



コンピュータグラフィックス特論Ⅱ

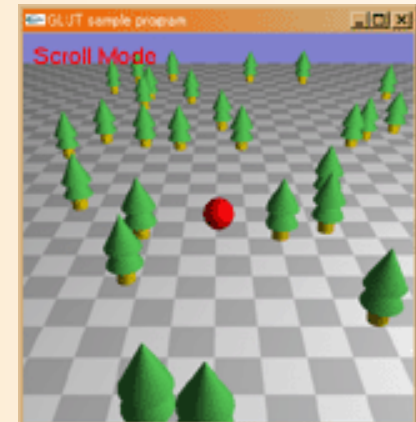
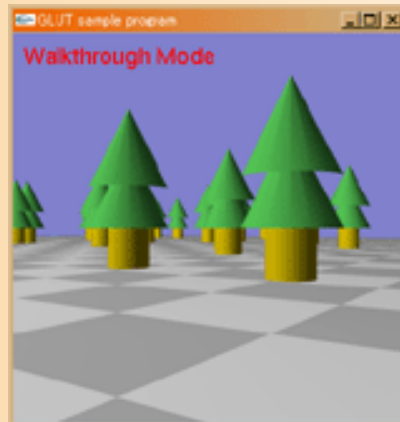
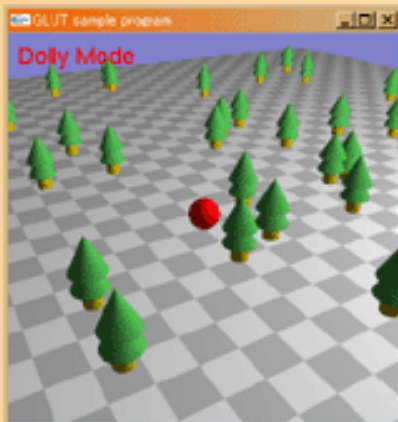
第3回 視点操作

九州工業大学 尾下 真樹

2021年度

視点操作

- 視点操作
 - 利用者が視点を操作して、仮想空間や物体を適切な位置・方向から見るための機能
 - 適切な視点操作のインターフェースや実現方法は、アプリケーションによっても異なる
 - 変換行列による、代表的な視点操作の実現方法

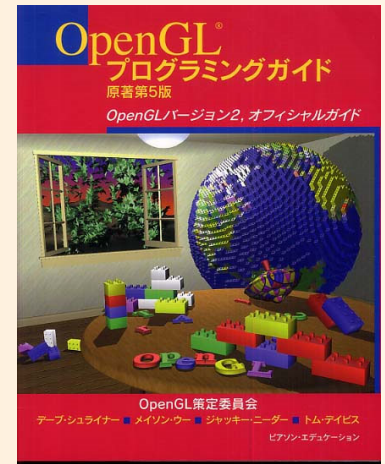


今日の内容

- 前回のサンプルプログラムの視点処理(復習)
- 視点操作のプログラム
- 視点操作方法1 (Dolly Mode)
 - 媒介変数を使う方法
 - 変換行列を直接更新する方法
- 視点操作方法2 (Scroll Mode)
- 視点操作方法3 (Walkthrough Mode)
- レポート課題



参考書



- 最低限の関数の使い方は資料を用意
- OpenGLの定番の本(高い)
 - OpenGLプログラミングガイド(赤本), 12,000円
 - OpenGLリファレンスマニュアル(青本), 8,300円
 - ピアソン・エデュケーション出版
- グラフィックスS(システム創成3年前期) 演習資料
 - <http://www.cg.ces.kyutech.ac.jp/lecture/cg/>
 - OpenGLの使い方を段階的に学べるチュートリアル
 - OpenGLに不慣れな人は一通り試しておくことを推奨
- 適当な入門書
 - 他にもOpenGLの入門書は多数ある



視点操作の方法

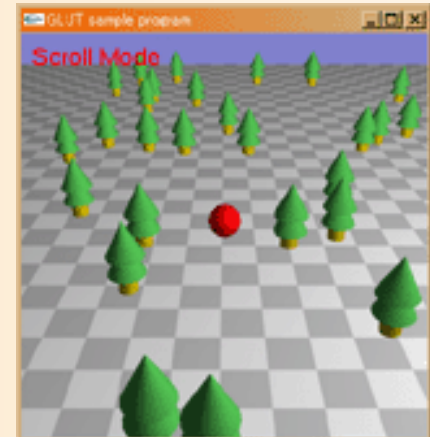
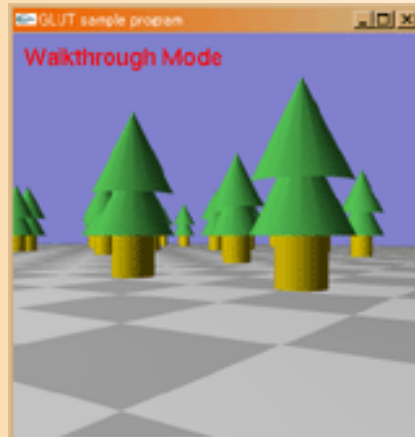
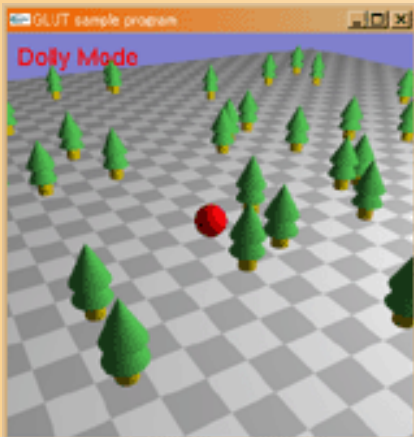
- 既存のアプリケーションでよく使われている代表的な視点操作方法
 - 方法1: 注視点の周囲を回るように視点を回転・移動 (Dolly Mode)
 - 方法2: 注視点に合わせて視点を平行移動、視点の向きは固定 (Scroll Mode)
 - 方法3: カメラを中心として視点を回転・移動 (Walkthrough Mode)



デモプログラム

- 視点操作

- mキーで視点操作モードを切り替え
方法1(Dolly) → 方法2(Scroll) → 方法3
(Walkthrough) の順番で切り替わる
- マウスの右ボタン・左ボタンドラッグで、各視点
操作モードに応じて視点変更



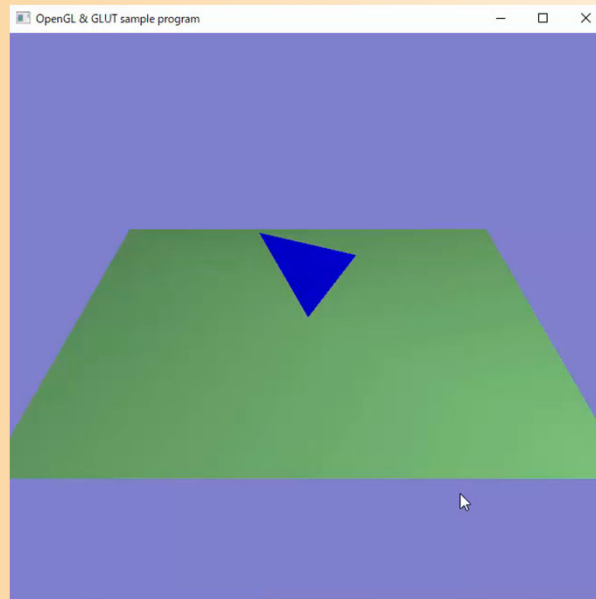




前回のサンプルプログラムの 視点操作処理(復習)

サンプルプログラム

- `opengl_sample.c`
 - OpenGL + GLUT を使った基本的なプログラム
 - 地面と1枚の青い三角形が表示される
 - マウスの右ボタンドラッグで、視点を上下に回転



サンプルプログラム

- `opengl_sample.c`

```
opengl_sample.c

1 //
2 // コンピュータグラフィックスS
3 // OpenGLによる3次元グラフィックス演習 サンプルプログラム
4 //
5
6
7 // 基本的なヘッダファイルのインクルード
8 #ifdef _WIN32
9     #include <windows.h>
10 #endif
11 #include <stdio.h>
12 #include <math.h>
13
14 // GLUTヘッダファイルのインクルード
15 #include <GL/glut.h>
16
17
18 // 視点操作のための変数
19 float camera_pitch = -30.0; // X軸を軸とするカメラの回転角度
20
21 // マウスのドラッグのための変数
22 int drag_mouse_r = 0; // 右ボタンをドラッグ中かどうかのフラグ (0:非ドラッグ中,1:ドラッグ中)
23 int last_mouse_x; // 最後に記録されたマウスカーソルのX座標
24 int last_mouse_y; // 最後に記録されたマウスカーソルのY座標
25
26
27 //
28 // 画面描画時に呼ばれるコールバック関数
29 //
30 void display( void )
31 {
32     // 画面をクリア (ピクセルデータとZバッファの両方をクリア)
33     glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
34
35     // 変換行列を設定 (ワールド座標系→カメラ座標系)
36     glMatrixMode( GL_MODELVIEW );
37     glLoadIdentity();
38     glTranslatef( 0.0, 0.0, - 15.0 );
39     glRotatef( - camera_pitch, 1.0, 0.0, 0.0 );
40
41     // 光源位置を設定 (モデルビュー行列の変更にあわせて再設定)
42     float light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
43     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
44
```



現在の視点操作の実現方法

- マウスの右ボタンを押しながら、上下にドラッグすると、カメラの回転角度(仰角)が変化
- カメラの回転角度を表す変数 `camera_pitch` を定義
- マウス操作に応じて、`camera_pitch` の値を変化
- `camera_pitch` に応じて、変換行列を設定



視点操作のための変数

- 視点操作のための変数の定義
 - グローバル変数(全ての関数からアクセス可能な変数)として定義

```
// 視点操作のための変数
```

```
float camera_pitch = -30.0; // X軸を軸とするカメラの回転角度
```

```
// マウスのドラッグのための変数
```

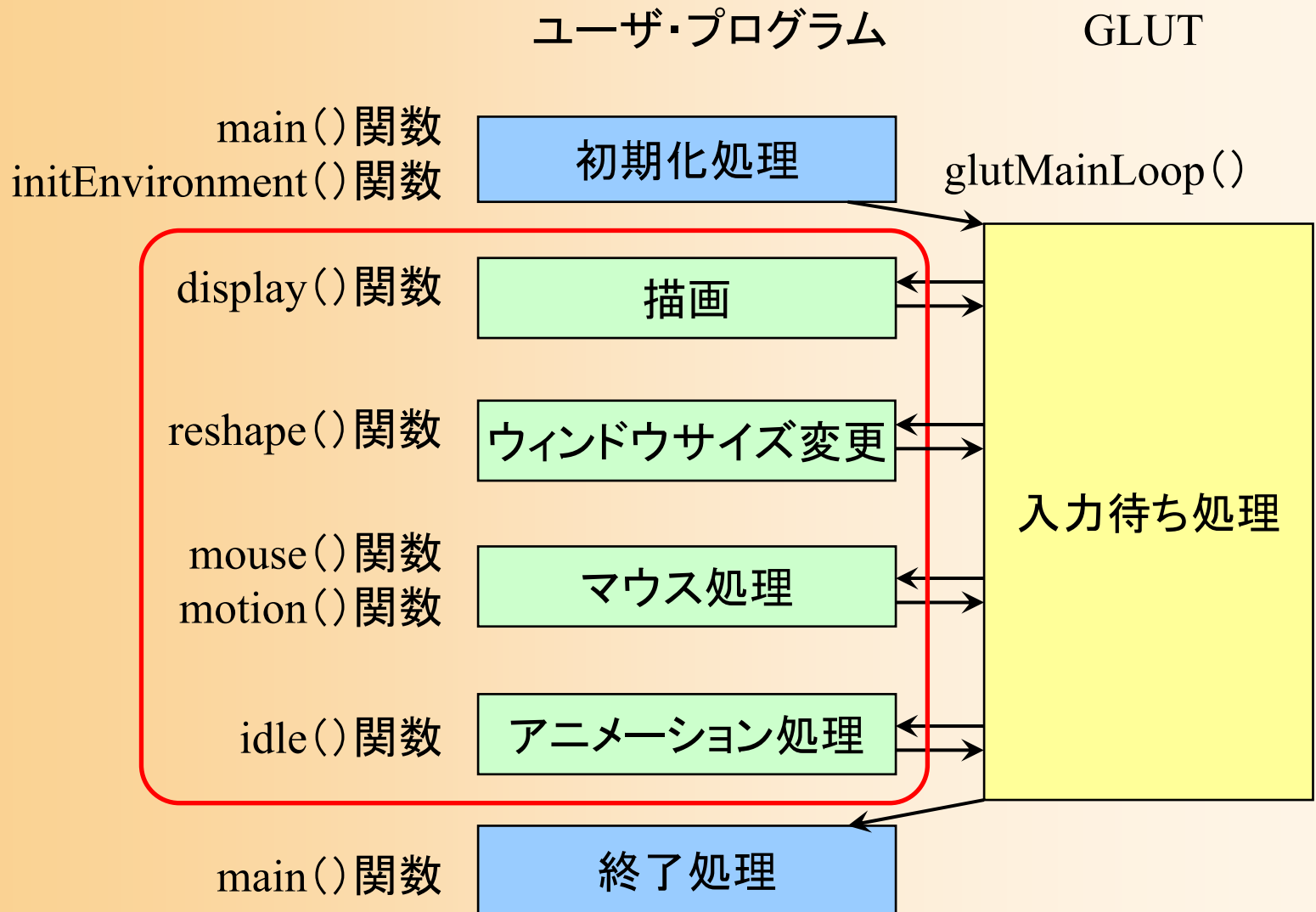
```
int drag_mouse_r = 0; // 右ボタンをドラッグ中かどうかのフラグ  
(0:非ドラッグ中,1:ドラッグ中)
```

```
int last_mouse_x; // 最後に記録されたマウスカーソルのX座標
```

```
int last_mouse_y; // 最後に記録されたマウスカーソルのY座標
```



サンプルプログラムの構成(復習)



コールバック関数(1)(復習)

- 描画コールバック関数 `display()`
 - 再描画が必要な時に呼ばれる
 - 本プログラムでは、変換行列の設定、地面と1枚のポリゴンの描画、を行っている
- サイズ変更コールバック関数 `reshape()`
 - ウィンドウサイズ変更時に呼ばれる
 - 本プログラムでは、視界の設定、ビューポート変換の設定、を行っている

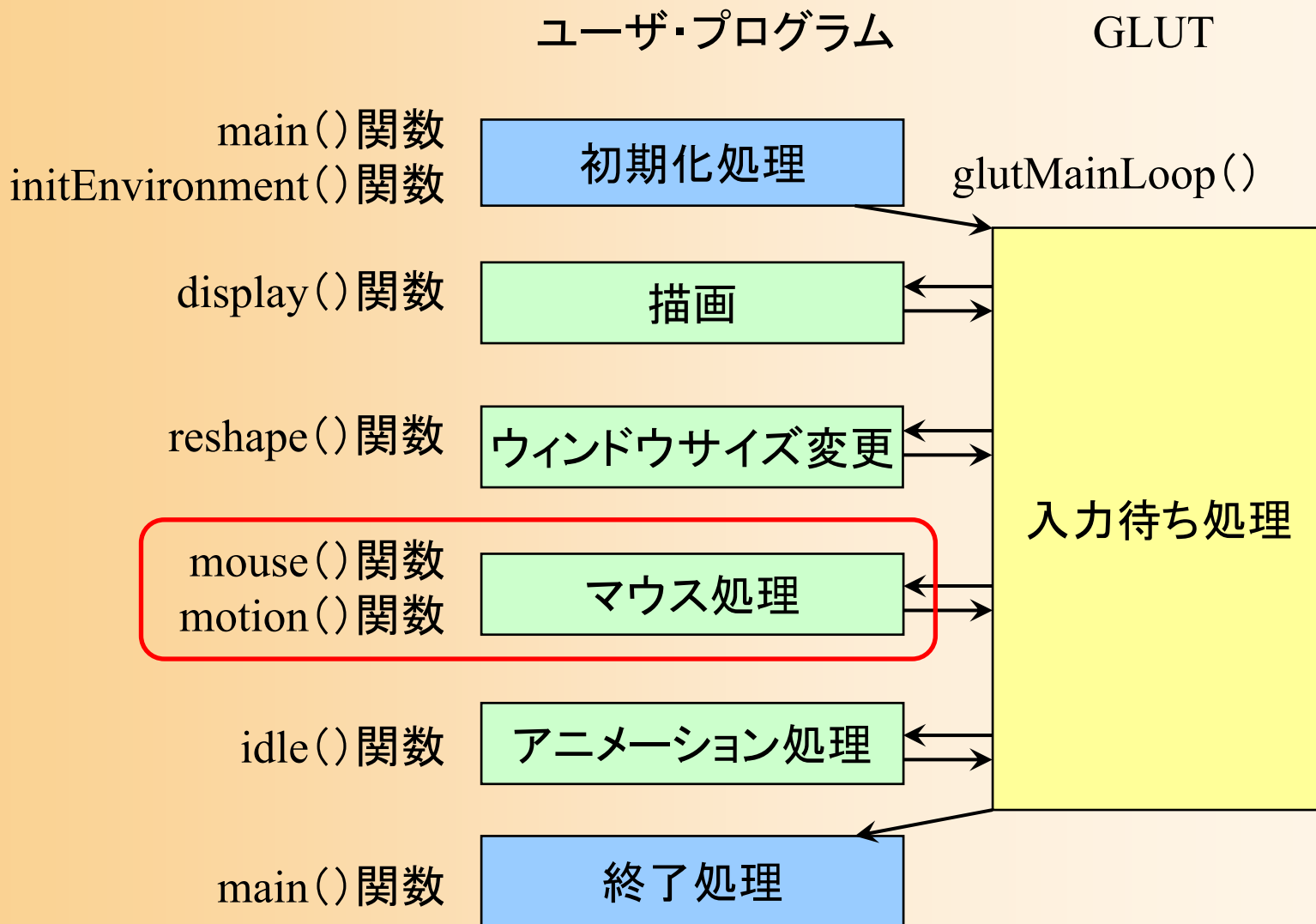


コールバック関数(2)(復習)

- マウスクリック・コールバック関数 `mouse()`
 - マウスのボタンが押されたとき、離されたときに呼ばれる
 - 本プログラムでは、右ボタンの押下状態を記録
- マウสดラッグ・コールバック関数 `motion()`
 - マウスがウィンドウ上でドラッグされたときに呼ばれる
 - 本プログラムでは、右ドラッグされたときに、視点の回転角度を変更
- アイドル・コールバック関数 `idle()`
 - 処理が空いた時に定期的に呼ばれる
 - 本プログラムでは、現在は何の処理も行っていない

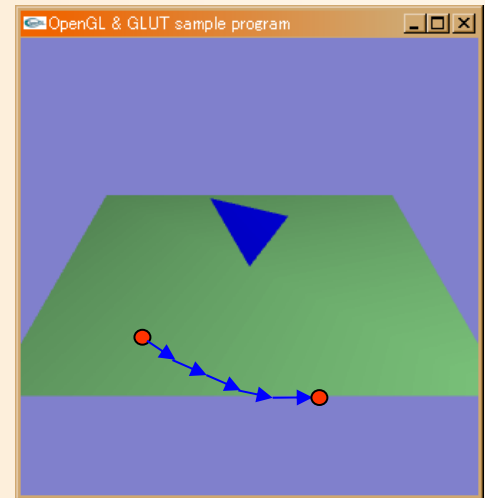


マウス操作時の処理



マウス操作時の処理

- マウス操作のコールバック関数
 - mouse() 関数
 - マウスのボタンが、**押されたとき**、または、**離されたとき**に呼ばれる
 - motion() 関数
 - マウスのボタンが押された状態で、マウスが**動かされたとき**(ドラッグ時)に定期的に呼ばれる
 - ボタンが押されない状態で、マウスが動かされたときに呼ばれる関数もある(今回は使用しない)



マウス操作時の処理(クリック処理関数)

- 右ボタンがクリックされたことを記録
 - 変数 `drag_mouse_r` に状態を格納

```
// マウスクリック時に呼ばれるコールバック関数
void mouse( int button, int state, int mx, int my )
{
    // 右ボタンが押されたらドラッグ開始のフラグを設定
    if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
        drag_mouse_r = 1;
    // 右ボタンが離されたらドラッグ終了のフラグを設定
    else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
        drag_mouse_r = 0;

    // 現在のマウス座標を記録
    last_mouse_x = mx;
    last_mouse_y = my;
}
```

マウス操作時の処理(ドラッグ処理関数1)

- ドラッグされた距離に応じて視点を変更
 - 視点の仰角 `camera_pitch` を変化
 - 前回と今回のマウス座標の差から計算

```
void motion( int mx, int my )
{
    // 右ボタンのドラッグ中であれば、
    // マウスの移動量に応じて視点を回転する
    if ( drag_mouse_r == 1 )
    {
        // マウスの縦移動に応じてX軸を中心に回転
        camera_pitch -= ( my - last_mouse_y ) * 1.0;
        if ( camera_pitch < -90.0 )
            camera_pitch = -90.0;
        else if ( camera_pitch > 0.0 )
            camera_pitch = 0.0;
    }
    .....
}
```

マウス操作時の処理(ドラッグ処理関数2)

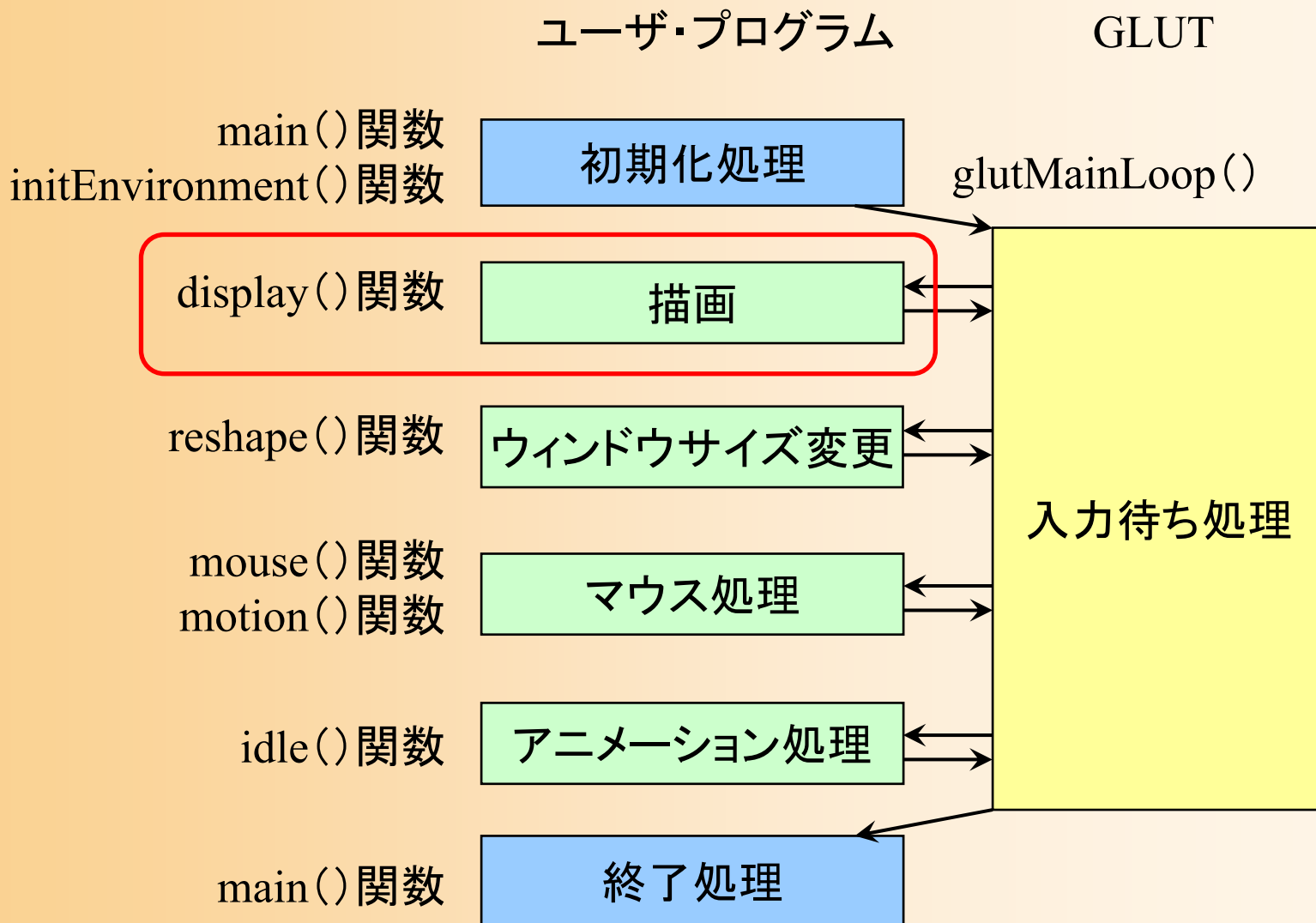
- 再描画の指示を行う
 - 視点の仰角camera_pitch の変化に応じて、画面を再描画するため

```
// 今回のマウス座標を記録
last_mouse_x = mx;
last_mouse_y = my;

// 再描画の指示を出す
glutPostRedisplay();
}
```

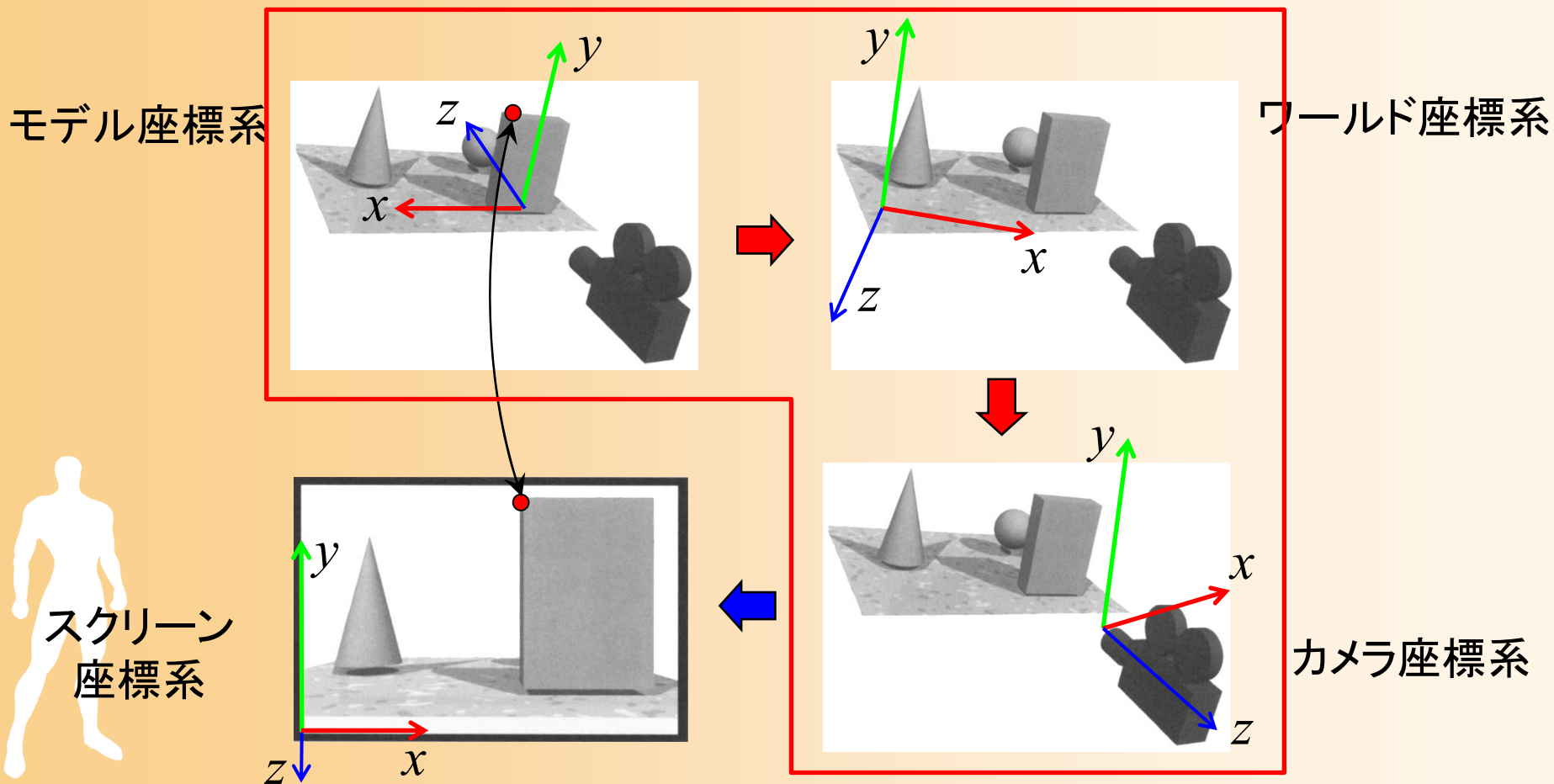


サンプルプログラムの構成



視野変換(復習)

- モデル座標系からカメラ座標系に変換




同次座標変換(復習)

- 同次座標変換

- 4 × 4行列の演算により、3次元空間における
平行移動・回転・拡大縮小(アフィン変換)などの
操作を統一的に実現

- (x, y, z, w) の4次元座標値(同次座標)を扱う
- 3次元座標値は(x/w, y/w, z/w)で計算(通常は w = 1)


$$\begin{pmatrix} R_{00}S_x & R_{01} & R_{02} \\ R_{10} & R_{11}S_y & R_{12} \\ R_{20} & R_{21} & R_{22}S_z \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} T_x \\ T_y \\ T_z \\ 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$

変換行列による座標変換(復習)

- 視野変換 + 射影変換

- アフィン変換(視野変換) + 透視変換(射影変換)
- 最終的なスクリーン座標は $(x'/w' \ y'/w' \ z'/w')$ となる

モデル座標系での頂点座標

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} R_{00}S_x & R_{01} & R_{02} & T_x \\ R_{10} & R_{11}S_y & R_{12} & T_y \\ R_{20} & R_{21} & R_{22}S_z & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$

射影変換
(カメラ→スクリーン)

視野変換
(モデル→カメラ)

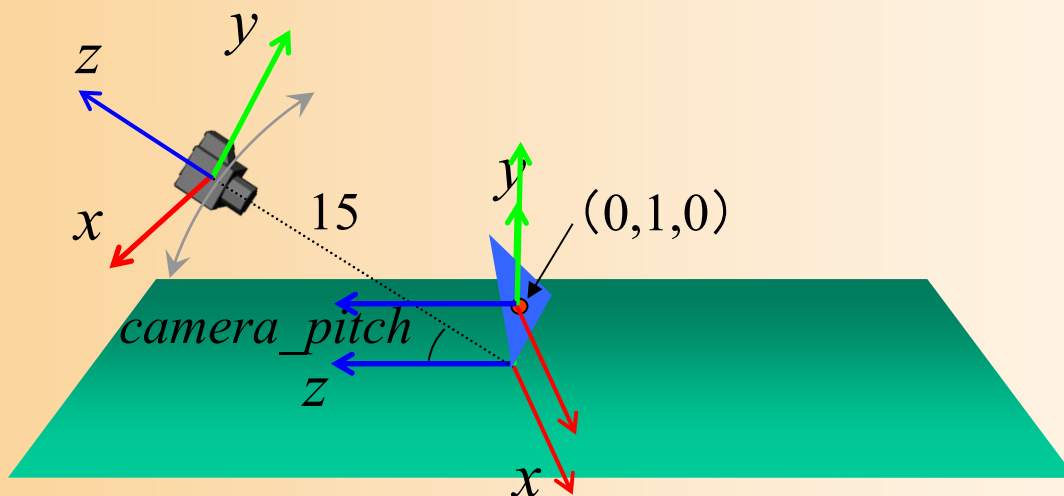
スクリーン座標系
での頂点座標



サンプルプログラムの視野変換行列

- サンプルプログラムのシーン設定

- カメラと水平面の角度(仰角)は `camera_pitch`
- カメラと原点の間の距離は 15
- ポリゴンを $(0,1,0)$ の位置に描画



サンプルプログラムの視野変換行列

- モデル座標系 → カメラ座標系 への変換行列

$$\begin{matrix} \textcircled{3} & & \textcircled{2} & & \textcircled{1} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -15 \\ 0 & 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & & & \\ & \cos(-camera_pitch) & & \\ & \sin(-camera_pitch) & & \\ & & & 1 \end{pmatrix} & \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & 1 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} \end{matrix}$$

ワールド座標系→カメラ座標系

モデル座標系→ワールド座標系

– x 軸周りの回転

– 2つの平行移動変換の位置に注意

- 原点から15離れるということは、回転後の座標系でカメラを後方(z 軸)に15下げることと同じ



サンプルプログラムの変換行列の設定

• 描画処理 (display() 関数)

```
// 変換行列を設定(ワールド座標系→カメラ座標系)
```

```
glMatrixMode( GL_MODELVIEW );
```

```
glLoadIdentity();
```

③ glTranslatef(0.0, 0.0, - 15.0);

② glRotatef(- camera_pitch, 1.0, 0.0, 0.0);

```
// 地面を描画(ワールド座標系で頂点位置を指定)
```

```
.....
```

```
// 変換行列を設定(モデル座標系→カメラ座標系)
```

① glTranslatef(0.0, 1.0, 0.0);

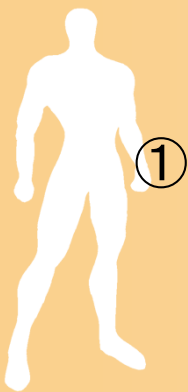
```
// ポリゴンを描画(モデル座標系で頂点位置を指定)
```

```
.....
```

以降、視野変換行列
を変更することを指定

単位行列で初期化

平行移動行列・
回転行列を順に
かけることで、
変換行列を設定





視点操作のプログラム

視点操作の方法

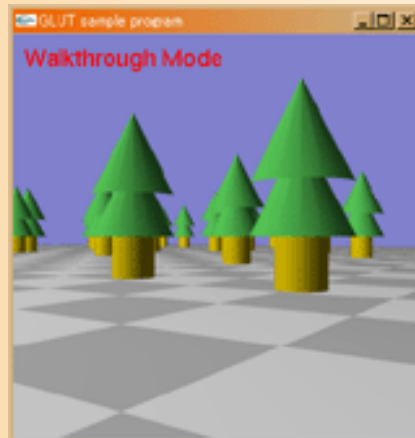
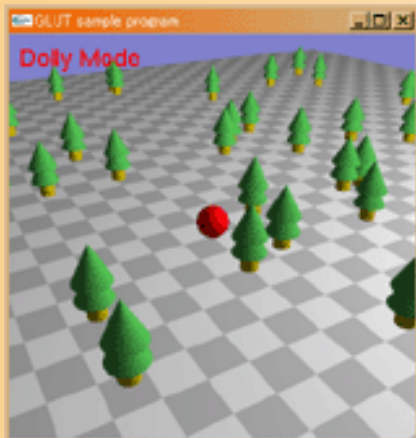
- 既存のアプリケーションでよく使われている、代表的な視点操作の方法を実現
 - 方法1: 注視点の周囲を回るように視点が回転・移動 (Dolly Mode)
 - 方法2: 注視点に合わせて視点が平行移動、視点の向きは固定 (Scroll Mode)
 - 方法3: カメラを中心に回るように視点が回転・移動 (Walkthrough Mode)



デモプログラム

- 視点操作のデモプログラム

- mキーで視点操作モードを切り替え
方法1(Dolly) → 方法2(Scroll) → 方法3
(Walkthrough) の順番で切り替わる
- マウスの右ボタン・左ボタンドラッグで、各視点
操作モードに応じて視点変更





サンプルプログラム

- `view_sample.cpp`
 - デモプログラムのもとになるプログラム
 - 全体の枠組みや一部の視点操作のみ実装済み
 - 残りの視点操作は、各自で実装する
(レポート課題)



視野変換行列の変更方法

- **方法1: 媒介変数を利用(変換行列を設定)**
 - 視点情報を媒介変数で管理する
 - 描画時に、媒介変数にもとづき、変換行列を設定
- **方法2: 変換行列を直接更新**
 - 視点情報を変換行列で管理する
 - 操作時に、変化分を適用し、変換行列を更新
- どちらの方法でも、同じ視点移動を実現可能
- 視点操作方法によってやりやすい方法が異なる




視点操作の実現方法

	媒介変数	直接更新
方法1: Dolly	サンプル	サンプル
方法2: Scroll	レポート課題	レポート課題
方法3: Walkthrough	レポート課題	レポート課題

サンプルプログラム(1)

- グローバル変数(視点操作パラメタ)
 - 全視点操作方法に使用する共通のパラメタ
 - 視点操作方法によっては変更しないパラメタもある

```
// 視点操作パラメタ
float view_center_x; // 注視点の位置  $x$ 
float view_center_y; // 注視点の位置  $y$ 
float view_center_z; // 注視点の位置  $z$ 
float view_yaw;      // 視点の方位角  $\alpha$ 
float view_pitch;    // 視点の仰角  $\beta$ 
float view_distance; // 視点と注視点の距離  $d$ 
```


$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\beta) & -\sin(-\beta) & 0 \\ 0 & \sin(-\beta) & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(-\alpha) & 0 & \sin(-\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\alpha) & 0 & \cos(-\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

サンプルプログラム(2)

- グローバル変数(視点操作モード)

```
// 視点操作モード
enum ViewControlModeEnum
{
    VIEW_DOLLY_PARAM,           // Dollyモード(媒介変数)
    VIEW_DOLLY_DIRECT,         // Dollyモード(直接更新)
    VIEW_SCROLL_PARAM,         // Scrollモード(媒介変数)
    VIEW_SCROLL_DIRECT,        // Scrollモード(直接更新)
    VIEW_WALKTHROUGH_PARAM,    // Walkthroughモード(媒介変数)
    VIEW_WALKTHROUGH_DIRECT,   // Walkthroughモード(直接更新)
    NUM_VIEW_CONTROL_MODES    // 視点操作モードの種類数
};

// 現在の視点操作モード
ViewControlModeEnum mode = VIEW_DOLLY_PARAM;
```



サンプルプログラム(3)

- 視点操作関連の関数

- 視点の初期化

- void InitView()
- プログラム開始時、モード切替時に呼ばれる

- 視点パラメタに応じて変換行列を更新

- void UpdateViewMatrix()
- 描画処理の最初に呼ばれる

- 操作に応じて視点パラメタ or 変換行列を更新

- void UpdateView(...)
- マウス操作時に呼ばれる



サンプルプログラム(4)

- 各視点操作の実現方法
- 方法1: 媒介変数を利用(変換行列を設定)
 - UpdateView()関数で、マウス操作に応じて媒介変数を更新
 - UpdateViewMatrix()関数で、媒介変数にもとづいて変換行列を設定(描画時に行列を設定)
- 方法2: 変換行列を直接更新
 - UpdateView()関数で、マウス操作に応じて変換行列を直接更新(マウス操作時に行列を設定)



サンプルプログラム(5)

- 操作に応じて視点パラメタ or 変換行列を更新
 - void UpdateView(int delta_mouse_right_x, int delta_mouse_right_y, int delta_mouse_left_x, int delta_mouse_left_y)
 - マウス操作を引数として受け取る
 - 右ドラッグ中の左右のマウス移動量
 - 右ドラッグ中の上下のマウス移動量
 - 左ドラッグ中の左右のマウス移動量
 - 左ドラッグ中の上下のマウス移動量
 - 媒介変数を使ったモード中は、視点操作パラメタを更新
 - 直接更新を使ったモード中は、変換行列を更新



サンプルプログラム(6)

- 6通りの各操作方法に対応する処理を記述

```
void UpdateView( int delta_mouse_right_x, int delta_mouse_right_y,
                 int delta_mouse_left_x, int delta_mouse_left_y )
{
    // 視点パラメタを更新(Dollyモード・媒介変数)
    if ( mode == VIEW_DOLLY_PARAM )
    {
        .....(視点操作パラメタを更新)
    }
    // 視点パラメタを更新(Scrollモード・媒介変数)
    if ( mode == VIEW_SCROLL_PARAM )
    {
        .....(視点操作パラメタを更新)
    }
    .....

    // 変換行列を更新(Walkthroughモード・直接更新)
    if ( mode == VIEW_WALKTHROUGH_DIRECT )
    {
        .....(変換行列を更新)
    }
};
```



サンプルプログラム(7)

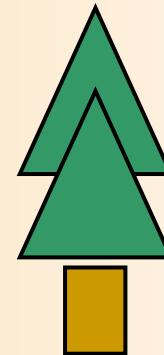
- 視点パラメタに応じて変換行列を更新
 - 決まった処理なので、変更の必要はない

```
void UpdateViewMatrix()
{
    // 視点パラメタを使った操作時のみ変換行列を更新
    if ( ( mode == VIEW_DOLLY_PARAM ) ||
        ( mode == VIEW_SCROLL_PARAM ) ||
        ( mode == VIEW_WALKTHROUGH_PARAM ) )
    {
        glMatrixMode( GL_MODELVIEW );
        glLoadIdentity();
        glTranslatef( 0.0, 0.0, -view_distance );
        glRotatef( -view_pitch, 1.0, 0.0, 0.0 );
        glRotatef( -view_yaw, 0.0, 1.0, 0.0 );
        glTranslatef( -view_center_x, -view_center_y, -view_center_z );
    }
}
```



シーン描画

- 視点の変化が分かりやすいように、ランダムに配置した木を描画
- OpenGL の円柱を描画する関数を使うことで、木のような物体を簡単に描画できる
 - gluCylinder(quad, 上の半径, 下の半径, 長さ, 横方向分割数, 縦方向分割数)
 - あらかじめ gluNewQuadric() 関数を使って、二次曲面情報 (quad) を作成する必要がある
 - 片方の半径を 0 にすると円すいになる
 - 3つの円柱+円すいとして描画
- 詳細はプログラムを参照





視点操作方法1 (Dolly Mode)

視点操作の実現方法

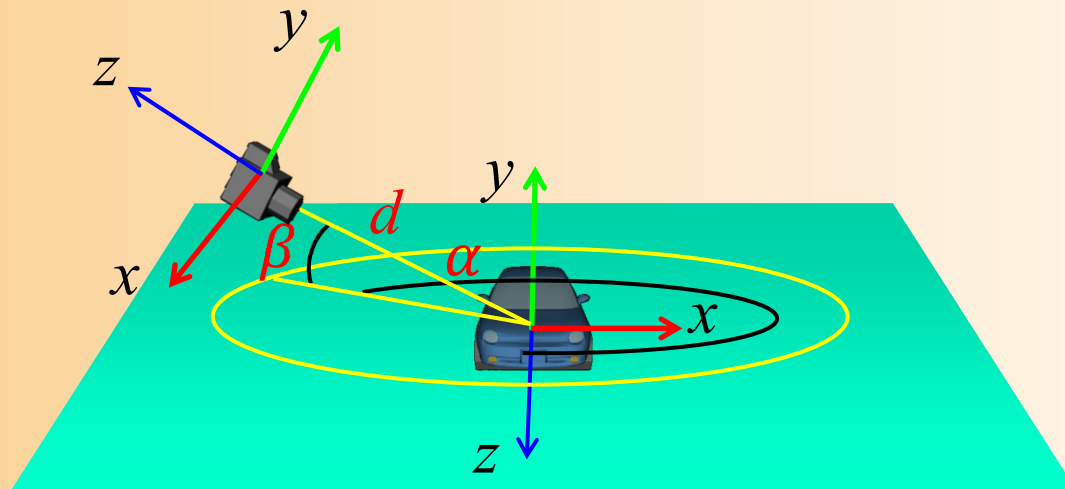
	媒介変数	直接更新
方法1: Dolly	サンプル	サンプル
方法2: Scroll	レポート課題	レポート課題
方法3: Walkthrough	レポート課題	レポート課題

視野変換行列

- Dolly Mode での視野変換行列

- 視点操作パラメタ

- 視点の方位角 (view_yaw) α
- 視点の仰角 (view_pitch) β
- 視点と注視点の距離 (view_distance) d




視野変換行列

• Dolly Mode での視野変換行列

– 視点操作パラメタ


- 視点の方位角 (view_yaw) α
- 視点の仰角 (view_pitch) β
- 視点と注視点の距離 (view_distance) d


$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\beta) & -\sin(-\beta) & 0 \\ 0 & \sin(-\beta) & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(-\alpha) & 0 & \sin(-\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\alpha) & 0 & \cos(-\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Y軸周りの回転の後に、X軸周りの回転をかける
- 2つの回転の後に、Z軸方向の平行移動をかける

媒介変数による変換行列の設定

- 媒介変数を利用した変換行列の設定
 - Dolly Mode は、こちらの方が簡単
 - マウス操作に応じて、視点操作パラメタを更新
 - 視点の方位角 (view_yaw) α
 - 視点の仰角 (view_pitch) β
 - 視点と注視点の距離 (view_distance) d
 - 視点操作パラメタにもとづき、変換行列を設定


$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\beta) & -\sin(-\beta) & 0 \\ 0 & \sin(-\beta) & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(-\alpha) & 0 & \sin(-\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\alpha) & 0 & \cos(-\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

プログラム(1)

```
void UpdateView( int delta_mouse_right_x, int delta_mouse_right_y,
                 int delta_mouse_left_x, int delta_mouse_left_y )
{
    // 視点パラメタを更新(Dorryモード・媒介変数)
    if ( mode == VIEW_DOLLY_PARAM )
    {
        // 横方向の右ボタンドラッグに応じて、視点を水平方向に回転
        if ( delta_mouse_right_x != 0 )
        {
            ..... ①
        }
        // 縦方向の右ボタンドラッグに応じて、視点を上下方向に回転
        if ( delta_mouse_right_y != 0 )
        {
            ..... ②
        }
        // 縦方向の左ボタンドラッグに応じて、視点と注視点の距離を変更
        if ( delta_mouse_left_y != 0 )
        {
            ..... ③
        }
    }
    .....
};
```



プログラム(2)

```
// 横方向の右ボタンドラッグに応じて、視点を水平方向に回転
```

```
if ( delta_mouse_right_x != 0 )  
{  
    view_yaw -= delta_mouse_right_x * 1.0;  
  
    // パラメタの値が所定の範囲を超えないように修正  
    if ( view_yaw < 0.0 )  
        view_yaw += 360.0;  
    else if ( view_yaw > 360.0 )  
        view_yaw -= 360.0;  
}
```

水平方向の角度の範囲に関しては、0~360 の範囲で連続するよう
に修正する
例えば、角度変更の結果、値が
-15 になったときには、0~360 の
範囲で同じ向きを表す 345 になる
ようにする

```
// 縦方向の右ボタンドラッグに応じて、視点を上下方向に回転
```

```
if ( delta_mouse_right_y != 0 )  
{  
    view_pitch -= delta_mouse_right_y * 1.0;  
  
    // パラメタの値が所定の範囲を超えないように修正  
    if ( view_pitch < -90.0 )  
        view_pitch = -90.0;  
    else if ( view_pitch > -2.0 )  
        view_pitch = -2.0;  
}
```

上下方向の角度の範囲に関しては、-90 ~ -2 の範囲で止まるよう
に修正する



プログラム(3)

```
// 縦方向の左ボタンドラッグに応じて、視点と注視点の距離を変更
if ( delta_mouse_left_y != 0 )
{
    view_distance += delta_mouse_left_y * 0.2;

    // パラメタの値が所定の範囲を超えないように修正
    if ( view_distance < 5.0 )
        view_distance = 5.0;
}
```



視点操作の実現方法

	媒介変数	直接更新
方法1: Dolly	サンプル	サンプル
方法2: Scroll	レポート課題	レポート課題
方法3: Walkthrough	レポート課題	レポート課題

変換行列を直接更新する方法(1)

- 変換行列を直接更新する方法
 - 起動時に初期状態で変換行列を初期化
 - マウス操作に応じて、適切な変換行列をかける

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\beta) & -\sin(-\beta) & 0 \\ 0 & \sin(-\beta) & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(-\alpha) & 0 & \sin(-\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\alpha) & 0 & \cos(-\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



$$\begin{pmatrix} R_{00} & R_{01} & R_{02} & T_x \\ R_{10} & R_{11} & R_{12} & T_y \\ R_{20} & R_{21} & R_{22} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



プログラム(1)

```
void UpdateView( int delta_mouse_right_x, int delta_mouse_right_y,
                 int delta_mouse_left_x, int delta_mouse_left_y )
{
    .....
    // 変換行列を更新(Dollyモード・直接更新)
    if ( mode == VIEW_DOLLY_DIRECT )
    {
        // 横方向の右ボタンドラッグに応じて、視点を水平方向に回転
        if ( delta_mouse_right_x != 0 )
        {
            .....  $\Delta\alpha$  を適用
        }
        // 縦方向の右ボタンドラッグに応じて、視点を上下方向に回転
        if ( delta_mouse_right_y != 0 )
        {
            .....  $\Delta B$  を適用
        }
        // 縦方向の左ボタンドラッグに応じて、視点と注視点の距離を変更
        if ( delta_mouse_left_y != 0 )
        {
            .....  $\Delta d$  を適用
        }
    }
    .....
}
```



プログラム(2)

- 視点を水平方向に回転($\Delta\alpha$)

```
// 横方向の右ボタンドラッグに応じて、視点を水平方向に回転
if ( delta_mouse_right_x != 0 )
{
    // 視点の水平方向の回転量を計算
    float delta_yaw = delta_mouse_right_x * 1.0;

    // 現在の変換行列の右側に、今回の回転変換をかける
    glMatrixMode( GL_MODELVIEW );
    glRotatef( delta_yaw, 0.0, 1.0, 0.0 );
}
```



変換行列を直接更新する方法(5)

- 視点と注視点の距離の変化 Δd
 - 現在の変換行列に左側から平行移動をかける
(①の位置に平行移動行列を挿入)
 - 現在の変換行列に左からかけることはできないため、
 - 現在の変換行列 M を記録した上で、単位行列に初期化
 - ①の位置に Δd の変換行列をかける
 - その右から、記録しておいた変換行列 M をかける

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\Delta d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{00} & R_{01} & R_{02} & T_x \\ R_{10} & R_{11} & R_{12} & T_y \\ R_{20} & R_{21} & R_{22} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

M



プログラム(3)

- 視点と注視点の距離を変更(Δd)

```
// 縦方向の左ボタンドラッグに応じて、視点と注視点の距離を変更
if ( delta_mouse_left_y )
{
    // 視点と注視点の距離の変化量を計算
    float delta_dist = delta_mouse_left_y * 1.0;

    // 現在の変換行列(カメラの向き)を取得
    float m[ 16 ];
    glGetFloatv( GL_MODELVIEW_MATRIX, m );

    // 変換行列を初期化して、カメラ移動分の
    // 平行移動行列を設定
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glTranslatef( 0.0, 0.0, - delta_dist );

    // 右からこれまでの変換行列をかける
    glMultMatrixf( m );
}
```


OpenGLに現在設定されている
変換行列を取得

(16次元の配列に4×4行列の
各要素が格納される)

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix}$$


変換行列を直接更新する方法(6)

- 視点の方位角の変化 $\Delta\beta$
 - やや複雑になる
 - ②の位置に回転行列を挿入


$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\beta) & -\sin(-\beta) & 0 \\ 0 & \sin(-\beta) & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(-\alpha) & 0 & \sin(-\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\alpha) & 0 & \cos(-\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

① ② ③ ④

プログラム(4)

• 視点を上下方向に回転($\Delta\beta$)

```
// 縦方向の右ボタンドラッグに応じて、視点を上下方向に回転  
if ( delta_mouse_right_y != 0 )
```

```
{
```

```
// 視点の上下方向の回転量を計算
```

```
float delta_pitch = delta_mouse_right_y * 1.0;
```

```
// 現在の変換行列を取得
```

```
float m[ 16 ];
```

```
glGetFloatv( GL_MODELVIEW_MATRIX, m );
```

```
// 現在の変換行列の平行移動成分を記録
```

```
float tx, ty, tz;
```

```
tx = m[ 12 ];  ty = m[ 13 ];  tz = m[ 14 ];
```

```
// 現在の変換行列の平行移動成分を0にする
```

```
m[ 12 ] = 0.0f;  m[ 13 ] = 0.0f;  m[ 14 ] = 0.0f;
```

```
....
```

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

現在の交換行列から、
左側の平行移動成分
を取得

現在の交換行列から、
左側の平行移動成分
をキャンセル



プログラム(5)

- 視点を上下方向に回転($\Delta\beta$) (続き)

```
// 変換行列を初期化
```

```
glMatrixMode( GL_MODELVIEW );
```

```
glLoadIdentity();
```

```
// カメラの平行移動行列を設定
```

①

```
glTranslatef( tx, ty, tz );
```

```
// 右側に、今回の回転変換をかける
```

②

```
glRotatef( delta_pitch, 1.0, 0.0, 0.0 );
```

```
// さらに、右側に、もとの変換行列から平行移動成分を取り除いたものにかける
```

③

```
glMultMatrixf( m );
```

```
}
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\Delta\beta) & -\sin(-\Delta\beta) & 0 \\ 0 & \sin(-\Delta\beta) & \cos(-\Delta\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{00} & R_{01} & R_{02} & 0 \\ R_{10} & R_{11} & R_{12} & 0 \\ R_{20} & R_{21} & R_{22} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

①

②

③



注意点1

- 変換行列を直接更新する方法の注意点
 - そのままでは、回転可能範囲や移動可能範囲などを制限するのは困難
 - これらの範囲の制限を加えるためには、変換行列から媒介変数を計算し、媒介変数を使って範囲を制限する必要がある



注意点2

- 変換行列を直接更新する方法の注意点

- 毎回、微少な回転行列をかけていくと、計算誤差の蓄積により、行列が歪んでくることがある
 - 回転成分の各ベクトルの長さが1、互いに直交している状態にならなくなる

$$\begin{pmatrix} R_{00} & R_{01} & R_{02} & T_x \\ R_{10} & R_{11} & R_{12} & T_y \\ R_{20} & R_{21} & R_{22} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- 定期的に回転成分の正規化が必要

- 長さ1に正規化、外積計算により直交ベクトルを計算



視点操作の実現方法

	媒介変数	直接更新
方法1: Dolly	サンプル	サンプル
方法2: Scroll	レポート課題	レポート課題
方法3: Walkthrough	レポート課題	レポート課題

レポートの視点操作の実装

- 方法2 (Scroll Mode) の変換行列
 - 媒介変数による方法の方が簡単
- 方法3 (Walkthrough Mode) の変換行列
 - 変換行列を直接更新する方法の方が簡単
- 以降の説明を参考に、これらの視点操作方法を実装して、レポート課題として提出すること





視点操作方法2 (Scroll Mode)

視点操作の実現方法

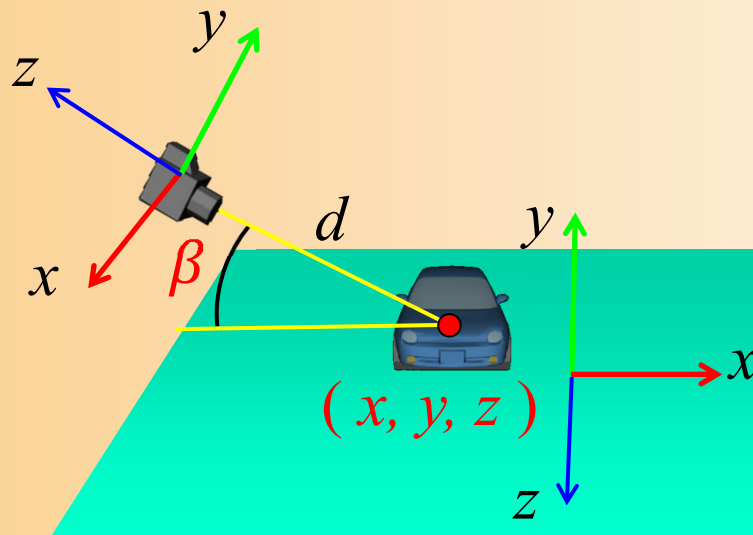
	媒介変数	直接更新
方法1: Dolly	サンプル	サンプル
方法2: Scroll	レポート課題	レポート課題
方法3: Walkthrough	レポート課題	レポート課題

視野変換行列

- Scroll Mode の視野変換行列

- 視点操作パラメタ

- 視点の仰角 (view_pitch) β
- 注視点の水平位置 (view_center_x|z) x, z
- 視点と注視点の距離 d , 注視点の垂直位置 y は固定値




視野変換行列

- Scroll Mode の視野変換行列

- 視点操作パラメタ

- 視点の仰角 (view_pitch) β
- 注視点の水平位置 (view_center_x|z) x, z
- 視点と注視点の距離 d , 注視点の垂直位置 y は固定値

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\beta) & -\sin(-\beta) & 0 \\ 0 & \sin(-\beta) & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



カメラ座標系での移動
(回転変換は影響しない)

ワールド座標系での移動
(回転変換が影響する)

媒介変数による方法

- 視点の仰角 β
 - 視点操作方法1 (Dolly Mode)と同様、上下方向の回転角度 β を変化させれば良い
 - 範囲の制限も適用する
- 注視点の水平位置 x, z
 - 方位角が変化しないと仮定すると(ワールド座標系とカメラ座標系の水平方向の向きが一致すると仮定すると)、簡単に実現できる
 - 横方向のマウス移動量に応じて x を、前後方向のマウス移動量に応じて z を変化させる
 - 方位角の変化も考慮する場合は、視点操作3を参照



プログラム

```
void UpdateView( int delta_mouse_right_x, int delta_mouse_right_y,
                 int delta_mouse_left_x, int delta_mouse_left_y )
{
    // 視点パラメタを更新(Scrollモード・媒介変数)
    if ( mode == VIEW_SCROLL_PARAM )
    {
        // 縦方向の右ボタンドラッグに応じて、視点を上下方向に回転
        if ( delta_mouse_right_y != 0 )
        {
            // 視点操作方法1(Dolly Mode)と同じ
        }

        // 左ボタンドラッグに応じて、視点を前後左右に移動(ワールド座標系を基準とした前後左右)
        if ( ( delta_mouse_left_x != 0 ) || ( delta_mouse_left_y != 0 ) )
        {
            view_center_x += ? ; // 左右方向の移動を加算
            view_center_z += ? ; // 前後方向の移動を加算
        }
    }
}
```

前の説明を参考に、どのような処理を記述すれば良いかを考える



視点操作の実現方法

	媒介変数	直接更新
方法1: Dolly	サンプル	サンプル
方法2: Scroll	レポート課題	レポート課題
方法3: Walkthrough	レポート課題	レポート課題

変換行列を直接更新する方法(2)

- 注視点の水平位置 $\Delta x, \Delta z$
 - 現在の変換行列に右側から平行移動をかける
(④)の位置に平行移動行列を挿入)

$$\begin{matrix} \textcircled{1} & & \textcircled{2} & & \textcircled{3} & & \textcircled{4} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\beta) & -\sin(-\beta) & 0 \\ 0 & \sin(-\beta) & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix}$$



$$\begin{pmatrix} R_{00} & R_{01} & R_{02} & T_x \\ R_{10} & R_{11} & R_{12} & T_y \\ R_{20} & R_{21} & R_{22} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

M

変換行列を直接更新する方法(3)

- 視点の仰角 $\Delta\beta$

- 右または左の平行移動成分をキャンセルして、回転行列をかける

- (②または③の位置に平行移動行列を挿入)

- 左の平行移動成分をキャンセルする方法(②の位置に平行移動行列を挿入)を用いる場合

- 今回は視点と注視点の距離 d は固定値であるため、左の平行移動成分を容易にキャンセルできる



$$\begin{matrix} \textcircled{1} & & \textcircled{2} & & \textcircled{3} & & \textcircled{4} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\beta) & -\sin(-\beta) & 0 \\ 0 & \sin(-\beta) & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix}$$

プログラム(1)

```
void UpdateView( int delta_mouse_right_x, int delta_mouse_right_y,
                 int delta_mouse_left_x, int delta_mouse_left_y )
{
    // 視点パラメタを更新(Scrollモード・直接更新)
    if ( mode == VIEW_SCROLL_DIRECT )
    {
        // 左ボタンドラッグに応じて、視点を前後左右に移動(ワールド座標系を基準として前後左右に移動)
        if ( ( delta_mouse_left_x != 0 ) || ( delta_mouse_left_y != 0 ) )
        {
            // 左右の移動量、前後の移動量を設定
            float dx = ?
            float dz = ?

            // 現在の変換行列の右側に、今回の平行移動をかける
            ?

            // 変換行列とは別に、注視点の位置を表すパラメタも更新する
            // (注視点の位置にオブジェクトを描画するため)
            view_center_x += dx;
            view_center_z += dz;
        }
    }
}
```

前の説明を参考に、どのような処理を記述すれば良いかを考える

変換行列の設定には媒介変数は用いないが、本プログラムでは、注視点位置を表す媒介変数を使って、注視点位置に赤い球を描画するため、媒介変数も更新する




プログラム(2)

```
// 縦方向の右ボタンドラッグに応じて、視点を上下方向に回転
if ( delta_mouse_right_y != 0 )
{
    // 視点の上下方向の回転量を計算
    float delta_pitch = delta_mouse_right_y * 1.0;

    // 現在の変換行列を取得
    float m[ 16 ];
    glGetFloatv( GL_MODELVIEW_MATRIX, m );

    // 変換行列を初期化
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();

    // 変換行列を更新
    
}
}
```

前の説明を参考に、どのような処理を記述すれば良いかを考える





視点操作方法3 (Walkthrough Mode)

視点操作の実現方法

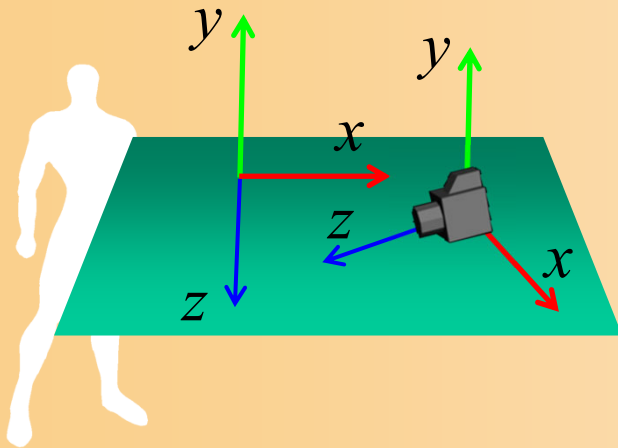
	媒介変数	直接更新
方法1: Dolly	サンプル	サンプル
方法2: Scroll	レポート課題	レポート課題
方法3: Walkthrough	レポート課題	レポート課題

視野変換行列

- Walkthrough Mode の変換行列

- 視点操作パラメタ

- 視点の方位角 (view_yaw) α
- 視点の水平位置 (view_center_x|z) x, z
- 視点の垂直位置 y は固定値



$$\begin{pmatrix} \cos(-\alpha) & 0 & \sin(-\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\alpha) & 0 & \cos(-\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

変換行列を直接更新する方法(1)

- 視点の方位角 $\Delta\alpha$

- 現在の変換行列に左側から回転をかける
(①の位置に回転行列を挿入)

$$\begin{matrix} \textcircled{1} & \begin{pmatrix} \cos(-\alpha) & 0 & \sin(-\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\alpha) & 0 & \cos(-\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} & \textcircled{2} & \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} & \textcircled{3} \end{matrix}$$

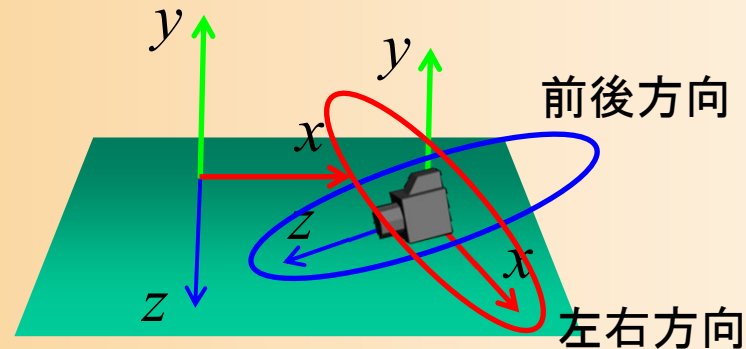
$$\begin{pmatrix} \cos(-\Delta\alpha) & 0 & \sin(-\Delta\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\Delta\alpha) & 0 & \cos(-\Delta\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{00} & R_{01} & R_{02} & T_x \\ R_{10} & R_{11} & R_{12} & T_y \\ R_{20} & R_{21} & R_{22} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

M



変換行列を直接更新する方法(2)


- 視点の水平位置 $\Delta x, \Delta z$
 - ワールド座標系ではなく、視点(カメラ座標系)から見て前後左右に移動する必要がある



変換行列を直接更新する方法(2)

- 視点の水平位置 $\Delta x, \Delta z$
 - ワールド座標系ではなく、視点(カメラ座標系)から見て前後左右に移動する必要がある
 - 現在の变换行列に左側から平行移動をかける
(①の位置に平行移動行列を挿入)

$$\begin{matrix} \textcircled{1} & \begin{pmatrix} \cos(-\alpha) & 0 & \sin(-\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\alpha) & 0 & \cos(-\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} & \textcircled{2} & \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} & \textcircled{3} \end{matrix}$$



カメラ座標系での移動
(回轉變換は影響しない)

$$\begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{00} & R_{01} & R_{02} & T_x \\ R_{10} & R_{11} & R_{12} & T_y \\ R_{20} & R_{21} & R_{22} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

プログラム(1)

```
// 変換行列を更新(Walkthroughモード・直接更新)
if ( mode == VIEW_WALKTHROUGH_DIRECT )
{
    // 横方向の右ボタンドラッグに応じて、視点を水平方向に回転
    if ( delta_mouse_right_x != 0 )
    {
        // 視点の水平方向の回転量を計算
        float delta_yaw = delta_mouse_right_x * 1.0;

        // 現在の変換行列(カメラの向き)を取得
        float m[ 16 ];
        glGetFloatv( GL_MODELVIEW_MATRIX, m );

        // 変換行列を初期化
        glMatrixMode( GL_MODELVIEW );
        glLoadIdentity();

        // 変換行列を更新
        

?


    }
}
```



プログラム(2)

```
// 左ボタンドラッグに応じて、視点を前後左右に移動(カメラの向きを基準として前後左右に移動)
if ( ( delta_mouse_left_x != 0 ) || ( delta_mouse_left_y != 0 ) )
{
    // 左右の移動量、前後の移動量を設定
    float dx = delta_mouse_left_x * 0.1f;
    float dz = delta_mouse_left_y * 0.1f;

    // 現在の変換行列(カメラの向き)を取得
    float m[ 16 ];
    glGetFloatv( GL_MODELVIEW_MATRIX, m );

    // 変換行列を初期化
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();

    // 変換行列を更新
    

?


}
}
```



視点操作の実現方法

	媒介変数	直接更新
方法1: Dolly	サンプル	サンプル
方法2: Scroll	レポート課題	レポート課題
方法3: Walkthrough	レポート課題	レポート課題

媒介変数による方法

- 視点の方位角 α
 - マウス操作に応じて、 α を変化させる
- 視点の水平位置 x, z
 - 視点(カメラ)から見て前後・左右に移動するように、 x, z を変化させる必要がある
 - ワールド座標系で見た、カメラ座標系の x, z 軸の方向ベクトルを求める必要がある
 - 現在の視野変換行列から取得することもできるし、
 - 三角関数を使って自分で計算することもできる



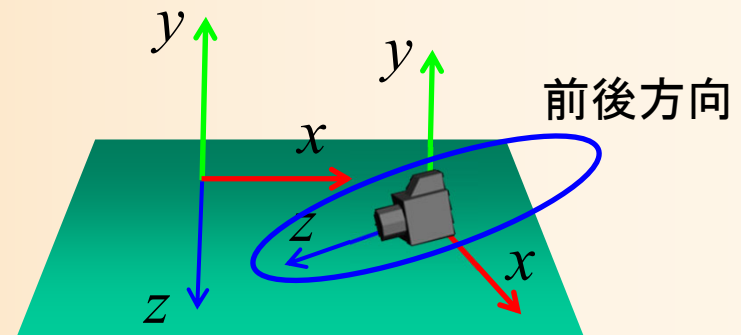
カメラの方向の求め方

- 現在の視野変換行列から取得する方法
 - 現在の視野変換行列を M とすると、ワールド座標系からみたカメラ座標系の Z 軸 (視点の前後方向のベクトル) は、 (Rz_x, Rz_y, Rz_z) になる
 - X 軸 (左右方向) も、同様の考え方で取得できる

$$M = \begin{pmatrix} Rx_x & Rx_y & Rx_z & x \\ Ry_x & Ry_y & Ry_z & y \\ Rz_x & Rz_y & Rz_z & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

ワールド座標系の Z 軸がカメラ座標系でどのようなベクトルになるか

カメラ座標系の Z 軸がワールド座標系でどのようなベクトルになるか



プログラム

```
// 視点パラメタを更新(Walkthroughモード・媒介変数)
if ( mode == VIEW_WALKTHROUGH_PARAM )
{
    // 横方向の右ボタンドラッグに応じて、視点を水平方向に回転
    if ( delta_mouse_right_x != 0 )
    {
        // 視点操作1(Dolly Mode)と同じ
    }
    // 左ボタンドラッグに応じて、視点を前後左右に移動(カメラの向きを基準とした前後左右)
    if ( ( delta_mouse_left_x != 0 ) || ( delta_mouse_left_y != 0 ) )
    {
        // 左右の移動量、前後の移動量を設定
        float dx = delta_mouse_left_x * 0.1;
        float dz = delta_mouse_left_y * 0.1;

        // 現在の変換行列(カメラの向き)を取得
        float m[ 16 ];
        glGetFloatv( GL_MODELVIEW_MATRIX, m );

        // ワールド座標系でのカメラの移動量を計算
        view_center_x += ? ;
        view_center_z += ? ;
    }
}
```

カメラの左右方向

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix}$$

カメラの前後方向





レポート課題

レポート課題

	媒介変数	直接更新
方法1: Dolly	サンプル	サンプル
方法2: Scroll	レポート課題	レポート課題
方法3: Walkthrough	レポート課題	レポート課題

レポート課題

- 視点操作方法2・方法3を実現するプログラムを作成せよ
 1. 視点操作方法2 (Scroll) ・媒介変数
 2. 視点操作方法3 (Walkthrough) ・直接更新
 3. 視点操作方法2 (Scroll) ・直接更新
 4. 視点操作方法3 (Walkthrough) ・媒介変数
 - サンプルプログラム (view_sample.cpp) をもとに作成したプログラムを提出
 - Moodleの本講義のコースから提出
 - 締切 : Moodleの提出ページを参照



レポート課題 提出方法

Moodleから、以下の2つのファイルを提出

- 作成したプログラム(テキスト形式)
 - view_sample.cpp
- 変更箇所のみを書き出したレポート(PDF)
 - 本講義のウェブサイトで公開している LaTeX のテンプレートをもとに、作成する
 - LaTeX の環境設定方法は、Moodleの説明を参照
 - LaTeX が使えない場合は、別のソフトウェアを使って作成しても構わないが、テンプレートと同様の様式・内容になるようにする



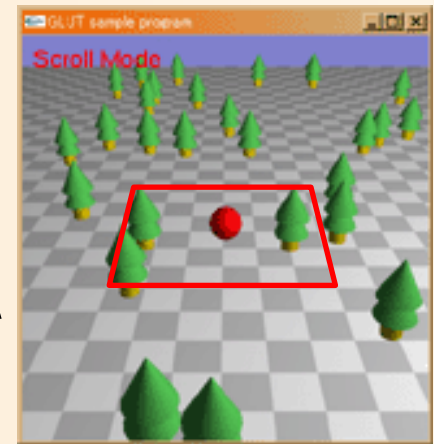
レポート課題 演習問題

- レポート課題の提出に加えて、レポート課題の理解度を確保するための Moodle 演習問題にも解答する
 - 解答締切は、レポート提出と同じ
 - 締切までは解答を変更可、締切後に正解が表示
 - レポート課題のヒントにもなっているので、レポート課題で分からない箇所があれば、演習問題の説明・選択肢を参考にして考えても良い
 - 本演習問題の点数は、演習課題の成績の一部として考慮する



より高度な視点制御

- 媒介変数の計算方法を工夫
- 例：注視点の細かい移動を防ぐ
 - 対象物が少し動くだけでも、視点が追従して動くため、見にくい
→ 対象物の位置変化が一定範囲を超えたときにのみ視点位置を変更



- 例：視点の角度を自動調節
 - 視点を注視点から離すと、全体を俯瞰して見られるように、自動的に視点の仰角を大きくする

まとめ

- 前回のサンプルプログラムの視点処理(復習)
- 視点操作のプログラム
- 視点操作方法1 (Dolly Mode)
 - 媒介変数を使う方法
 - 変換行列を直接更新する方法
- 視点操作方法2 (Scroll Mode)
- 視点操作方法3 (Walkthrough Mode)
- レポート課題



次回予告

- 幾何形状データの読み込みと管理
 - 幾何形状データ
 - ファイル形式
 - データ構造と描画処理
 - ファイル読み込み処理の作成
 - Cによる実装
 - C++による実装
 - 頂点配列の利用

