

View 関数について

初期化の方法

ビューの初期化は njSetView()関数を実行する前に必ず済ませておかねばなりません。
初期化には次の 2 通りの方法があります。

1. njInitView()を使用する。

これによりビューは

現在の視点の位置 (px, py, pz) = (0,0,0)
現在の視点の向き (vx, vy, vz) = (0,0,-1)
現在の視点の傾き (roll、視線の Z 軸に対する傾き) 0 度
基の視点の位置 (apx, apy, apz) = (0,0,0)
基の視点の向き (avx, avy, avz) = (0,0,-1)
基の視点の傾き (aroll、視線の Z 軸に対する傾き) 0 度
ビューマトリクス = 単位マトリクス

に設定されます。

2. VIEW 構造体のメンバーに直接値をセットする。

セットしなければならないのは、

相対的な操作を行う場合は、

- px, py, pz
- vx, vy, vz
- roll

絶対的な操作を行う場合は、

- px, py, pz
- vx, vy, vz
- roll

以上に直接値をセットした後、void njSetBaseView(NJS_VIEW *v) を実行。

もしくは

- apx, apy, apz
- avx, avy, avz
- aroll

にもそれぞれ同じ値をセットします。

ビューマトリクスの設定は必要ありません。

% 注 %

視線ベクトルは必ず単位ベクトル化してください。

ビューの移動、回転

全ての `nj*Relative()` 及び `nj*Absolute()`関数は `njSetView()`関数の前に実行されなければなりません。

例)

ビューの初期化。

```
        .  
        .  
while(1){  
    njSetView();  
    .  
    .  
    .  
    描画。  
    .  
    . <-----ここで nj*Relative() / nj*Absolute()を実行、次の視点へ。  
    .  
}
```

重要

プログラムの流れとして、必ず `nj*Relative()`、もしくは `nj*Absolute()`関数が `njSetView()`関数より先に実行されます。

旧 View 関数使用時の注意

njMultiViewMatrix
njRotateViewX
njRotateViewY
njRotateViewZ
njRotateViewXYZ
njTranslateView
njTranslateViewV
njUnitViewMatrix

以上の 8 関数はこれまでの Ninja との互換性を保つ為に残してある関数です。
将来ライブラリから削除される可能性がある為、新しくプログラムを組む場合には
使用しないようお願いします。
また、やむなく使用する際には次の注意事項を守るようお願いします。

- 1) njSetView()関数の後で実行する。
- 2) 上記関数の実行後、必ず njClearMatrix()関数を実行する。

以上の 2 点を守らない場合は動作の保証がされません。

1. 注意事項 1) の詳細

旧 VIEW 関数は VIEW 構造体のメンバー、NJS_MATRIX m を直接操作しますが、
現在の VIEW では

Float	px,py,pz;	// 現在の視点の位置
Float	vx,vy,vz;	// 現在の視点の向き (ベクトル)
Angle	roll;	// 現在の視線の Z 軸に対する傾き
Float	apx,apy,apz;	// 基の視点の位置
Float	avx,avy,avz;	// 基の視点の向き (ベクトル)
Angle	aroll;	// 基の視線の Z 軸に対する傾き

の各メンバーを操作し、メンバー、NJS_MATRIX m は njSetView()関数によって
更新されます。

従って、njSetView()関数を実行する以前に旧 VIEW 関数を用いて VIEW の操作を
行ってもマトリクスは njSetView()関数によって上書きされてしまいます。

2. 注意事項 1) の詳細

VIEW は必ずマトリクススタックに反映されなければなりません。

この機能は現在 njSetView()関数に組み込まれていますが、上記の理由から njSetView()関数を用いて VIEW をマトリクススタックに反映させることは出来ません。そのため旧 VIEW 関数を使用する場合、旧 MATRIX 関数、njClearMatrix()の実行を省略することは出来ません。

3. 旧ビュー関数の使用例 (正しい例 / 誤った例)

正しい旧 VIEW 関数の使用例

```
NJS_VIEW _view_;

njInitView(&_view_);
njSetView(&_view_);
njTranslateView(&_view_, 0.f, 0.f, 1000.f); // etc
njClearMatrix();
```

誤った旧 VIEW 関数の使用例 1

```
NJS_VIEW _view_;

njInitView(&_view_);
njTranslateView(&_view_, 0.f, 0.f, 1000.f); // etc
njSetView(&_view_);
njClearMatrix();
```

njSetView()関数によって VIEW のマトリクスが上書きされてしまう為 VIEW は初期化状態のままになってしまいます。

誤った旧 VIEW 関数の使用例 2

```
NJS_VIEW _view_;

njInitView(&_view_);
njSetView(&_view_);
njTranslateView(&_view_, 0.f, 0.f, 1000.f); // etc
```

njTranslateView()関数を実行した結果がマトリクススタックに
反映されていない為 VIEW は初期化状態のままになってしまいます。

/* VIEW 構造体 */

```
typedef struct {
    Float    px,py,pz;        // 現在の視点の位置
    Float    vx,vy,vz;        // 現在の視点の向き（ベクトル）
    Angle    roll;            // 現在の視線のZ軸に対する傾き
    Float    apx,apy,apz;     // 基の視点の位置
    Float    avx,avy,avz;     // 基の視点の向き（ベクトル）
    Angle    aroll;           // 基の視線のZ軸に対する傾き
    NJS_MATRIX m;             // ビューマトリクス
} NJS_VIEW;構造体

typedef struct {
    Float    px,py,pz;        // 現在の視点の位置
    Float    vx,vy,vz;        // 現在の視点の向き（ベクトル）
    Angle    roll;            // 現在の視線のZ軸に対する傾き
    Float    apx,apy,apz;     // 基の視点の位置
    Float    avx,avy,avz;     // 基の視点の向き（ベクトル）
    Angle    aroll;           // 基の視線のZ軸に対する傾き
    NJS_MATRIX m;             // ビューマトリクス
} NJS_VIEW;
```