3DS

3DS プログラミングマニュアル

動的立体視表現編

2015-01-15

Version 1.0

目次

コード

図

1. はじめに

このドキュメントでは動的立体視表現とは何か、またどのように実現するかを説明します。はじめに立体視表現での動的立体視表現構築のための理論的な枠組みを説明します。

次に、快適な立体表現を行うため、どのように理論的な制約から離れ、快適な表示を実現できるか、レンダリングにおける芸術性の高い制御ができるかを示します。

最後に、提供する API の詳細を説明します。

1.1. このドキュメントのスコープ

このドキュメントでは 3D シーンからの画像生成手法として透視投影のみを扱います。透視投影はグラフィックス開発者によく知られた手法であり、このドキュメントの読者は透視投影行列について既に理解していると想定しています。

1.2. 用語定義

本ドキュメントで以下の用語を使用します。

- 現実空間
  プレイヤーから上液晶までの距離などの現実世界の要素が CTR/SNAKE での立体視表示と関連があります。これらとアプリケーション内の空間を区別するため、プレイヤーや CTR/SNAKE 本体が存在する空間を指すのに「現実空間」という用語を使います。
- 仮想空間
  現実空間とは逆に、アプリケーションの中で作られる空間を仮想空間と呼びます。
- ビューポート
  ビューポートはレンダリングされた画像が表示される現実空間での領域です。この領域はスクリーン面の中にあるということを強調しておきます。多くの場合、ゲームでは上画面全体をビューポートとして扱います。
- ウィンドウ
  ウィンドウは仮想空間におけるビューポート表現です。カメラのビューボリュームのベース面/ウィンドウ面での切断面がウィンドウです。ユーザーがビューポートを見たときに、ウィンドウはまさに窓として、ユーザーに仮想世界を見せてくれます。ウィンドウ面に配置されたオブジェクトは、ユーザーから見て丁度スクリーンの距離に存在するように見えます。カメラから見て、ウィンドウ面より遠くのオブジェクトはスクリーンの後ろに、近くのオブジェクトはスクリーンより前に見えることになります。仮想空間でのスクリーン表現として仮想スクリーンも定義します。
- ベースカメラ
  ベースカメラはアプリケーションがシーンに応じて設定する単一のカメラです。ベースカメラの設定情報は、ユーザーの左右の目に相当する2つのカメラ位置を計算するのに必要です。
- 視点カメラ
  左目用画像や右目用画像の計算に用いるカメラを視点カメラと呼びます。

図 1-1. 仮想空間

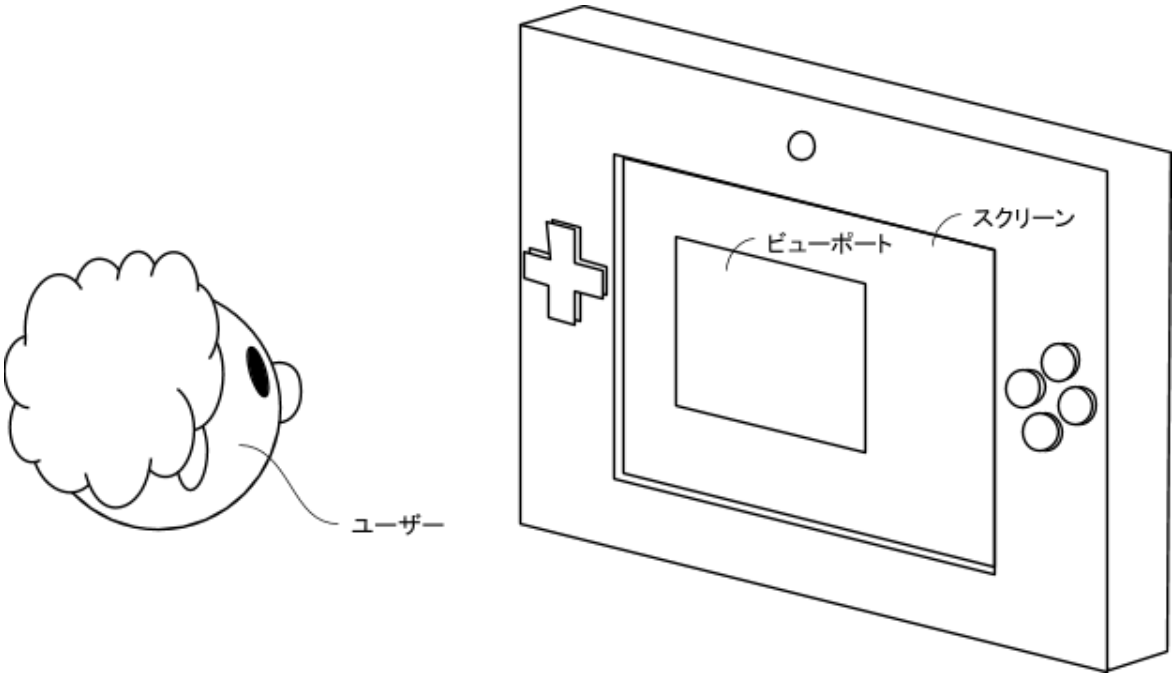
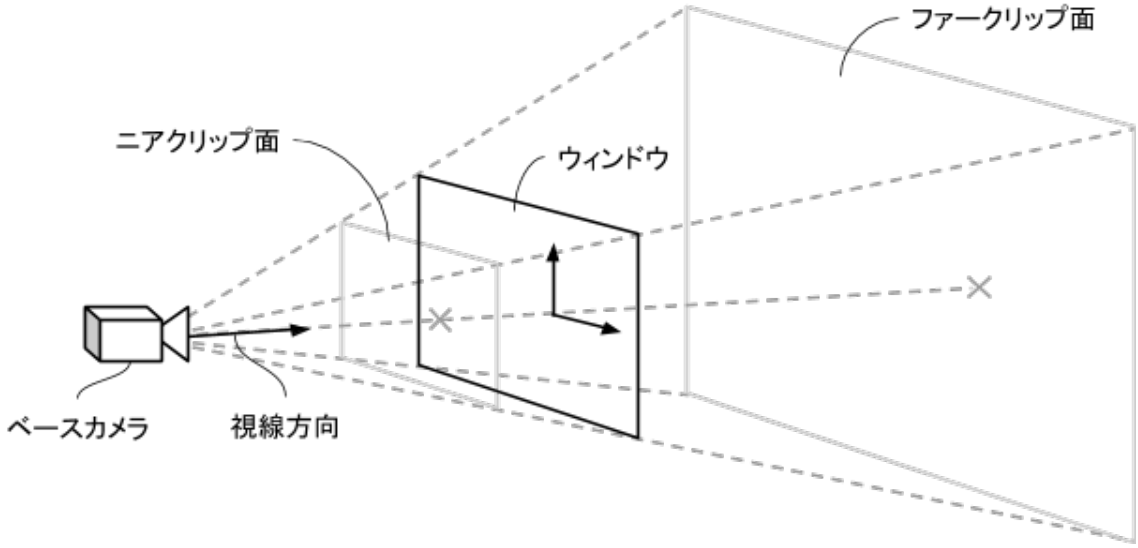


図 1-2. 現実空間



2. 理想的な画像のキャプチャーと表示

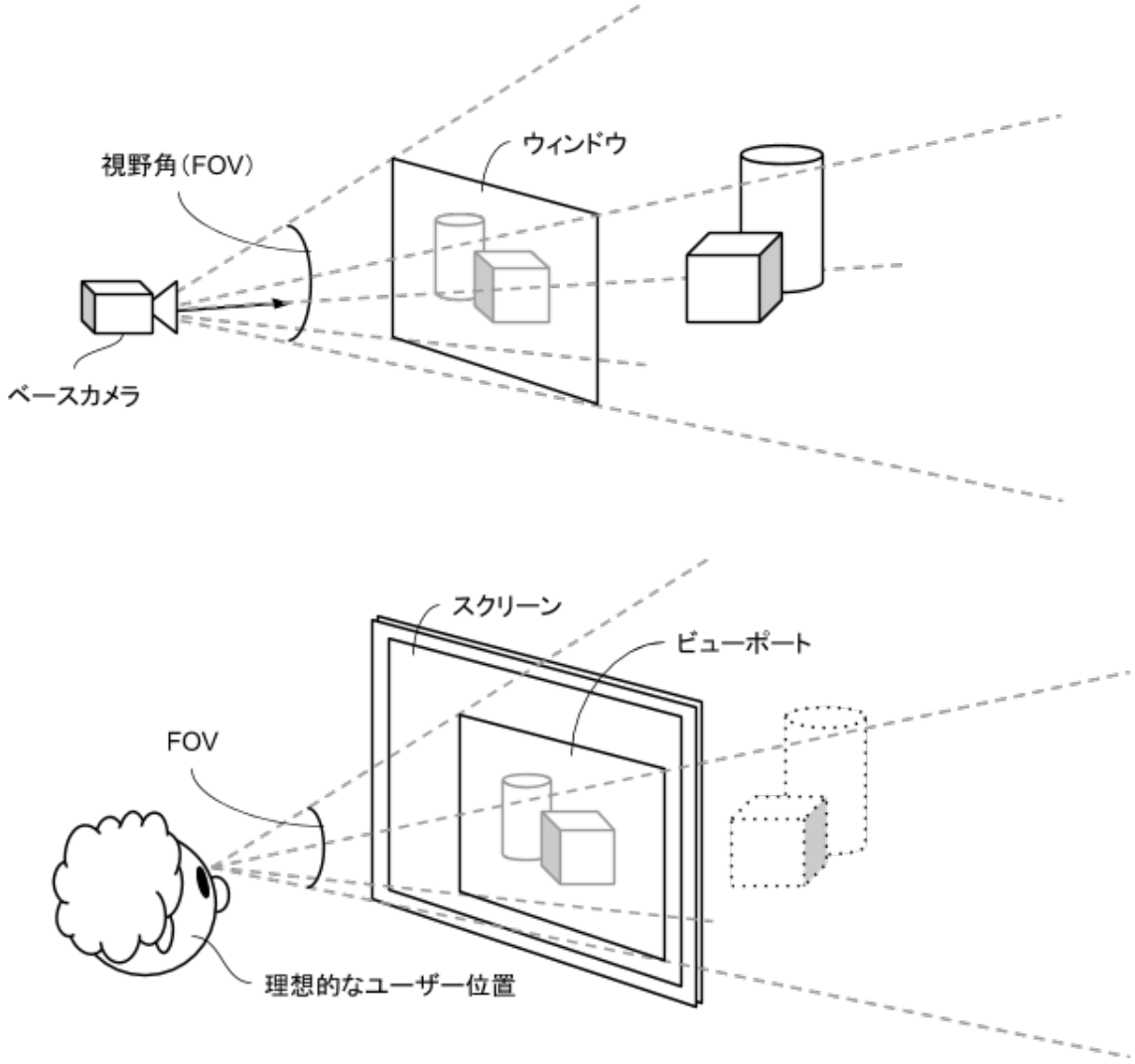
2.1. 理想的な単一カメラ表現

透視投影行列を使って 3D シーンから画像を生成する場合、3D 座標から 2D 座標への変換に透視投影（中心射影）を使います。（図 2-1 参照）

単一カメラ表現では、カメラ中心から任意の距離に基準面を配置できます。ビューポートのサイズに合わせてウィンドウのコンテンツの大きさも変わります。

ビューポートに画像が表示されるとき、ビューポートとユーザー視点の角度関係がウィンドウとカメラの角度関係と同じになるのが、理想的なユーザー位置です。理想的なディスプレイに関する自由度はスケールだけです。ディスプレイが大きくなると、スクリーンから視点までの距離も比例して大きくなります。

図 2-1. キャプチャー構成（上図）と理想的な表示構成（下図）



理想的なユーザー位置は一定の距離を置いたスクリーン正面だと考えられます。この場合、左右対称の視錐体のカメラを使い、視野角（FOV）を考慮すればよいわけです。しかし、ビューポートがスクリーン全域ではない場合は、左右非対称の視

錐体が必要かもしれません。

話をシンプルにするため、次の節ではビューポートがスクリーン全体と一致することを前提とします。

しかしながら、理想的な視点位置ではないからといって表示される画像が破たんするわけではありません。この効果を芸術的に活かすことも可能です。例えば、映画では観客に伝えたい印象を考慮して理想的ではない FOV を使うことがあります。ダイナミックに FOV を変えることで特別な効果を出すことも可能です。(例:ドリーズーム)

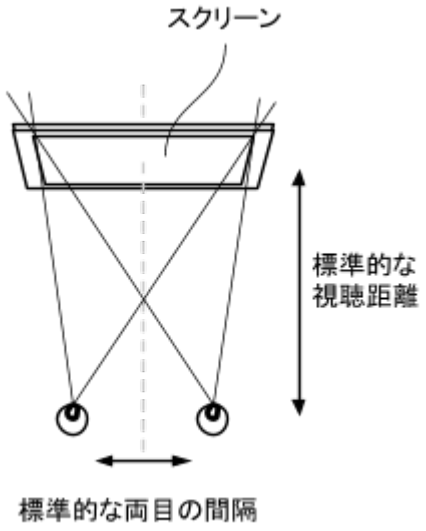
2.2. 理想的な立体視表現

1 つのスクリーンに 2 つの画像を表示することで立体視が実現されます。1 つはユーザーの左目が見る画像で、もう 1 つは右目用です。

ユーザーの標準的なプレイ姿勢における目の位置と、仮想空間における理想的な視点位置とを合致させることで、理想的な立体視表現を行うことが可能です。

通常、スクリーンの中央正面にプレイヤーがいるものとして視点位置が設定されます。そのときの設定は左右対称となります。つまり目は中心線を挟んで左右に等間隔に位置しているといえます。(図 2-2 参照)

図 2-2. 標準的な視点設定

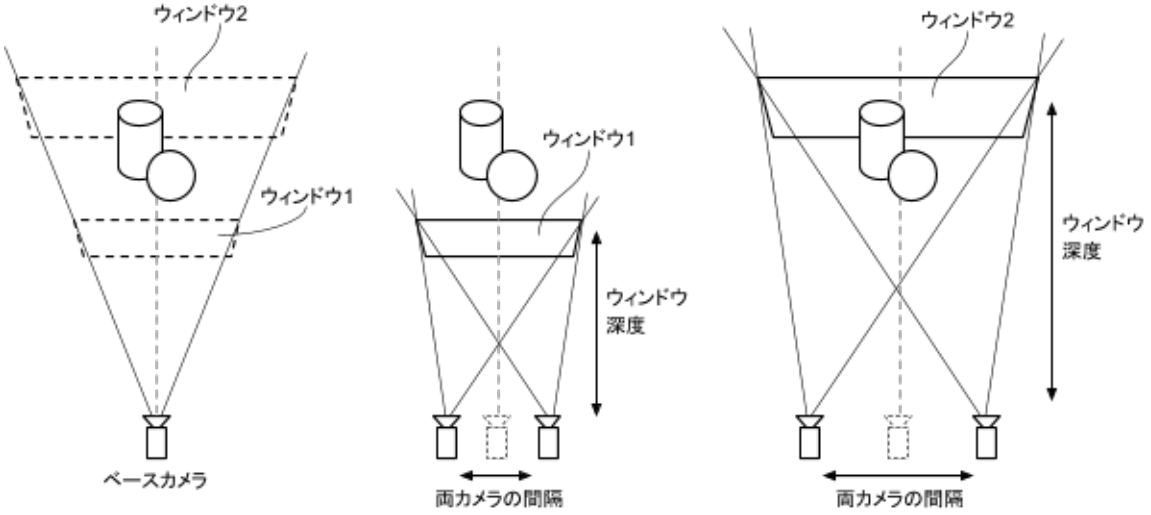


仮想空間における標準的な目の位置に合致する 2 つのカメラで立体視カメラは構成されます。

2 つのカメラを水平方向にずらして標準的な視聴状況でのユーザーの目の位置、FOV になるように配置することで理想的な立体視カメラを生成できます。

しかしながら、ここで 1 つ自由度が残ります。仮想のウィンドウはカメラから任意の距離に置くことができます(図 2-3 参照)。立体視表現にあわせて、このウィンドウ位置は開発者が定めなければなりません。仮想空間のどの面がスクリーン面として適切なのか、どのオブジェクトをスクリーンの前もしくは後ろに描画するのかを決めてください。ウィンドウの深度つまり位置を変えた場合は、描画されるオブジェクトの表示サイズも変わります。

図 2-3. ベースカメラ設定(左)、ウィンドウ 1 の場合の立体視カメラ(中央)、ウィンドウ 2 の場合の立体



ウィンドウはベースカメラから任意の距離に置くことができます。例としてウィンドウ 1、ウィンドウ 2 を考えます。ウィンドウ 1 の場合、オブジェクトはスクリーンの後ろにあります。一方ウィンドウ 2 の場合、オブジェクトはスクリーンの前にあります。

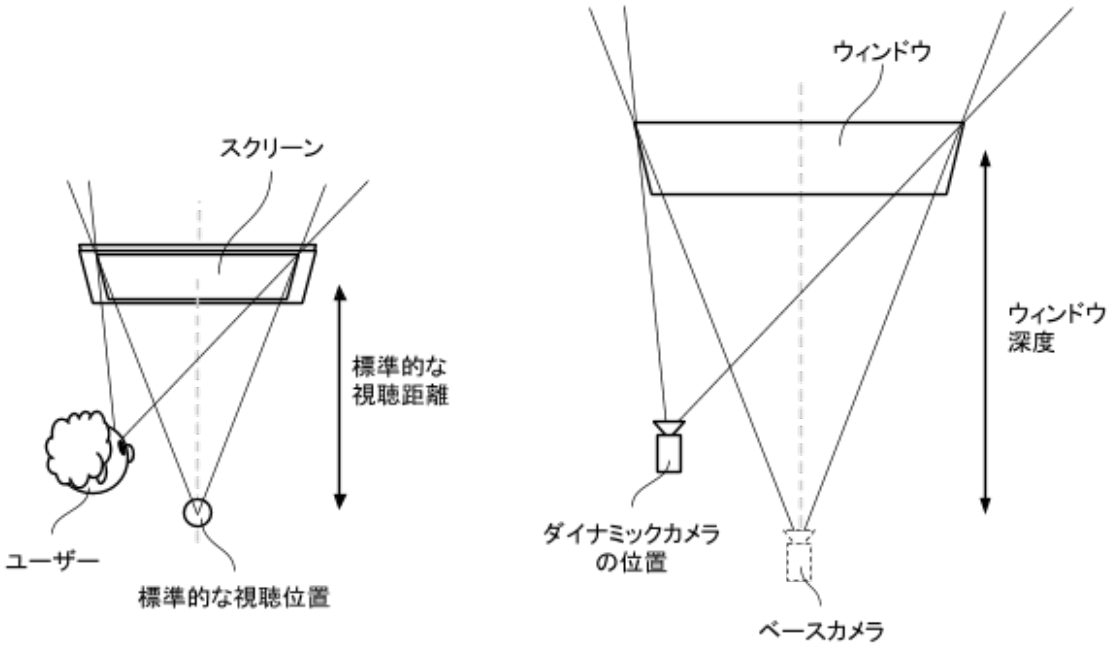
図 2-3 のように、両カメラのウィンドウは一致しなければなりません。結果として視点方向(ウィンドウ面の法線)は同じになり、ウィンドウ面を覆う視錐台は左右非対称になります。

画面サイズに沿った標準的な視認距離により、スクリーンに覆われる FOV が決まっています。理想的な立体視を実現する場合は、カメラはこの FOV を使うべきです。

2.3. 理想的なダイナミック単一カメラ

スクリーンに対するユーザーの相対位置がわかれば、現在のユーザー位置に合致する理想的な位置の画像を算出することが可能です。この手法をダイナミックパースペクティブと呼びます。

図 2-4. ダイナミックパースペクティブ



ユーザーの正確な位置がビューポート座標系で把握できるとします。カメラを配置するため、仮想空間のウィンドウ座標系を定義する必要があります。ウィンドウは現実空間における標準の視聴位置に相当するベースカメラから生成できます。

ウィンドウは視線方向に直交する任意の位置に設定できます。ウィンドウの位置が回転の中心になるため、カメラからウィンドウの距離によって、描画されるオブジェクトの見え方が変わります。理想的なダイナミックカメラを実現するには、ユーザーの移動に合わせて、図 2-4 のダイナミックカメラの位置のようにベースカメラを回転させる必要があります。これは立体視カメラでも同じことを意味します。

2.4. 理想的なダイナミック立体視カメラ

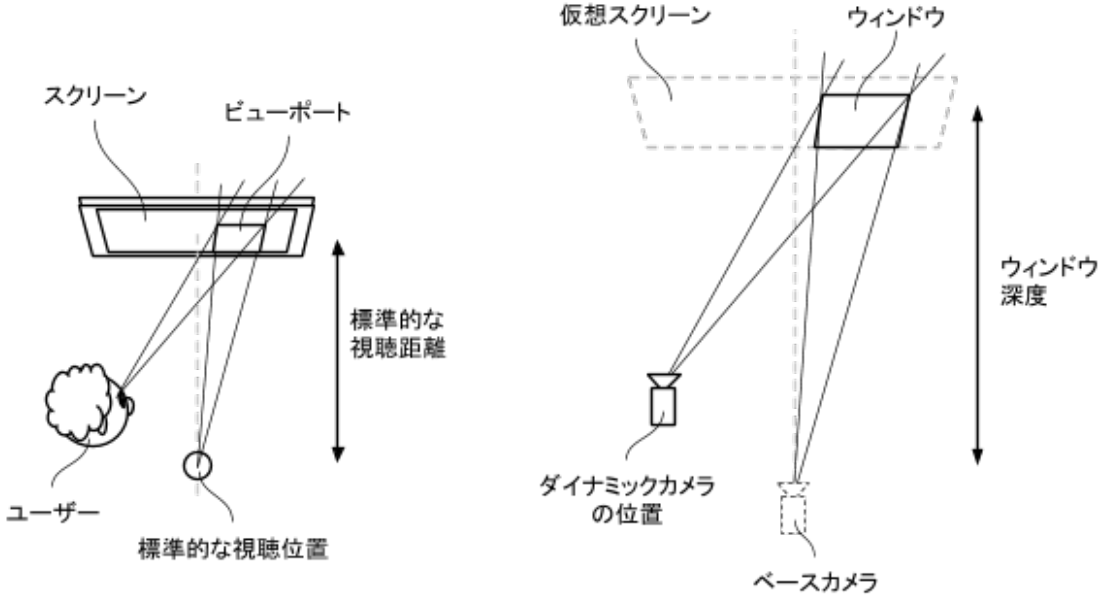
ダイナミックな立体視カメラを開発するためには、仮想空間でのウィンドウの位置を決め、先ほどのダイナミック単一カメラの場合と同様に、ユーザーの目の位置をカメラ位置として使う必要があります。

2.5. スクリーンを部分的にカバーするビューポートの利用

これまで画像がスクリーン全体に表示されると想定してきました。しかしながら、部分領域に表示する 3D 画像では特別な配慮が必要です。

システムが知ることのできるのはスクリーンに対するユーザーの相対位置だけです。開発者がスクリーンの部分領域としたビューポートに 3D 画像を表示しようとしたら、カメラ位置を正確に計算するためにビューポートとの相対的なユーザー位置が必要になります。開発者は理想的なダイナミック立体視カメラを生成するため、スクリーンの中のビューポートの正確な位置を規定しなければなりません。

図 2-5. ダイナミックカメラとビューポート



3. SNAKE への組み込み

3.1. フェイストラッキングシステムの制約

3.1.1. デプス推定

これまで、スクリーンに対するユーザー相対位置は完全に把握できると想定してきました。しかし残念ながら、SNAKE では完全な位置の把握はできません。フェイストラッキング機能により検知できるユーザー位置は、角度の面では良い精度で取得できますが、ユーザーとの距離については正確に計算することはできません。

そこでユーザーとの距離については、立体視効果を最大限享受できる最適視認距離にユーザーがいる、と想定することにします。

3.1.2. 平均的な両目の間隔値の使用

たとえシステムによって両目の位置を追跡できたとしても、個々のユーザーの実際の両目の間隔を知ることはできません。カメラ画像に写るユーザーの両目の間隔はスクリーンからの距離に応じて変わるため、そのまま両目の間隔として使うことはできません。そこで、平均的な両目の間隔を持つ仮想のユーザーを想定します。最適視認距離にユーザーがいるときに両目の間隔が平均値になるという想定で、カメラ画像に写る実際のユーザーも同じだと考えます。カメラ画像から得られる生の値ではなく、この仮想のユーザーの目の位置を全ての計算で使います。

3.1.3. ユーザー検出失敗時の対処

ユーザーがフェイストラッキングシステムの視認範囲外にでることがあります。悪条件のせいで認識がされないこともありえます。この場合、フェイストラッキングシステムは直前の位置情報とジャイロ入力を用いてユーザー位置について最善と思われる推測値を算出します。推測は直前に検出したユーザー位置から始めます。しばらくの間ジャイロ入力から SNAKE 本体に動きがないと分かった場合、推測値は時間とともに画面中央に移ります。検出できない期間が続いたときに、ユーザーは画面中央に戻ってくるだろうと想定しているからです。SNAKE 本体が動いている場合、ユーザーは静止し本体が動かされていると想定し、本体の動きからユーザー位置を推定します。

この挙動は少なくとも 3D ブレ防止機能のない立体液晶と同等のユーザー体験が得られるように設計されています。また、ユーザーが本体を動かした場合にも、良いユーザー体験を提供できます。

3.2. 快適性と最大視差

実験によって視差を保つことの重要性がわかっています。視差の限界距離という特別な値を考慮して、両目の左右画像におけるオブジェクトの距離を設計する必要があります。もしこの限界を超えていたら、多くのユーザーは不快に感じるでしょう。

先の章で説明した理想的な立体視カメラを利用し、オブジェクトがウィンドウ面からはるか遠くにある場合に、そのような状況が発生します。実際、無限遠に配置されたオブジェクトを両目の距離と同じだけ離して描画されると視差が大きくなってしまいます。この制約を踏まえて使える手法があります。

3.2.1. ニアクリップ面とファークリップ面

ウィンドウ面の近くにニアクリップ面とファークリップ面を設定すれば、目にするオブジェクトの最大視差を低減できます。

3.2.2. 両目間隔の縮小

他に両目間隔を短くしてシーンを描画し最大視差を低減させる方法があります。こうすることで遠くのオブジェクトの視差を低減できますが、近くのオブジェクトの立体視効果も低減されてしまいます。

3.2.3. FOV の拡大

以降のパートで説明しますが、ファークリップ面での視差は FOV を大きくすることで低減可能です。

4. ダイナミック立体視カメラの設定

先に定義した理論的なモデルを念頭におきつつ、芸術性や快適性や技術的な制約を解決する、モデルの制約を緩和したカメラ制御も可能です。

これから説明する機能を実装する場合に、ユーザーの目の位置を、ユーザーの絶対位置と目の相対位置とに分けて考えることとします。こうすることで、立体視効果とダイナミックパースペクティブ効果を独立して把握することが可能です。

立体視効果とダイナミックパースペクティブの分離のため、ユーザー位置としての目の中間位置と、そこからの相対位置としての目の位置とを使います。ユーザー位置はダイナミックパースペクティブ向けで、目の相対位置は立体視向けです。

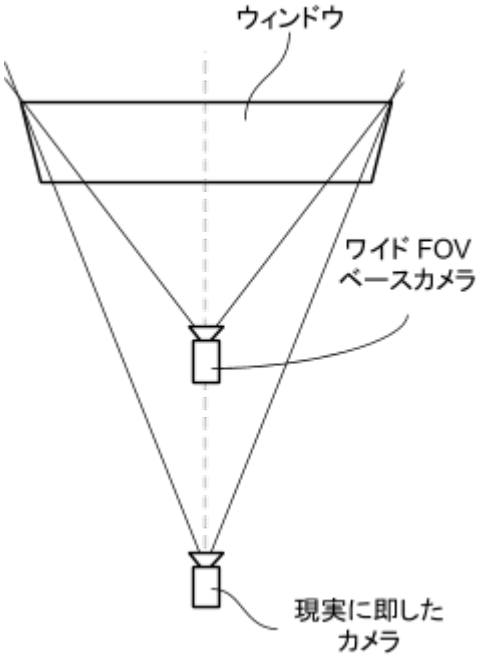
図 4-1. ユーザー位置としての目の中間位置と目の位置の分離

4.1. ワイド FOV

単一のカメラを扱う場合、ゲーム制作者が特定の FOV を使うのは自然なことです。一方で、理想的な立体視で必要な固定された FOV つまり標準的な FOV は、ユーザーが見るスクリーンの角度と一致しています。

たいていの場合、特定の FOV は標準的な FOV に比べて大きくなります。このような FOV を ワイド FOV と呼びます。

図 4-2. ワイド FOV カメラ

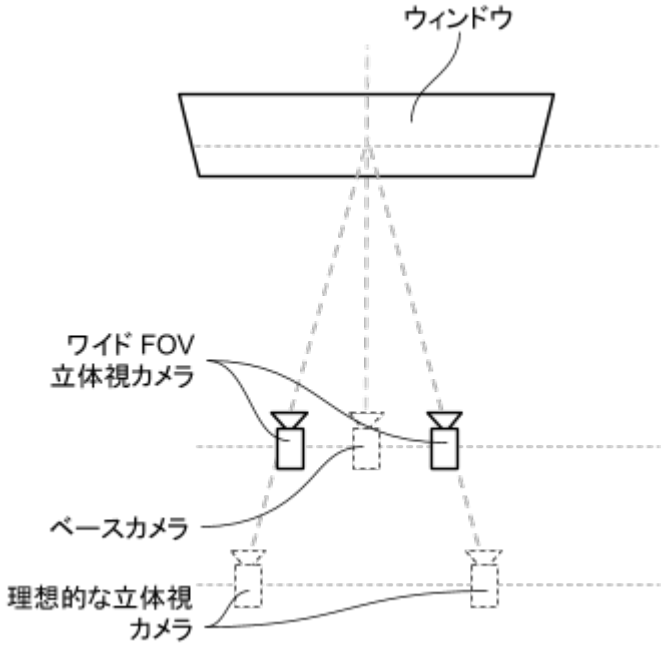


標準的な FOV に合致し、ウィンドウがベースカメラと合致する現実に即したカメラを使うことも可能です。しかし、芸術性の観点から異なる FOV が必要になるかもしれません。

もし ワイド FOV を使いたい場合、左右のカメラ距離をうまく設定することで、スクリーンの中心にあるオブジェクトのゆがみを低減することが可能です。左右カメラがウィンドウの中心点に向いている方向と、ユーザーがスクリーン中央を見ている方向とを合致させることで、このゆがみを最小限に抑えることができます。

図 4-3にあるように、理想的な立体視カメラのカメラ間距離と、ワイド FOV の左右のカメラ距離がウィンドウからの距離に比例するように配置することで、ゆがみを低減することができます。

図 4-3. ワイド FOV カメラの理想的なカメラ間距離



ダイナミックパースペクティブの場合、ユーザー位置を基にどのようにカメラ位置を計算すればよいのでしょうか。これには 2 つの手法が考えられます。

1 つ目は両目の距離を拡大縮小したように、動きを拡大縮小する方法です。この方法では、スクリーン中央のオブジェクトを適切な角度から見ることができます。しかし、遠くのオブジェクトの動きが小さくなります。

2 つ目はスケーリングせずに、カメラ画像から得られたユーザーの動きを再現することです。このモードでは遠方のオブジェクトは現実に即した動きをするように感じ、一方で前方のオブジェクトは回転したように、またはゆがんだように見えます。

どちらの手法も正しいと言えますが、描画シーンに応じて両者の中間的な振る舞いをさせるのが有効です。0 から 1 の値をとるスカラー係数 であるワイド FOV 動作係数を使って中間的な振る舞いを実現することが可能です。

図 4-4. FOV 動作係数 0 の場合

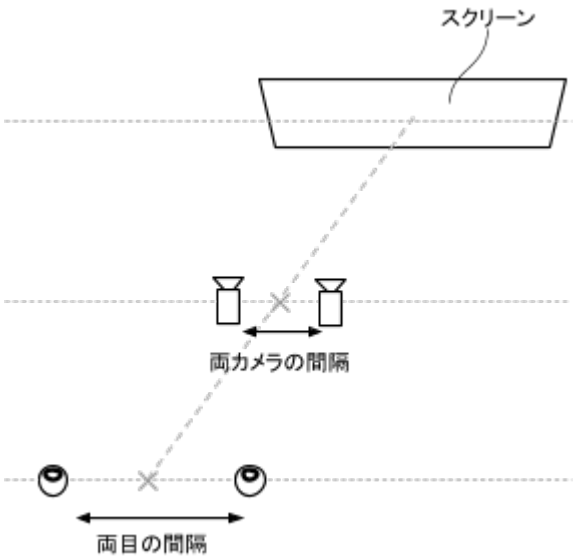
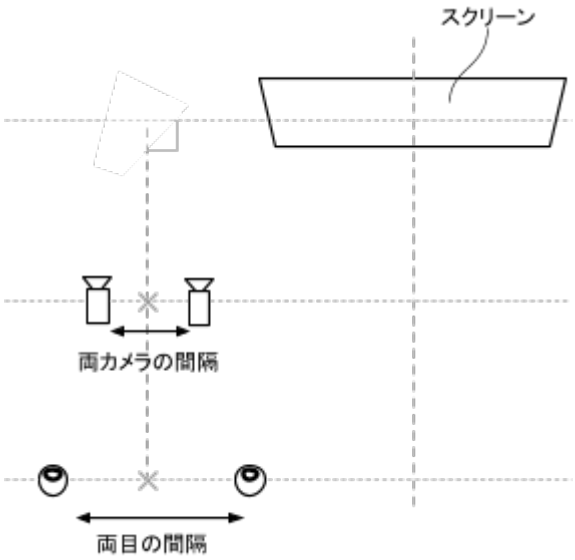


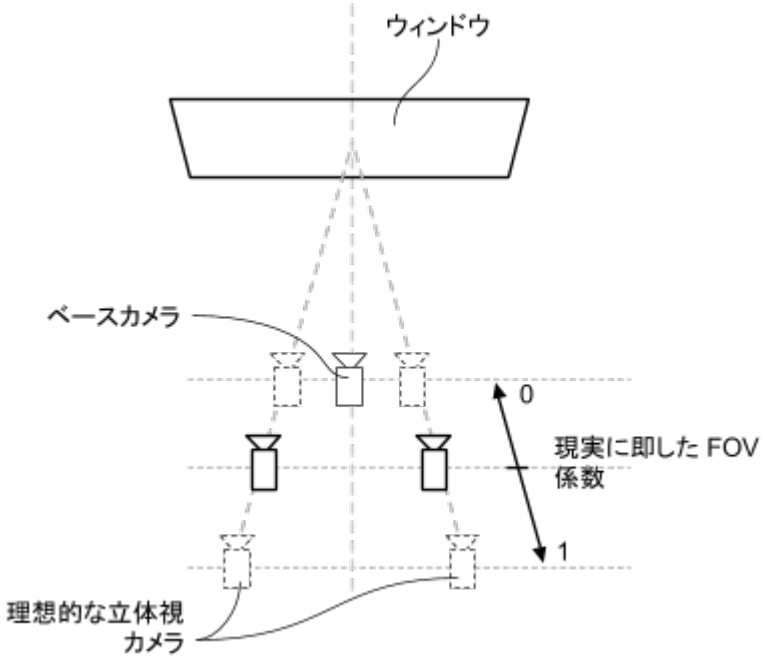
図 4-5. FOV 動作係数 1 の場合



4.2. 現実に即した FOV 係数

もし ワイド FOV 立体視カメラを理想的な立体視カメラにスムーズに変形したい場合は、両者を補間してみると興味深い動きをします。仮想カメラをある位置から別の位置に線形補間を使って動かすことになります。この線形補間の係数が現実に即した FOV 係数です。

図 4-6. 現実に即した FOV 係数



4.3. ダイナミックパースペクティブ振幅係数

ダイナミックパースペクティブ効果は係数をかけることで大きくも小さくもできます。この係数がダイナミックパースペクティブ振幅係数で、ユーザーの標準位置からの変位に相当します。

この振幅係数は意図する効果や固有の制約によって、x と y とを独立して別々の値を適用することも可能です。

4.4. ダイナミックパースペクティブクランプ係数

ユーザーの顔が検出される領域はカメラの FOV により決まっています。ユーザーがこの領域の外に動けば、システムはユーザー位置がわからなくなります。これ以上動けば領域外に出るということをユーザーが見てわかるフィードバックとして返すため、最大視認角度を狭くクランプすることが可能です。

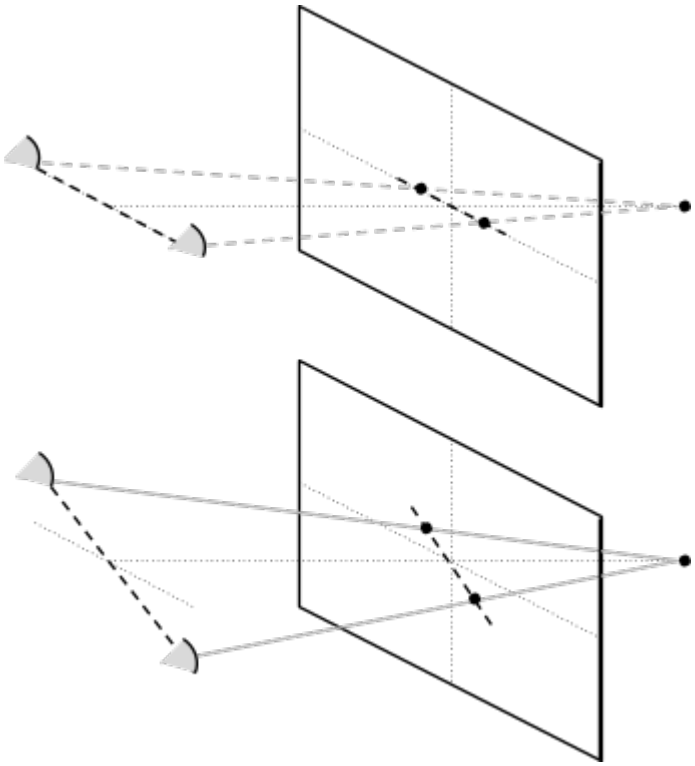
このクランプはカメラの視界となるボリューム制御にも使えます。

4.5. 傾きと立体カメラの改善

クランプ係数もしくは振幅係数を 0 に設定した場合、カメラの変位はなくなり、カメラは静的な立体視カメラのように見えるでしょう。

しかしながら、検出された目の位置を基にユーザーと本体の角度がわかるので、仮想カメラにこの傾きを適用することも可能です。ユーザーが車のハンドルのようにスクリーンを傾けた場合に、立体視の品質はさらに良くなるでしょう。

図 4-7. 静的な立体視カメラの改善



非常にまれなケースですが、傾きを使おうとすると問題が起きる場合があります。仮想カメラの動きによっては不自然な画像が生成されることもありえます。この場合は傾きを無効にしてカメラを水平にして使うことになります。

4.6. 限界視差

ユーザーの快適性を維持するため、アプリケーション開発者は視差の限界距離を指定することができます。ファークリップ面にあるオブジェクトでもこの距離より視差がつくことがないように、カメラ間距離の最大値はこの制約で決まります。この制約はニアクリップ面のオブジェクトには強制されません。

4.7. 3D ボリュームと立体視係数

ユーザーが快適に 3D 効果をチューニングできるように、3D ボリュームによりカメラ間の距離を伸縮させる必要があります。開発者は 0 から 1 の値をとる立体視係数を提要することが可能です。この係数を 3D ボリュームの値と乗算することでカメラ間の距離を縮めることが可能です。必要に応じて立体視効果を低減させたり、2D から 3D へのスムーズな遷移が可能になります。

5. 組み込み時の注意

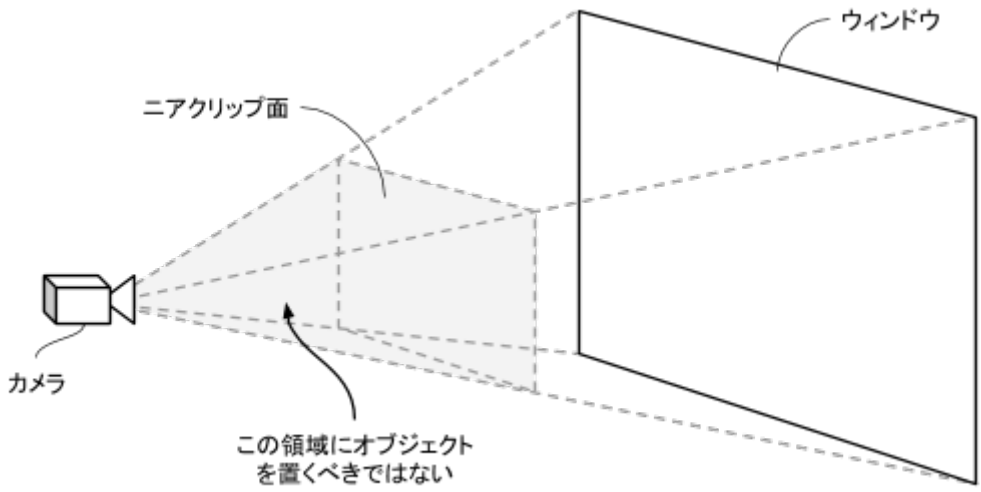
5.1. カメラの占有領域を決める

ゲームではカメラの配置場所に制限があり、配置可能な場所は決まっています。カメラはジオメトリオブジェクトと衝突しない位置に配置するのが望ましいです。動的立体視表現のカメラを使う場合、カメラはユーザーの動きを正確に再現するべきです。そのため静的な単一カメラであれば配置できるが最終的な両目のカメラは配置できない、というのは許容できません。プレイヤーの動きと一致しない挙動になってしまいます。

このような制約は、ベースカメラから両目のカメラを生成する前に考慮するべきです。衝突を避けるため、ジオメトリオブジェクトと衝突しない領域をカメラ用に確保する必要があります。

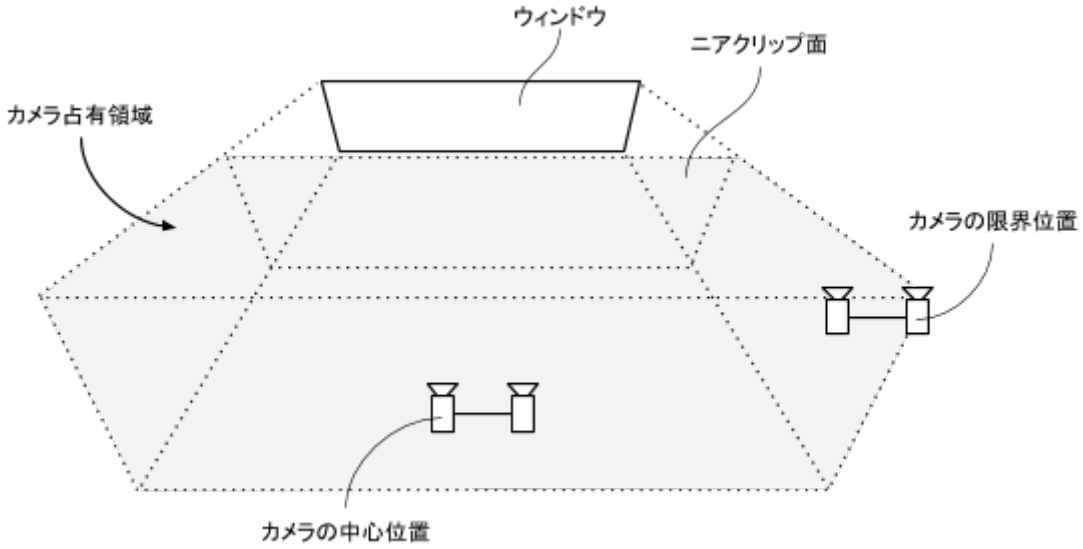
ユーザーが許容できるカメラ位置について説明します。カメラからニアクリップ面までのカメラ FOV の中にジオメトリオブジェクトがなにもない場合、カメラ位置は許容可能といえます。そうでなければレンダリングによる不自然な結果が見えてしまいます。

図 5-1. 静的な単一カメラのためのカメラ占有領域



動的立体視表現のカメラの場合、さまざまなユーザー位置に対して、対応するカメラ位置が許容可能か確認する必要があります。対応するカメラ位置の領域は、カメラ面からニアクリップ面までの視錐体として定義可能です。側面は、カメラが到達可能な限界位置からウィンドウ端までをつないだ面になります。図 5-2 を参照してください。

図 5-2. 動的立体視表現カメラのためのカメラ占有領域



6. DynamicStereoCamera クラス

6.1. ウィンドウの指定

固定のカメラを使う場合に指定する視錐体の代わりに、ダイナミックカメラを使う場合はウィンドウを指定しなければなりません。ニアクリップ面とファークリップ面も同様に指定が必要です。

6.1.1. 仮想空間でのサイズを基にウィンドウを指定

カメラ中心からウィンドウの幅と深さを決めることができます。ウィンドウはカメラの Z 軸が中心になり、高さは本体の上画面と同じアスペクト比となるよう幅から計算されます。

ウィンドウ幅の比とウィンドウの深さが FOV を定めていることに注意してください。

この方法は、画像がスクリーン全体に表示されることを想定しています。

コード 6-1. SetWindowFromSize() の定義

```
void SetWindowFromSize(const f32 windowWidth,
                       const f32 nearDepth,
                       const f32 windowDepth,
                       const f32 farDepth);
```

6.1.2. ベースカメラ視錐体を基にウィンドウを導出

ベースカメラ視錐体の左辺、右辺、底辺、上辺への角度(タンジェント値)を指定することが可能です。この視錐体とウィンドウ深度の面との共通集合をウィンドウと定義します。この方法は、スクリーン全体をカバーしない場合のビューポート表現にも使えます。ビューポート境界のタンジェント値は、スクリーン正面の標準的なユーザー位置から計算されるべきです。ビューポート表現に合致する、スクリーンの最大領域に画像表示することを想定しています。

コード 6-2. SetWindowFromTangent() の定義

```
void SetWindowFromTangent(const f32 tangentLeft,
                          const f32 tangentRight,
                          const f32 tangentBottom,
                          const f32 tangentTop,
                          const f32 nearDepth,
                          const f32 windowDepth,
                          const f32 farDepth);
```

6.1.3. ビューポートからウィンドウを設定

スクリーンを部分的にカバーするビューポートを扱う場合、スクリーンのどこがビューポートなのかを知る必要があります。スクリーンを部分的にカバーするビューポートや現実感のある FOV を使う場合には、以下のメソッドが有用です。

コード 6-3. SetWindowFromViewport() の定義

```
void SetWindowFromViewport(const f32 screenHorizontalHalfFOVTangent,
                           const f32 viewportLeftInUnitSceen,
                           const f32 viewportRightInUnitSceen,
                           const f32 viewportBottomInUnitSceen,
```

```
                                const f32 viewportTopInUnitSceen,
                                const f32 nearDepth,
                                const f32 windowDepth,
                                const f32 farDepth);
```

6.1.4. 既存のプロジェクション行列を基にウィンドウを設定

視錐体を計算するベースカメラのプロジェクション行列とウィンドウ深度を指定することが可能です。

左右対称の視錐体を前提とした上画面全体をターゲットとするプロジェクション行列の場合にしか、この方法は使えません。

コード 6-4. SetWindowFromProjectionMatrix() の定義

```
void SetWindowFromProjectionMatrix(const nn::math::Matrix44 *proj,
                                   const f32 windowDepth);
```

6.2. ベースカメラビューの指定

コード 6-5. SetView() の定義

```
void SetView(const nn::math::MTX34* viewMatrix);
void SetView(const nn::math::VEC3& position,
             const nn::math::VEC3& upDirection,
             const nn::math::VEC3& forwardDirection);
```

nn::math::MTX34LookAt のような関数で生成されるビュー行列を指定する、もしくはベースカメラ位置を直接指定してください。ビュー行例や nn::math::Vector3 引数を指定するときに、CTR のグラフィックシステムに合うように右手座標系を使うことに注意してください。

6.3. 左目カメラと右目カメラの計算

コード 6-6. ComputeViewsAndProjections() の定義

```
f32 ComputeViewsAndProjections(
    nn::math::Matrix44* projL, nn::math::Matrix34* viewL,
    nn::math::Matrix44* projR, nn::math::Matrix34* viewR,
    const nn::math::VEC2* leftEyeTangent,
    const nn::math::VEC2* rightEyeTangent,
    const Setting& setting,
    const nn::math::PivotDirection pivot = nn::math::PIVOT_UPSIDE_TO_TOP
) const;
```

この関数は左目用画像と右目用画像をレンダリングするためのプロジェクション行列とビュー行列の生成に使います。

この関数では液晶から最適視認距離ほど離れた位置にユーザーがいると想定しています。現実に即したビューも、以下で説明する setting 引数に基づいた現実に即したわけではない(芸術的な)ビューも生成可能です。加えて、この関数はユーザーの頭の位置を考慮してダイナミックパースペクティブを調整することも可能です。

leftEyeTangent 引数と rightEyeTangent 引数はスクリーンに対するユーザーの左目の位置と右目の位置を表します。これらの値は nn::qtm::CTR::GetTrackingData 関数で取得できます。

leftEyeTangent と rightEyeTangent のいずれかが null の場合、この関数の動作は(ダイナミックではない)静的な立体視カメラと同じになります。CTR 本体向けの開発であれば、この引数は null となるはずです。

生成されるプロジェクション行列のカメラの上向き方向が、pivot 引数で規定される上向き方向になります。pivot 引数に nn::math::PIVOT_NONE が設定された場合、回転は適用されません。pivot 引数に無効な値が指定されたら、関数はアサーションで止まります。デフォルトでは nn::math::PIVOT_UPSIDE_TO_TOP が指定されます。

返り値はカメラ間の距離の係数 f_Iです。付録で説明します。

6.4. Setting クラス

コード 6-7. Setting クラスの定義

```
struct Setting
{
    f32 StereoFactor;
    f32 ParallaxDistanceLimit;
    f32 DynamicPerspectiveAmplitudeX;
    f32 DynamicPerspectiveAmplitudeY;
    f32 DynamicPerspectiveClampingLeft;
    f32 DynamicPerspectiveClampingRight;
    f32 DynamicPerspectiveClampingBottom;
    f32 DynamicPerspectiveClampingTop;
    f32 RealisticFovFactor;
    f32 WideFovMotionFactor;
    bool TiltEnabled;
}
```

ComputeViewsAndProjections 関数の Setting 引数を使って立体視カメラの振る舞いを調整できます。

- StereoFactor は 4.7. 3D ボリュームと立体視係数 節で説明したように立体視効果を調整します。
- ParallaxDistanceLimit はミリメートルで指定する視差の限界値です。左目用画像のオブジェクトと右目用画像のオブジェクトの間の最大視差を意味します。4.6. 限界視差 節に説明があります
- DynamicPerspectiveAmplitudeX と DynamicPerspectiveAmplitudeY はユーザーの位置に乗算される係数です。頭の動きに対してダイナミックパースペクティブの効果を増幅させたり低減させたりするのに使います。
- DynamicPerspectiveClampingLeft、DynamicPerspectiveClampingRight、DynamicPerspectiveClampingBottom、DynamicPerspectiveClampingTop はユーザー位置の水平方向および垂直方向のクランプに使います。これらは 0 から 1 の範囲の値をとります。1 が設定された場合は、GetTrackingData で取得できる限界値で位置がクランプされます。中間の値が設定された場合は、限界値にたいして設定値に比例した位置でクランプされます。0 が設定された場合は、カメラは指定の方向には動かなくなります。DynamicPerspectiveAmplitude による増幅の前にクランプがされることに注意してください。
- RealisticFovFactor は 4.2. 現実に即した FOV 係数 節で説明した現実に即したカメラとベースカメラの間にくるカメラ位置の補間に使われます。
- WideFovMotionFactor は 4.1. ワイド FOV 節で説明したワイド FOV の状況で、ユーザーの動き応じてカメラがどのように動くかを調整します。
- TiltEnabled は 4.5. 傾きと立体カメラの改善 節で説明した、傾きの有効無効を示します。

6.5. 占有領域

コード 6-8. OccupancyVolume 構造体の定義

```
struct OccupancyVolume
(
    math::VEC3 CameraCenter;
    math::VEC3 ForwardDirection;
    math::VEC3 OrthoUpDirection;
    math::VEC3 RightDirection;
    f32 NearPlaneDepth;
    f32 NearLeft;
    f32 NearRight;
    f32 NearBottom;
    f32 NearTop;
    f32 CameraLeft;
    f32 CameraRight;
    f32 CameraBottom;
    f32 CameraTop;
};
```

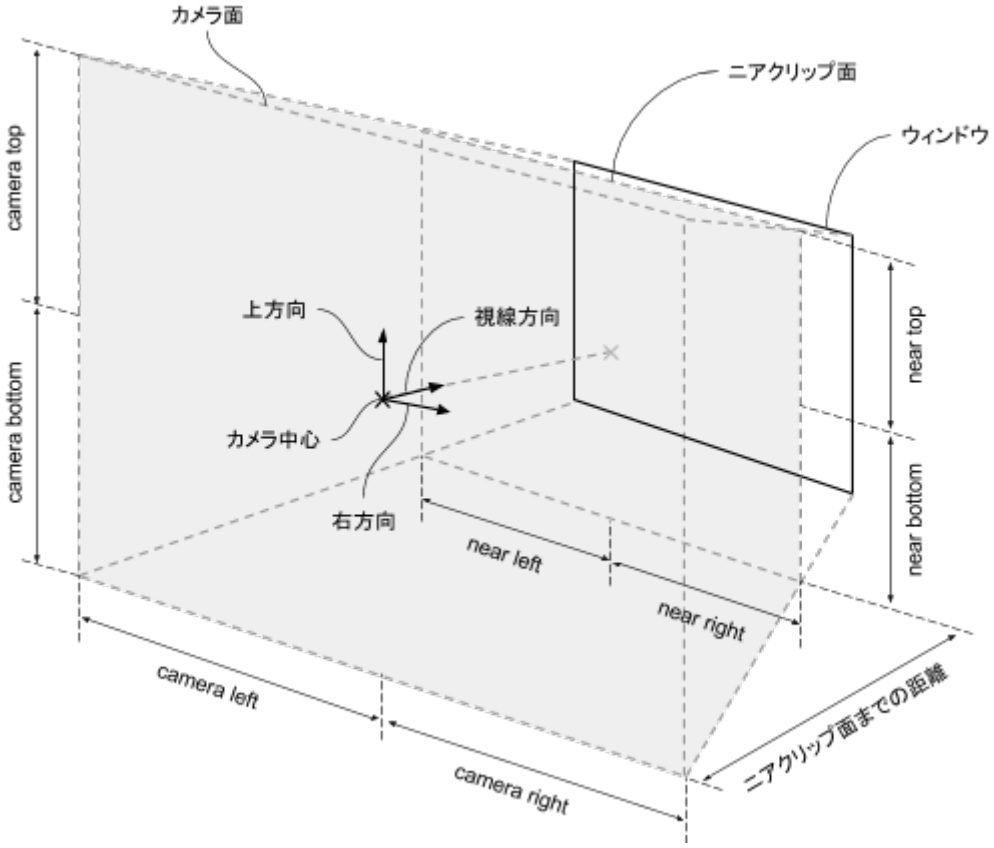
OccupancyVolume 構造体を使ってカメラの占有領域を表現できます。オブジェクトと交わることで不自然な見た目にならないことを保証するため、この領域にオブジェクトを置くべきではありません。

- CameraCenter はダイナミックパースペクティブを無効化したときのカメラ位置です。RealisticFovFactor が 0 の場合、この位置は SetView メソッドで指定される位置と同じです。0 より大きい場合、前方向の軸に沿ってカメラ位置が調整され、現実に即した FOV に近づきます。ダイナミックパースペクティブでカメラが動ける面のことをカメラ面と呼びます。
- ForwardDirection は前方向を示す単位ベクトルです。SetView メソッドで指定された値と同じです。
- OrthoUpDirection は上方向を示す単位ベクトルです。ForwardDirection と直交します。
- RightDirection は右方向を示す単位ベクトルです。ForwardDirection と OrthoUpDirection と直交するため、それらの外積を計算することで、右方向のベクトルを求めることができます。

  $$RightDirection = ForwardDirection \times OrthoUpDirection$$

- NearPlaneDepth はカメラ位置からニアクリップ面までの距離です。
- NearLeft と NearRight はニアクリップ面における、(右方向ベクトルに沿った)水平方向の占有領域範囲です。
- NearBottom と NearTop はニアクリップ面における、(上方向ベクトルに沿った)垂直方向の占有領域範囲です。
- CameraLeft と CameraRight はカメラ面における水平方向の占有領域範囲です。
- CameraBottom と CameraTop はカメラ面における垂直方向の占有領域範囲です。

図 6-1. 占有領域のパラメータ



7. 付録:公式

7.1. 定数

現在のシステムでは下記の値を使っています。

I_{eye} 標準的なユーザーの両目の間隔 62 mm

d_{ideal} 上画面とユーザーの標準的な距離 316.5 mm (SNAKE 301 mm、CLOSER 332 mm の平均値)

W_{screen} スクリーンの幅 95.4 mm (SNAKE 84.6 mm、CLOSER 106.2 mm の平均値)

H_{camera} スクリーン中心とカメラ中心との距離 37.62 mm (SNAKE 34.38 mm、CLOSER 40.86 mm の平均値)

7.2. 仮想空間と現実空間の変換

図 7-1. 現実空間の座標軸(左)と仮想空間の座標軸(右)

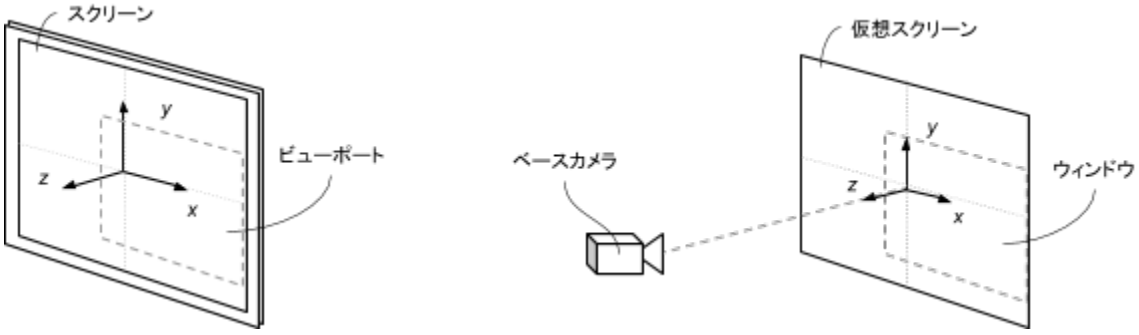


図 7-1 に現実空間と仮想空間の座標軸を定義しました。どちらの座標軸もスクリーンの中心に原点があります。

カメラ位置のウィンドウへの正射影はウィンドウの中央だと想定します。

現実空間の座標から仮想空間の座標への距離を変換するためのスケールを示します。

$$scale_{r2v} = \frac{W_{window}}{W_{viewport}}$$

W_{window} は仮想空間でのウィンドウの幅です。

$W_{viewport}$ は現実空間でのビューポートの幅です。

7.3. ユーザー位置

現実空間でのユーザーの目の位置を $X_{lefteye}$ 、$X_{righteye}$ と定義します。これらの位置はタンジェント、つまり角度でのみ知ることができます。

$$(u_{left\ eye}, v_{left\ eye})$$

$$(u_{right\ eye}, v_{right\ eye})$$

ユーザーは立体視効果を最も楽しめる最適視認距離にいると想定することで、ユーザーの目の位置を推定しています。

$$X_{left\ eye} = (u_{left\ eye} \cdot d_{ideal}, v_{left\ eye} \cdot d_{ideal} + scale_{r2v} \cdot H_{camera}, d_{ideal})$$
$$X_{right\ eye} = (u_{right\ eye} \cdot d_{ideal}, v_{right\ eye} \cdot d_{ideal} + scale_{r2v} \cdot H_{camera}, d_{ideal})$$

これらの位置をユーザー位置 X_{user} と両目間ベクトル（左目から右目のベクトル）Δ_{eyes} に分離します。

$$X_{user} = \frac{1}{2}(X_{left\ eye} + X_{right\ eye})$$
$$\Delta_{eyes} = X_{right\ eye} - X_{left\ eye}$$

両目間ベクトル方向の単位ベクトルを δ_{eyes} と呼びます。

$$\delta_{eyes} = \frac{\Delta_{eyes}}{||\Delta_{eyes}||}$$

ユーザーの位置は標準的な位置からの変位 X'_{user} として表現することもあります。

$$X'_{user} = X_{user} - (0,0,d_{ideal})$$

7.4. カメラ間の距離の計算

カメラ間の距離 I は左目画像用のカメラと右目画像用のカメラとの間の距離です。

$$I = ||X_{right\ cam} - X_{left\ cam}||$$

現実に即した構成を選んだ場合、この距離は現実に即したカメラ間距離 $I_{realistic}$ と等しくなります。カメラ間の距離 I_{eye} から計算可能です。

$$I_{realistic} = scale_{r2v} \cdot I_{eye} \cdot I_{amp}$$

7.4.1. ワイド FOV の計算

カメラ間の距離が理想的な距離ではない場合（ワイド FOV 構成の場合）、カメラ間の距離は縮小されなければなりません。

$$k_{fov} = \frac{d_{window}}{scale_{r2v} \cdot d_{ideal}}$$

d_{window} はウィンドウとカメラの距離で、開発者が設定した $d_{window\ base}$ と仮想空間での標準的な距離 d_{ideal} との線形補間で計算されます。

$$d_{window} = lerp(d_{window\ base}, scale_{r2v} \cdot d_{ideal}, f_{realistic})$$

$f_{realistic}$ は現実に即した FOV 係数です。

7.4.2. 限界視差の計算

ユーザーが快適にプレイできるように、開発者は限界視差 P_{limit} を指定することができます。カメラ間距離の最大値 I_{max} はこの制限から導出できます。

$$I_{max} = \frac{d_{far}}{d_{far} - d_{window}} \cdot scale_{r2v} \cdot P_{limit}$$

d_{far} はファークリップ面の深度で、開発者が指定した $d_{far\ base}$ から導出できます。

$$d_{far} = d_{far\ base} + d_{window} - d_{window\ base}$$

7.4.3. カメラ間の距離係数

最終的なカメラ間の距離 I がレンダリングに使われます。

$$I = f_{3D} \cdot f_{stereo} \cdot min(k_{fov} \cdot I_{realistic}, I_{max})$$

現実に即したカメラ間距離からどれだけ縮小されたかを示す、カメラ間の距離係数 f_I は、I から算出可能です。

$$f_I = \frac{I_{realistic}}{I}$$

7.5. カメラ中心位置の計算

仮想空間での左目カメラと右目カメラの位置を $X_{leftcam}$ と $X_{rightcam}$ とします。2つのカメラの中間点となる位置を X_{cam} と定義します。

$$X_{cam} = \frac{1}{2}(X_{left\ cam} + X_{right\ cam})$$

仮想空間での z 軸上の標準距離にリファレンスカメラ位置 X_{camref} を置きます。

$$X_{cam\ ref} = (0,0,d_{window})$$

X_{cam} はリファレンスカメラ位置からユーザー位置の変位を計算することで得られます。

$$X_{cam} = X_{cam\ ref} + f_{amplitude} \cdot k_{fov\ motion} \cdot scale_{r2v} \cdot clamp(X'_{user})$$

$f_{amplitude}$ はダイナミックパースペクティブ振幅係数です。

$k_{fov\ motion}$ はワイド FOV の運動係数 k_{wide} から算出される振幅係数です。

$$k_{fov\ motion} = lerp(k_{fov}, 1, k_{wide})$$

$clamp(X'_{user})$ は検知可能な最大ユーザー変位をクランプした、ユーザー変位です。

7.6. 最終的な目のカメラ位置の計算

最終的な目のカメラ位置は中央のカメラ位置からの変位を計算することで求めます。傾きを考慮することも可能ですし、傾きパラメータに依存しなくても構いません。

7.6.1. 傾きが有効な場合

$$X_{left\ cam} = X_{cam} - \frac{1}{2} \cdot I \cdot \delta_{eyes}$$

$$X_{right\ cam} = X_{cam} + \frac{1}{2} \cdot I \cdot \delta_{eyes}$$

7.6.2. 傾きが無効な場合

$$X_{left\ cam} = X_{cam} - \frac{1}{2} \cdot I \cdot (1,0,0)$$

$$X_{right\ cam} = X_{cam} + \frac{1}{2} \cdot I \cdot (1,0,0)$$

更新履歴

Version 1.0 2015-01-15

追加/変更

- 初版