# 計算機序論2
2009/11/02
亀田能成

- 課題CはWWW参照
- Ubuntuのスクリーン設定に関して指示

## Ubuntu上での設定

OpenGLのサンプルプログラムを正常に表示させるために必要な設定

- スクリーンの何もないところで右クリック
  - メニュー→「背景の変更」
    - 「外観の設定」
      - 視覚効果（パネル）で「効果なし」(N)を選択

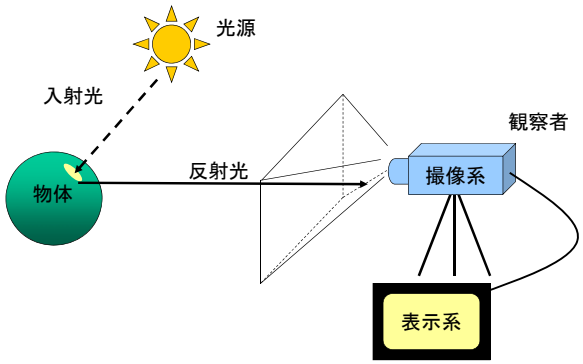


## レンダリング基礎

光源･物体･観測者･･三位一体

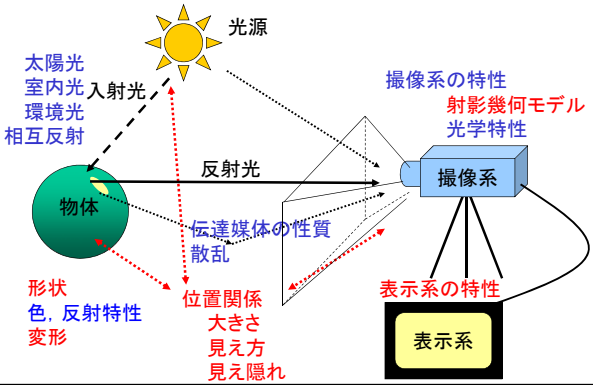


## 画像撮影



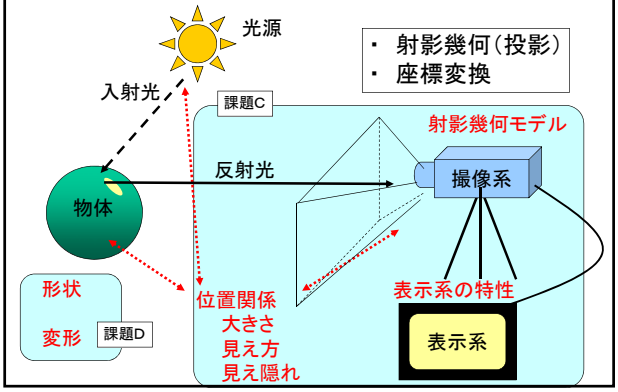



## CGで考慮すべき要因

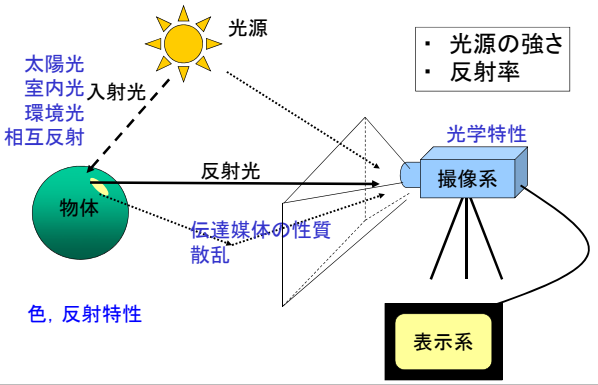





## 幾何的要因



# 光学的要因（課題Eに向けて）





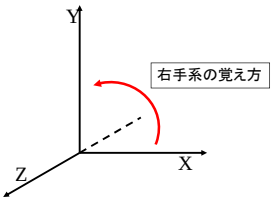

# 図形の表現

物体座標系・世界座標系・カメラ座標系
同次（斉次）座標表現

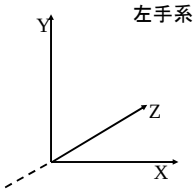


# 3次元空間の座標系

- 右手系と左手系の違い




右手系：OpenGLはこちら



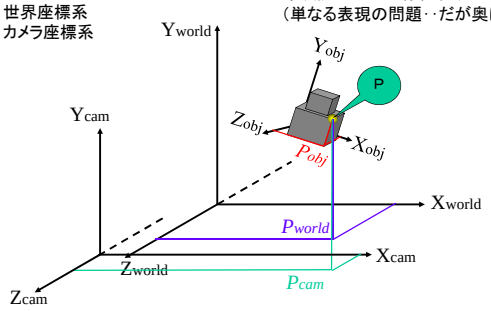

Zの向きが
（左手系と右手系は鏡像関係）



# 座標系

- 物体座標系
- 世界座標系
- カメラ座標系

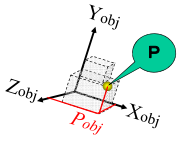
ある点Pがどこに存在するか？
（単なる表現の問題‥だが奥は深い）






# 物体座標系

物体を記述するための座標系

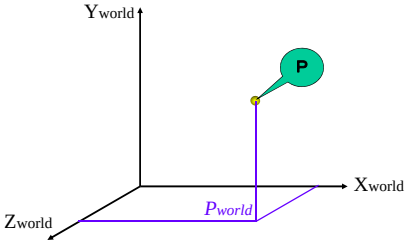



**Pobj = (Xobj, Yobj, Zobj)**



# 世界座標系

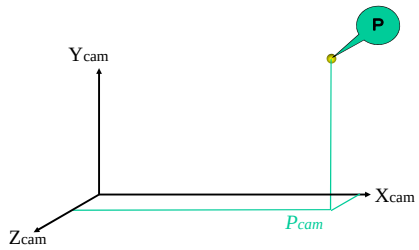
世界全体を記述するための座標系




**Pworld = (Xworld, Yworld, Zworld)**

# カメラ座標系

撮影カメラを中心とした座標系




**Pcam = (Xcam, Ycam, Zcam)**



# 同次(斉次)座標表現

- CG高速レンダリング(幾何学計算)を可能にするアイデア

通常の3次元位置の表現
3要素によるベクトル

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

同次座標表現
4要素目を追加した4次元ベクトル

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

当然4つ目に意味はない‥
が計算のトリックに多大な貢献
⇒あとで出てきます



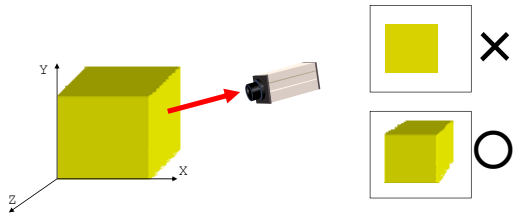
# 射影幾何

3次元空間中の物体を
2次元で表現する方法



# 平面上で3次元物体をどうやって描く？

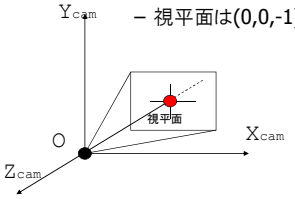
- 描画する(見る)のは平面上※
  - 3次元空間から2次元平面への投影（幾何変換）
  - 次元が一つ減るだけではなく、見え方の計算が必要
    （見え方にはいろいろ考えられる！






# OpenGLでの撮影カメラの設定

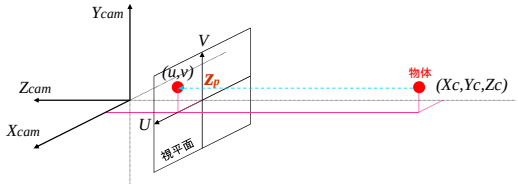
- デフォルト設定
  - カメラ視点(焦点)は原点に設置
  - カメラの向き(光軸)は(0,0,-1)
  - 画面上では、X軸正方向が右、Y軸正方向が上
- 本演習のプログラムでの追加設定
  - 視平面は(0,0,-1)を通りZ軸に直交






# 平行投影(直行変換)

- ある点を光軸と平行に視平面に投影する.




- 物体からの光は視平面にそれぞれ平行に到着.
- 近くの物体も遠くの物体も同じ大きさに見える.
- 視点の概念は必要ない（視線は必要）.
- 視点と物体の距離が十分大きい(物体の奥行きが，物体への距離に比べて十分小さい)場合に適する(ただしスケーリング必須).
- 製品の分解図等には重宝．日本の古来の画法.

# 平行投影の行列演算

視平面の奥行きが$Z_p$の場合

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&Z_p\\0&0&0&1\end{bmatrix}\begin{bmatrix}X_c\\Y_c\\Z_c\\1\end{bmatrix}=\begin{bmatrix}X_c\\Y_c\\Z_p\\1\end{bmatrix}$$

X項、Y項だけを残す

$$\begin{bmatrix}s&0&0&0\\0&s&0&0\\0&0&0&Z_p\\0&0&0&1\end{bmatrix}\begin{bmatrix}X_c\\Y_c\\Z_c\\1\end{bmatrix}=\begin{bmatrix}sX\\sY\\Z_p\\1\end{bmatrix}=\begin{bmatrix}u\\v\\Z_p\\1\end{bmatrix}$$

大抵の場合、画像平面と大きさが合わないのでXc,Ycについて S倍のスケーリング



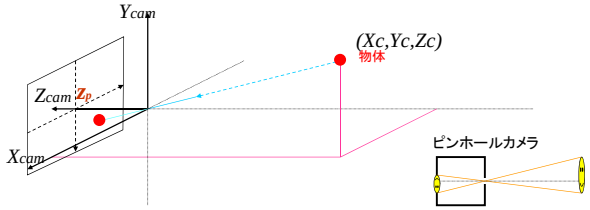

# 平行投影(直行変換)・余談

- 日本絵画の例(洛中洛外図)
  - 平行な直線の組は画像上で常に平行
  - 消失点がない
  - スケール感を雲とかで誤魔化す



# 透視投影(射影変換)

- 一点(焦点)を通過した光線のみを視平面に投影する.

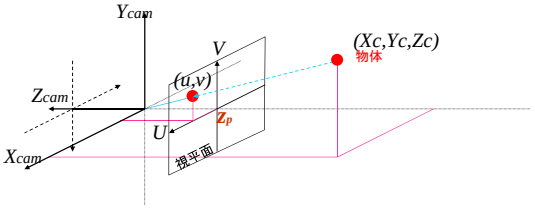



- 視点位置によって見え方が異なる
- 物体との距離が任意の場合に適する.
- 像の上下が反転する.



# 透視投影(射影変換)

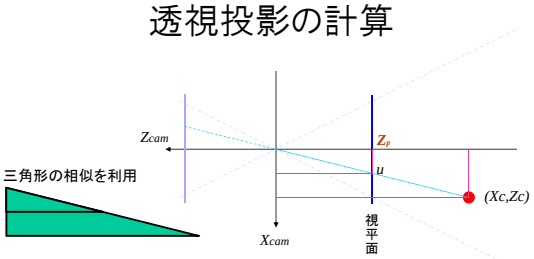
- 一点(焦点)を通過した光線のみを視平面に投影する.




- 視平面を物体側(マイナス側)に設定することにより 上下反転の問題を解消



# 透視投影の計算




求めたいのは $u$
与えられているのは(Xc,Zc)と $Z_p$(焦点距離)

$$\frac{X_c}{Z_c}=\frac{u}{Z_p}\Rightarrow u=\frac{Z_p}{Z_c}X_c=Z_p\frac{X_c}{Z_c}$$



# 透視投影の行列演算

$$\begin{bmatrix}\frac{Z_p}{Z_c}&0&0&0\\0&\frac{Z_p}{Z_c}&0&0\\0&0&\frac{Z_p}{Z_c}&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}X_c\\Y_c\\Z_c\\1\end{bmatrix}=\begin{bmatrix}\frac{Z_p}{Z_c}X_c\\\frac{Z_p}{Z_c}Y_c\\Z_p\\1\end{bmatrix}$$

行列内に $Z_c$ が混じる
⇒点ごとに異なる行列が必要

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&1/Z_p&0\end{bmatrix}\begin{bmatrix}X_c\\Y_c\\Z_c\\1\end{bmatrix}=\begin{bmatrix}X_c\\Y_c\\Z_c\\Z_c/Z_p\end{bmatrix}\Rightarrow\frac{Z_p}{Z_c}\begin{bmatrix}X_c\\Y_c\\Z_c\\Z_c/Z_p\end{bmatrix}=\begin{bmatrix}\frac{Z_p}{Z_c}X_c\\\frac{Z_p}{Z_c}Y_c\\Z_p\\1\end{bmatrix}=\begin{bmatrix}u\\v\\Z_p\\1\end{bmatrix}$$

同次項を利用してZcを外す(右辺の大きさは変わる)

得られた同次表現の第4項が1になるように除算!

# 透視投影(射影変換)・余談

- ルネッサンス期の西洋絵画(最後の晩餐)
  - 平行な直線群はどこか一点(消失点)で交わる
    (画面に平行な直線群だけは例外として画面上でも並行)
  - 消失点を前提に描かれた絵画として有名




# 物体の運動

座標系の変換で考える
斉次(同次)座標系と蓄積行列



# 平行移動

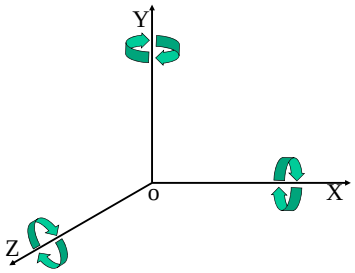
- 同次座標表現を用いれば、行列の積和計算だけで可能

$$\begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} X+tx \\ Y+ty \\ Z+tz \\ 1 \end{bmatrix}$$



# 回転の表現

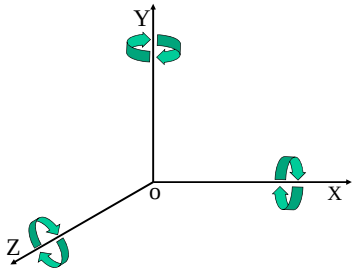
- 本講義では直交軸まわりの回転のみを考える.






# 直交軸回転

- 直交軸回転には3種類ある.(x軸回り, y軸回り, z軸回り)
- 順番が違うと結果が違う(行列演算では交換則は成立しない).

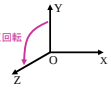





# 直交軸回転の行列表現

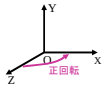
正回転は軸に対して右ネジの法則

- X軸回りの回転.



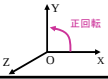

$$R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Y軸回りの回転.




$$R_y(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Z軸回りの回転.




$$R_z(\theta_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

座標変換

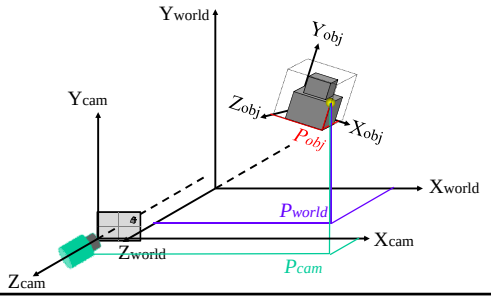
# 座標変換

OpenGLにおける画像提示の流れ
物体座標系→世界座標系→カメラ座標系

座標変換

# 座標系

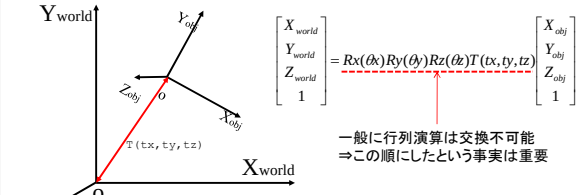
- 物体座標系(obj)　　物体表現を簡単に
- 世界座標系(world)　　物体とカメラが載る空間
- カメラ座標系(cam)　　写像計算を簡単に




座標変換

# 座標系の間の射影変換

- 座標系の間の射影変換も行列計算
  - 原点を動かす　:平行運動
  - 回す　:回転運動




$$\begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \\ 1 \end{bmatrix} = Rx(\theta x)Ry(\theta y)Rz(\theta z)T(tx,ty,tz) \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

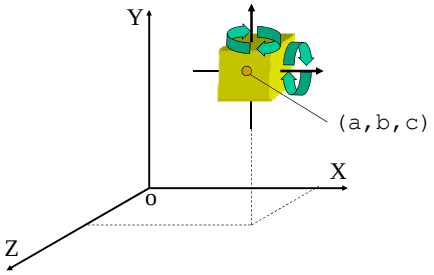
一般に行列演算は交換不可能
⇒この順にしたという事実は重要

調べてみよう(ただしあとの行列の蓄積も学習してから):
本演習のスクリプトのコマンド「アニメーション(A/C)」で
複数の運動が1行内で指定された場合はどの順で実行されるか？

物体の運動

# 物体の複合運動に対応する行列計算

例:(a,b,c)を中心にX軸回りα、Y軸回りβ回転
‥実はこれでは表現があいまい(不十分)




# 物体運動と座標系の計算

行列に掛けるベクトル「の座標系の上で」
- 回転行列
- 移動行列

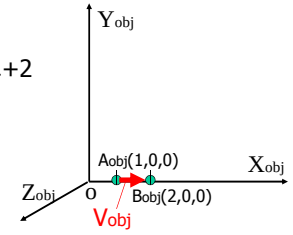
の演算は行われる。以後:
　解釈①　すべてを世界座標系の値で
　解釈②　座標系をどんどん乗り換えて
演算する例を示す(結果は同じ！)

座標変換

# 物体運動と座標変換の行列計算

物体の表現で用意したもの:ここではベクトル1本だけ
物体座標系:点$A_{obj}(1,0,0)$から点$B_{obj}(2,0,0)$に向かうベクトル$V_{obj}$

スタート:物体座標系と世界座標系が一致
1. X軸に方向に+4
2. その場で物体座標系の
   Y軸回りに $\alpha$ 回転
3. 物体座標系のX軸方向に+2

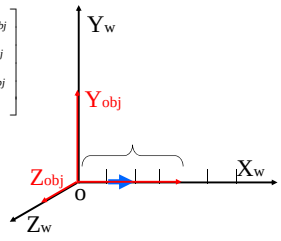


# 物体運動の行列計算

初期状態の考え方

- 物体座標系と世界座標系が一致

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix} = \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$





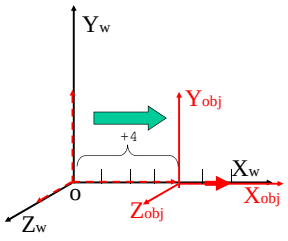

# 物体運動の行列計算①

「世界座標系の中で」

1. X軸に方向に+4

$$\begin{bmatrix} X_{w\_step1} \\ Y_{w\_step1} \\ Z_{w\_step1} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

$$= T(4,0,0)\begin{bmatrix} X_{obj} & Y_{obj} & Z_{obj} & 1 \end{bmatrix}^T$$





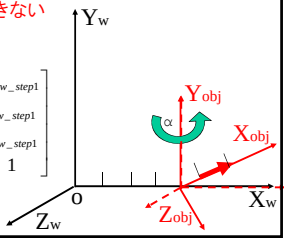

# 物体運動の行列計算①

「世界座標系での回転行列を使って」

2. その場で物体座標系の
   Y軸回りに $\alpha$ 回転

原点へ戻さないと軸回転は表現できない

$$\begin{bmatrix} X_{w\_step2} \\ Y_{w\_step2} \\ Z_{w\_step2} \\ 1 \end{bmatrix} = T(4,0,0)Ry(\alpha)T(-4,0,0) \begin{bmatrix} X_{w\_step1} \\ Y_{w\_step1} \\ Z_{w\_step1} \\ 1 \end{bmatrix}$$



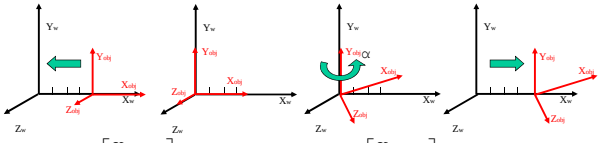



# 物体運動の行列計算①

「世界座標系での回転行列を使って」

2. その場で物体座標系のY軸回りに $\alpha$ 回転

原点へ戻さないと軸回転は表現できない




$$\begin{bmatrix} X_{w\_step2} \\ Y_{w\_step2} \\ Z_{w\_step2} \\ 1 \end{bmatrix} = T(4,0,0)Ry(\alpha)T(-4,0,0) \begin{bmatrix} X_{w\_step1} \\ Y_{w\_step1} \\ Z_{w\_step1} \\ 1 \end{bmatrix}$$



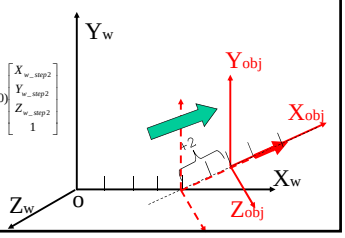
# 物体運動の行列計算①

「世界座標系の回転行列を使って」

3. 物体座標系のX軸方向に+2

原点へ戻して変換を施す

$$\begin{bmatrix} X_{w\_step3} \\ Y_{w\_step3} \\ Z_{w\_step3} \\ 1 \end{bmatrix} = T(4,0,0)Ry(\alpha)T(2,0,0)Ry(-\alpha)T(-4,0,0) \begin{bmatrix} X_{w\_step2} \\ Y_{w\_step2} \\ Z_{w\_step2} \\ 1 \end{bmatrix}$$



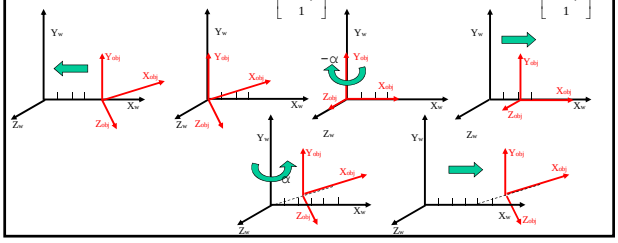



# 物体運動の行列計算①

3. 物体座標系のX軸方向に+2

原点へ戻して変換を施す

$$\begin{bmatrix} X_{w\_step3} \\ Y_{w\_step3} \\ Z_{w\_step3} \\ 1 \end{bmatrix} = T(4,0,0)Ry(\alpha)T(2,0,0)Ry(-\alpha)T(-4,0,0) \begin{bmatrix} X_{w\_step2} \\ Y_{w\_step2} \\ Z_{w\_step2} \\ 1 \end{bmatrix}$$



# 物体運動の行列計算①

全体としては：

これまでの演算結果を合成して変換を作成

$$\begin{bmatrix} X_{w\_step3} \\ Y_{w\_step3} \\ Z_{w\_step3} \\ 1 \end{bmatrix} = T(4,0,0)Ry(\alpha)T(2,0,0)Ry(-\alpha)T(-4,0,0) \begin{bmatrix} X_{w\_step2} \\ Y_{w\_step2} \\ Z_{w\_step2} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{w\_step2} \\ Y_{w\_step2} \\ Z_{w\_step2} \\ 1 \end{bmatrix} = T(4,0,0)Ry(\alpha)T(-4,0,0) \begin{bmatrix} X_{w\_step1} \\ Y_{w\_step1} \\ Z_{w\_step1} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{w\_step1} \\ Y_{w\_step1} \\ Z_{w\_step1} \\ 1 \end{bmatrix} = T(4,0,0) \begin{bmatrix} X_{obj'} \\ Y_{obj'} \\ Z_{obj'} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{obj'} \\ Y_{obj'} \\ Z_{obj'} \\ 1 \end{bmatrix} = I \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$



# 物体運動の行列計算①

全体としては：

これまでの演算結果を合成して変換を作成

$$\begin{bmatrix} X_{w\_step3} \\ Y_{w\_step3} \\ Z_{w\_step3} \\ 1 \end{bmatrix} = T(4,0,0)Ry(\alpha)T(2,0,0)Ry(-\alpha)T(-4,0,0) \begin{bmatrix} X_{w\_step2} \\ Y_{w\_step2} \\ Z_{w\_step2} \\ 1 \end{bmatrix}$$

$$T(4,0,0)Ry(\alpha)T(-4,0,0)$$

$$T(4,0,0)$$

$$I \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{w\_step3} \\ Y_{w\_step3} \\ Z_{w\_step3} \\ 1 \end{bmatrix} = T(4,0,0)Ry(\alpha)T(2,0,0)I \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix} = C \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

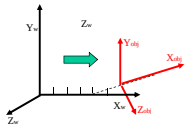


# 物体運動の行列計算②

別の解釈：

日本語の語順と同じ

$$\begin{bmatrix} X_{w\_step3} \\ Y_{w\_step3} \\ Z_{w\_step3} \\ 1 \end{bmatrix} = T(4,0,0)Ry(\alpha)T(2,0,0)I \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$




「X軸方向に+4移動させた先でY軸周りにα度回転させて」からX軸方向に+2移動させた「物体座標の情報」」」

世界座標系での測値

幾何的な意味を把握できるならこの覚え方でも可



# OpenGLでの行列の実装

- 幾何変換などの変換行列
  - １次元の配列を利用して4x4の行列を表現
  - 行列に対して右から縦ベクトルを掛ける(という約束)

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

$$\begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**配列に格納される順番に注意**

- `ic2_translate,ic2_rotateX,ic2_rotateY,ic2_rotateZ`の代入要素
- `ic2_multmatmat()`における要素参照の順番に注目



# 行列変換の蓄積



# 蓄積行列

- ic2_matrixoperation()　で実行
- 演習の行列演算は、以下のような手順で処理される.
  *k=basematrix（初期設定で与えられる行列:通常単位行列）*
  *k=k I*
  *k=k T(2,0,0)*
  *k=k Ry(a)*
  *k=k T(4,0,0)*
- 行列$k$ を蓄積行列と呼ぶ （演算処理がドンドン蓄積されている）

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = T(4,0,0)Ry(\alpha)T(2,0,0)I \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

# 蓄積演算

- 行列tを行列kに蓄積
  行列tと行列kの積算結果を行列kに代入する
  ic2_multmatmat(k, k, t);
- ic2_matrixoperation ()での注意事項
  - 行列計算は演算用のコピーで実行
  - 最終的にプログラムに戻す行列(ポインタ渡し)は？

アニメーション描画の場合、さらに蓄積の回数が増えるが、
それについては次回以降



# OpenGLにおける描画フロー

物体座標系→世界座標系→カメラ座標系

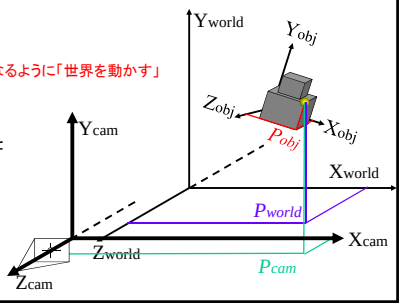


# 物体を撮影する手順

考えがちな手順
(1) 物体を世界の中に置く
(2) カメラを世界の中に置く
(3) 撮影する

CGの実際手順
(1) 物体を世界の中に置く
(2') カメラが世界の中心になるように「世界を動かす」
(3) 撮影する

これは(3)の撮影の計算が
自由なカメラ位置に対してだと
計算式が面倒になることと、
(2)と(2')の計算コストが
変わらないことから。




# 物体を撮影する行列演算

$$\begin{bmatrix} u \\ v \\ Z_p \\ 1 \end{bmatrix} = (3)(2')(1)(1) \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \\ 1 \end{bmatrix} = (1)(1) \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix} = (2') \begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ Z_p \\ 1 \end{bmatrix} = (3) \begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix}$$

「(1) 物体を世界の中に置く」は
多段になることがある(アニメーションの継続など)

# 物体を撮影する行列演算

コマンドA(アニメーション)のみが来た場合
(1)ic2_matrixoperation()関数 … mvm_work行列
(2')ic2_lookat()関数 … mvm行列
(3)gluPerspective()関数

$$\begin{bmatrix} u \\ v \\ Z_p \\ 1 \end{bmatrix} = (3)(2')(1) \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

mvm行列
mvm_work行列

OpenGLの関数gluPerspective()を使って
行列を暗示的に作成

合成した結果(mvm_work行列)を渡す

GL_PROJECTION行列
(2D←3D)

GL_MODELVIEW行列
(3D←3D)

# 物体を撮影する行列演算

コマンドA1/C2/C3(アニメーション)が連続して来た場合
(1)ic2_matrixoperation()関数
… mvm_work行列 と cumulatedmatrix行列

(2'),(3)は前と同じ

$$\begin{bmatrix} u \\ v \\ Z_p \\ 1 \end{bmatrix} = (3)(2')(1)_1(1)_2(1)_3 \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

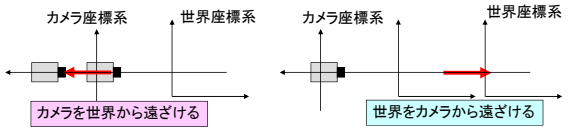
cumulatedmatrix
に保存

mvm_work
で操作

描画フロー

## (2)カメラを世界の中に置く
## (2')カメラが世界の中心になるように「世界を動かす」

- カメラの移動は、物体を逆向きに移動させることで表現
  - 実は、カメラの動いた分、世界が逆に"動かされて"いる
  - 世界座標系からカメラ座標系への変換
  - デフォルトでは、撮影カメラはカメラ座標系の中で原点に配置され、(0,0,-1)を向いている
- ic2_LookAt(float *mvm)
  - mvm: 動かされた量を返す(蓄積)
  - 大域変数 camdist でカメラを世界中心から遠ざけている
  - 他にも？？(隠しコマンド)




描画フロー

## (3)撮影する

- gluPerspective(double fovy, double aspect, double znear, double zfar)
  - fovy:垂直画角 *field of view in y direction*
  - aspect:縦横比(width/height)
  - znear:前面
  - zfar:後面