);
        glEnd();

        glMatrixMode(GL_TEXTURE);
        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glClearDepth(1.0);
        glClear(GL_DEPTH_BUFFER_BIT);
        glEnable(GL_LIGHTING);

        cgmode();
      }
      glClearDepth(1.);
      glClear(GL_DEPTH_BUFFER_BIT);
      glMatrixMode(GL_PROJECTION);
      glLoadMatrixf(mat1);
      glMatrixMode(GL_MODELVIEW);
      glLoadMatrixf(mat2);
    } else {
      glClearColor(menu->back[0], menu->back[1], menu->back[2], 1.0);
      glClear(GL_COLOR_BUFFER_BIT);
      glClearDepth(1.);
      glClear(GL_DEPTH_BUFFER_BIT);
    }

    ////set "scale" for zooming of character////
```

```
  d1 = CHARA_Z-py;
  d2 = BACK_Z-py;
  scale = (d2/d1)/(1-(2*py)/(7*range_s));
  lpy = py;                          //last_py is py
}

//
void
Demo::resetValue(void)
{
  sf = 1.0;
  scale = 1.0;
  movex = 0.0;
  movey = 0.0;
  rota = 0.0;
  elea = 0.0;
}
//
void
Demo::calibHumanData(void)
{
  HumanData   *p;
  CLIENT      *cl;
  HumanOption hp;

  hp.sync = 1;
  hp.calibration = 1;
  hp.sfplay = 0;

  if (!(cl = clnt_create(body1.mname,
HUMANDATAPROG, HUMANDATAVERS, "tcp")))
{
 clnt_pcreateerror(body1.mname);
      exit(1);
   }
```

```
  p = human_1(&hp, cl);
  if (!p) {
    clnt_perror(cl, body1.mname);
    exit(-1);
  }

  memcpy(&body1.fd, p, sizeof(HumanData));
  memcpy(&body2.fd, p, sizeof(HumanData));

  clnt_destroy(cl);

  printf("CENTROID:%f,%f\n", body1.fd.centroid[0], body1.fd.centroid[1]);
  printf("HEAD:%d,%d\n",     body1.fd.head[0], body1.fd.head[1]);
  printf("LEFTELBOW:%d,%d\n", body1.fd.leftelbow[0],
body1.fd.leftelbow[1]);
  printf("RIGHTELBOW:%d,%d\n", body1.fd.rightelbow[0],
body1.fd.rightelbow[1]);
  printf("LEFTHAND:%d,%d\n", body1.fd.lefthand[0], body1.fd.lefthand[1]);
  printf("RIGHTHAND:%d,%d\n", body1.fd.righthand[0],
body1.fd.righthand[1]);
  printf("LEFTFOOT:%d,%d\n", body1.fd.leftfoot[0], body1.fd.leftfoot[1]);
  printf("RIGHTFOOT:%d,%d\n",  body1.fd.rightfoot[0],
body1.fd.rightfoot[1]);

  body1.calibBones();
  body2.calibBones();
  d_head = body1.fd.head[1];
  fflag = STD;    //set fflag STD(stand)
}
```

A.2.3 その他

付録1に示した関数のうち、''initVideo''と、''initGfx''
をmain.ccで、''to_videomode''をdemoMenuX.ccで呼び出しておく必要がある。