エルゴノミクスコンピューティングレポート 1

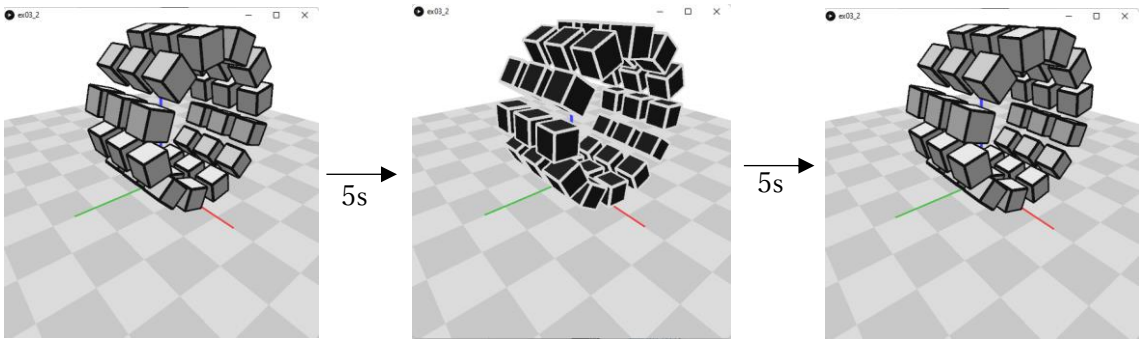
27019679 グレゴリウス　ブライアン

ex03_2

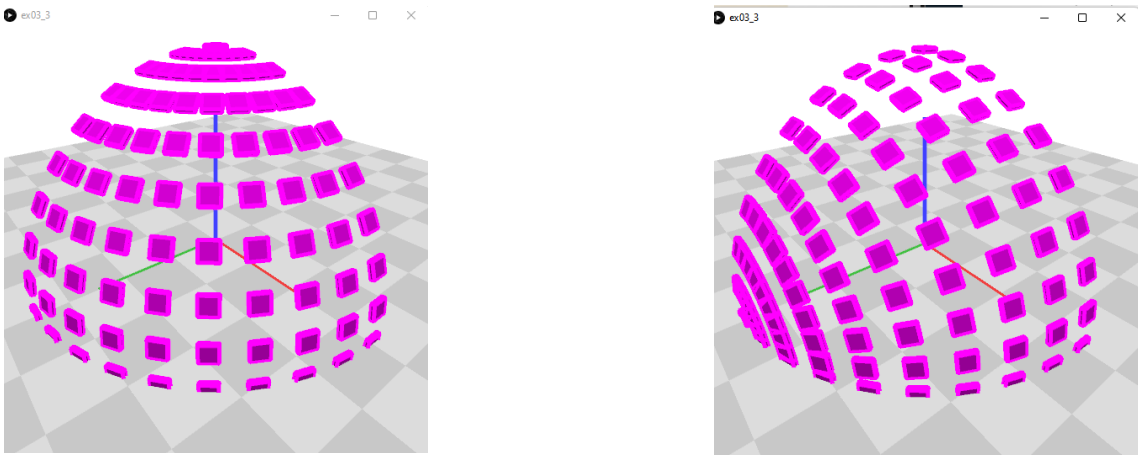
自分が書いたものはすべて関数 drawRing()にまとめた。一個の「輪」の描き方はまず座標軸を中心点に移動させてから、その時点の全体回転位相（全体の回転中の度合）分だけ回転させる。その後、12 分の 2πずつ回転し、立方体を描く。

工夫した点は二つある。

1. X 軸方向に 2 回クローンして並べた
2. 1 周期で色を徐々に変化させて戻した。具体的に、以下の色ループになるように設定した。

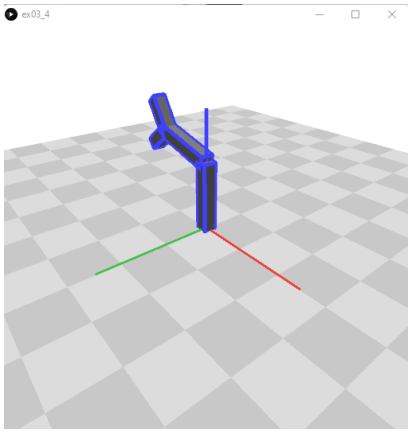


Ex03_3



初期座標系からループを i と j のループを走査する。i と j は 0,10,20,…,90 の角度を取る。ここで、x 軸と z 軸の角度をθとし、x 軸と y 軸の角度をϕとする。i と j をそれぞれラジアンに変換し、-i をθ、j をϕとし、関数 polarTransform にそれぞれ渡す。関数 polarTransform は Z 軸の周りϕに回転してから、Y 軸に周りにθ回転し、X 軸方向に r だけ並進させる。着いた原点で box(1,a,a)を描けば以上の左の結果になる。ここで、注意したいのは、回転の順番を変えたら上の右図になる。

Ex03_4



3 つの関数、drawHand(),drawArm(),drawFinger()を作った。drawArm()はアームの中心から box を描画後、さらにもう一回 translate(0,0,ARM_LENGTH/2)をする。これは次の関節の位置の相当し、次のパーツの原点として解釈できる。drawFinger()は単に drawArm()の半分の長さのみなので drawHand()について次に解説する。drawHand 自体は Z 軸周りにθ_1だけ回転し、drawArm()を呼び出す。次に X 軸周りにθ_2だけ回転し、drawArm()を再び呼び出す。最後に、一旦今の状況を pushMatrix()し、X 軸周りに$-\theta_3$だけ回転し、drawFinger()を呼び出して、popMatrix を行い、Y 軸周りθ_3だけ回転したあと再び drawFinger()を呼び出す。ここでは単純だが、$\theta_1, \theta_2, \theta_3$を先に決める必要がある。

まず$f(t,T)$の関数を以下のように定義する

$$f(t,T) = \frac{t}{T} 2\pi \bmod 2\pi$$

ただし t は経過時間(millis())であり、T は周期(PERIOD)である。mod 操作は単純にfが$[0,2\pi]$に収まるように制限されるためのもので、デバッグしやすい利点がある。実際のプログラム中挙動は単調増加関数と等しい。ここで、$\theta_1 = f(t)$が利用される。
次にθ_2, θ_3は周期性が必要であるため、$g(x)$の関数を以下のように定義する

$$g(x) = \frac{\sin x + 1}{2}$$

$g(x)$ は sin の形を保ったまま取りうる値を$[0,1]$に収めさせる関数である。θ_2, θ_3は次のように定義する。

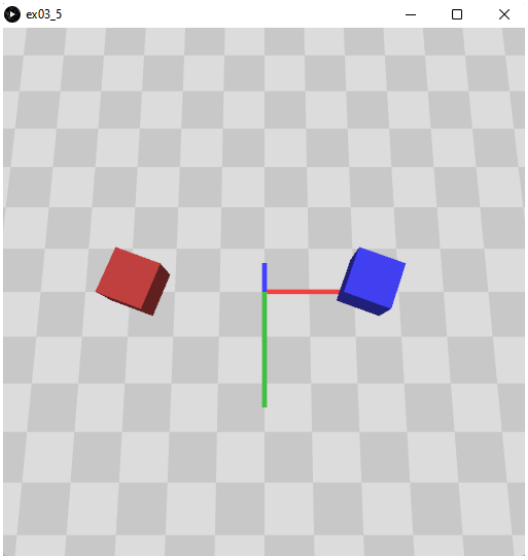
$$\theta_2 = g\big(f(t,T)\big) \cdot \frac{\pi}{2}$$

$$\theta_3 = g\big(f(t,T)\big) \cdot \frac{\pi}{4}$$

これで、θ_2, θ_3は両方 sin のなめらかな動きを保ちながらθ_2が$[0, \frac{\pi}{2}]$、θ_3が$[0, \frac{\pi}{4}]$の範囲に収まる。回転速度を変化させたいなら、Tの値を変えればいい。実際のプログラムでは$\theta_1, \theta_2, \theta_3$

のTはそれぞれ3000,2000,1000[ms]に設定されている。

Ex03_5



最終的に行列は以下となる。

$$\begin{pmatrix} c & -s & 0 & -sy \\ s & c & 0 & cy \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

左の赤い箱のコードの部分を見てみると、行われる変換は 2 つあり、一つは Z 軸周りに theta1 だけ回転してから、(0,y,0)方向に並進させる。$s = \sin\theta, c = \cos\theta$と置くと、Z 軸周りに θ だけ回転する変換行列は次通りである

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

次に、(0,y,0)に並進させるので現座標の行列に次の行列を右から乗じる

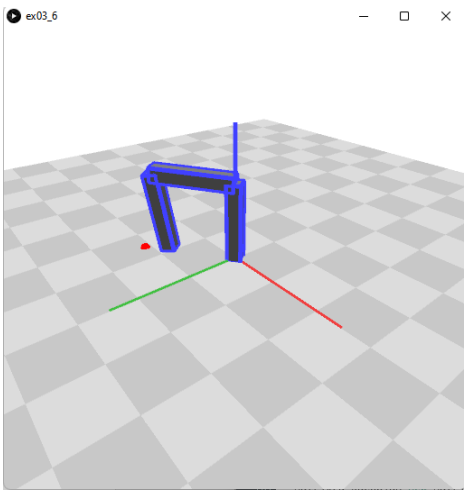
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

したがって、

$$\begin{pmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} c & -s & 0 & -sy \\ s & c & 0 & cy \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

になる。

Ex03_6



行列計算を簡潔にするために、Matrix2D という 2 次元行列クラスを定義した。このクラスのメッソドは以下の 3 つ：

1. print() – 行列をコンソールに出力する
2. inverse() – 今ある行列を逆行列にする、ただし、失敗した場合、false を返し、何も起こらず
3. mult_to_right(float[] v) – $\boldsymbol{v}$という 2 次元ベクトルを引数とし、自分の行列を$\boldsymbol{M}$とすると、$\boldsymbol{Mv}$（float[]型)を返す

ロボアームの 3 つの角度を保存する変数 theta(float[]型)をあらかじめ定義し、初期値を

{0,HALF_PI,HALF_PI}にする。また、今のターゲット（小さい赤ボール）の xyz 位置を target(float[]型)に格納する。z の値は今のところ常に 0 に設定する。

theta[0]の回転後のターゲットは atan(target[1]/target[0])で決める。つまり、XY 平面上の X 軸に対しての傾きをインバースタンジェントに入れたもの。しかし、これは-pi/2 から pi/2 の範囲、つまり、第 1 象限と第 4 象限のものにしか対応していない。これに対応するためには、第 2 象限か第 3 象限にあるかどうかをあらかじめ target[0]と target[1]の符号で調べ、第 2 象限にあたると、atan は第 4 象限に対応する角度を出すので、その値を+PI する。第 3 象限も同じく第 1 象限の角度を-PI することで求まる。これで、target の XY 平面上の x 軸に対しての角度が決まる。この値は[-PI,PI]に収まる(以下この値を target_theta0 と呼ぶ)。

target_theta0 と現在の theta[0]の差を取れば、回転すべき角度 dtheta0 が決まる。そしてその角度分だけ回転させるが、そのまま代入すれば 1 フレームでロボアームがワープのような感じで飛ぶだろう。そこで、回転速度を次の式で制限する：

$$今のフレームの回転 = sign(dtheta0) \cdot min(|dtheta0|, HALF\_PI/20);$$

ただし、sign 関数は引数が負の場合-1、正か 0 の場合 1 を返す関数であり、回転方向を表す。このフレームで実際にどれぐらい回転するのかは min の中からで決める。多くの場合、min の結果が第 2 引数になり、これは回転の制限速度を与える。

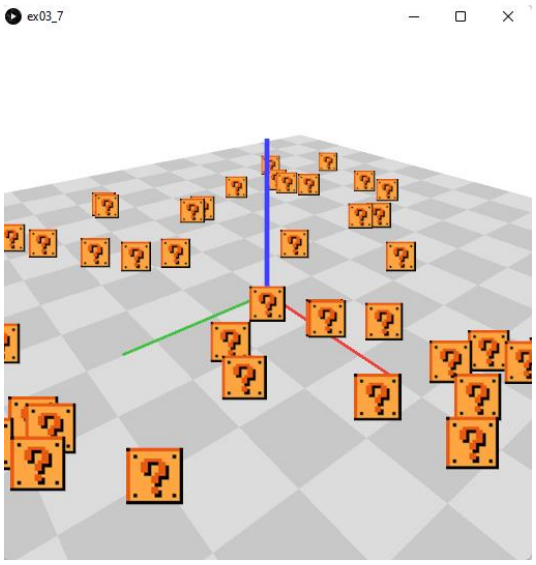
以上が theta[0]の回転角度決め方である。theta[1]と theta[2]の回転角度決め方はスライド通りで、Jacobian の逆行列を用いて計算する。回転角度を決めたあと、theta[0]の回転と同じように制限速度を与える。

ボールが当たるときに新しいボールを spawn させるが、ボールのあたっているかどうかの判定は回転すべき角度の絶対値がすべて一定以下の閾値(今は 0.01)であればあたっていると判定する。

実装上の注意点がいくつかある：

1. theta0 の回転は PI を超える回転ができないため、例えば 7PI/8 から-7PI/8 は正の方向の短い回転ができない、長い負の回転になる。
2. 今ボールの spawn 位置は-35 から 35 に設定しているが、z=0 制約条件の元でのアームの最大の r は計算していない。r 範囲外になる可能性があるため、そうなったら再起動するか spawn 位置の制限を厳しくしてください。

Ex03_7



資料と別考え方で実装した。また、ビルボードの位置自体は使っていない。カメラの注視点の位置から視点の位置ベクトルを $\mathbf{c}$ と置く。今回は注視点の位置はワールド原点であるので

$$\boldsymbol{c} = (EYE\_X \quad EYE\_Y \quad EYE\_Z)^{\mathrm{T}}$$

である。プログラムを修正せずに実行するとオブジェクトはx軸の正の方向に描画される。そのため、このオブジェクトをビルボードにするためには、x軸の正の方向を$\boldsymbol{c}$と同じ方向を向くようにする。そのようになるためには、座標軸を 2 回回転させる必要がある。第 1 回転は z軸周りの回転で第 2 回転は y軸周り回転とする。

まず、第 1 回転の角度θ_1を求める。z軸周りの回転なので、回転後のベクトルが XY 平面上にある。回転角度自体は$(1\ 0)^{\mathrm{T}}$を$(EYE_X \quad EYE_Y)^{\mathrm{T}}$に射影する角度である。この角度は内積の定義から求めることができるが、原点と(EYE_X, EYE_Y)を通る直線とx軸の角度を見れば、

$$\theta_1 = \arctan\left(\frac{EYE\_Y}{EYE\_X}\right)$$

であることがわかる。回転後(EYE_X, EYE_Y, EYE_Z)の座標は

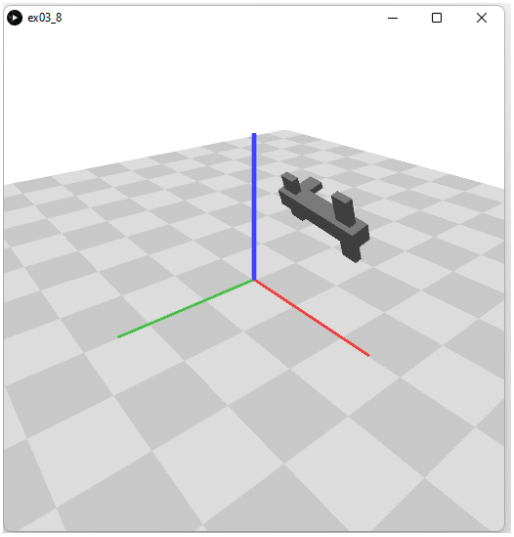
$$\left(\sqrt{EYE\_X^2 + EYE\_Y^2}\,, \quad 0, \quad EYE\_Z\right)$$

である。同じ原理で XZ 平面上の回転を考えると

$$\theta_2 = \arctan\left(\frac{EYE\_Z}{\sqrt{EYE\_X^2 + EYE\_Y^2}}\right)$$

となる。回転の向きに注意して最終的に、座標系 rotateZ(θ_1)したあとに rotateZ$(-\theta_2)$をすればビルボードが作れるのである。資料の applyMatrix()に直接かける方法と違うが、実際この 2 回の回転を行列の掛け算で表現することもできる。

Ex03_8



PMatrix3D クラスを用いて作成した。特にこだわった工夫はない。Apply メソッドを使って乗算を行った。119 行のコメントアウトされた部分は飛行機を自動的に前進する仕組みになっている。