WIDE Technical-Report in 2007

地球自転の立体視用動画作成

wide-tr-netviz-rotatingearth-00.pdf

WIDE Project : http://www.wide.ad.jp/

If you have any comments on this document, please contact to ad@wide.ad.jp

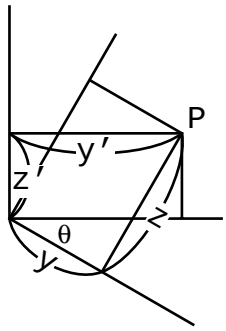
地球自転の立体視用動画作成

2006-12-19　和田英一 (IIJ)

目的

地球上をインターネットのパケットが飛び交う様子を表示したいと思う人は少なくない. それに日本科学未来館の Geo Cosmos が使えないか, 国立天文台の 4D2U プロジェクトの装置が使えないかなど考えたが, とりあえずは赤青の眼鏡を使った立体視で感触をつかむことにした.
赤青の眼鏡による立体視は anaglyph という. 通常は赤と青といっているが, 補色の赤 (Red rgb=1,0,0) とシアン (Cyan rgb=0,1,1) を使う. 同じ地球を多少ずれた二つの視点から見たような絵をこの 2 色で書き, 眼鏡で見ると立体に見える. そういう絵を自転にあわせてたくさん用意し (ここでは 3 度おきに 120 組), それを順次表示すると立体の地球が自転しているように見える. その地球の上にインターネットの通路を描こうというものである.
以下, 地球の透視図を書く; それに地図を投影するという順に記述する

座標変換



図のようにある点 P(y, z) を時計回りに θ だけ回転し, P'(y', z') に来たとする.
$y' = y\cos\theta + z\sin\theta$
$z' = z\cos\theta - y\sin\theta$
これは 2 次元だが, 3 次元で考える. transxyz2 は P(x, y, z) を x 軸に対して α, y 軸に対して β, z 軸に対して γ だけこの順に回転したときの移動先 P'(x', y', z') の座標を計算する

$y' = y\cos\alpha + z\sin\alpha$　　$x' = x\cos\beta - z'\sin\beta$
$z' = z\cos\alpha - y\sin\alpha$　　$x'' = x'\cos\gamma + y'\sin\gamma$
$z'' = z'\cos\beta + x\sin\beta$　　$y'' = y'\cos\gamma - x'\sin\gamma$

$a = \cos\alpha, b = \sin\alpha, c = \cos\beta, d = \sin\beta, e = \cos\gamma, f = \sin\gamma$ として計算し, ダブルプライム $''$ をプライム $'$ にすると,
$x' = ce\,x + (bde + af)y - (ade - bf)z$
$y' = -cf\,x + (ae - bdf)y + (adf + be)z$
$z' = d\,x - bc\,y + ac\,z$
となる. それを PostScript で計算する.

```
/transxyz2{12 dict begin /z exch def /y exch def /x exch def
/alpha -45 def /beta -15 def /gamma -30 def
/a alpha cos def /b alpha sin def /c beta cos def /d beta sin def
/e gamma cos def /f gamma sin def
e c mul x mul e d mul b mul f a mul add y mul add
e d mul a mul f b mul sub z mul sub
f c mul x mul neg e a mul f d mul b mul sub y mul add
e b mul f d mul a mul add z mul add
d x mul c a mul z mul add c b mul y mul sub end} def
%x'=ecx+(edb+fa)y-(eda-fb)z
%y'=-fcx+(ea-fdb)y+(eb+fda)z
%z'=dx-cby+caz
```

これは後にある

```
/xturn {4 dict begin /th exch def /z exch def /y exch def /x exch def
x y th cos mul z th sin mul add z th cos mul y th sin mul sub end} def
/yturn {4 dict begin /th exch def /z exch def /y exch def /x exch def
x th cos mul z th sin mul sub y z th cos mul x th sin mul add end} def
/zturn {4 dict begin /th exch def /z exch def /y exch def /x exch def
x th cos mul y th sin mul add y th cos mul x th sin mul sub z end} def
/xyzturn {-45 xturn -15 yturn -30 zturn} def
/trans {xyzturn /z exch def /y exch def /x exch def} def
```

と同じ. xturn は x,y,z,deg を貰い, x 軸に deg だけ回転した新 x,y,z を返す. xyzturn は x,y,z を貰い, −45 度 xturn し, 次に −15 度 yturn し, 最後に −30 度 zturn をする. 最後の trans は x,y,z を貰い, xyzturn し, 結果を x, y, z に格納する.

地球を描く

次に上の座標変換を使い地球を描く部分は, Postscript のプログラムにコメントを挿入しながら示す.

`/globe{0 0 200 xyzturn /zaxis exch def /yaxis exch def /xaxis  exch def`
図は x 軸の無限遠点から見たとして描く. 地球は 0, 0(原点) と 400, 400 の正方形の中にあり, 200, 200 が中心.

`200 yaxis sub 200 zaxis sub moveto yaxis yaxis add zaxis zaxis add rlineto stroke`
原点に対する北極の位置が 200−yaxis, 200−zaxis. そこへ moveto してから, yaxis, zaxis の 2 倍移動して南極まで地軸を描く.

`400 200 moveto 200 200 200 0 360 arc stroke`
半径 200 の円で地球の輪郭を描く.

`-75 15 75` 南緯 75 度から 15 度おきに北緯 75 度まで緯線を描く.

```
{/ph exch def /r 200 ph cos mul def /zz 200 ph sin mul def
xaxis yaxis zaxis r zz hiddenline} for
```
変数 ph が緯度, r がその緯度での地球半径. zz はその緯線の中心の地球中心からの距離. hiddenline は xaxis, yaxis, zaxis の法線方向を持つ半径 r の円を zz を中心にして描き, 地球の裏側を破線にする手続き. 後述

`lonbase 30 150 lonbase add` 経度を lonbase から 30 度おきに lonbase+150 度まで描く

```
{4 dict begin /lon exch def 200 lon sin mul 200 lon cos mul 0 xyzturn
/zaxis exch def /yaxis exch def /xaxis  exch def
/ph 0 def xaxis yaxis zaxis 200 0 hiddenline end}for
```
変数 lon に経度がある. その経度面の法線を計算し, xaxis, yaxis, zaxis に置いて経線を描く.

`drawmap` 地図を描く.

`}def %globe`

次は hiddenline. この中で thcalc を呼ぶ. これは中心が (0,0,z), 半径が r の円の半径方向の x 成分が 0 になる角度を計算する. その角度を境にして, 見えたり見えなくなったりする. 円にそって中心からのベクトルはある基準点からの角度 θ に対し,

$x = rcos\theta, y = rsin\theta, z = zz$. 座標変換で使った a, b, c, d, e, f で変換すると

$x' = e\,c\,x + (e\,d\,b + f\,a)y - (e\,d\,a - f\,b)z$ $\quad cos\theta = \sqrt{1 - sin^2\theta}$ だから $sin\theta = S$ と書いて

$x' = e\,c\,r(\sqrt{1 - S^2}) + (e\,d\,b + f\,a)r\,S - (e\,d\,a - f\,b)zz$

$(e\,d\,a - f\,b)zz = C$, $e\,c\,r/C = A$, $(e\,d\,b + f\,a)r/C = B$ と書くと $x' = A(\sqrt{1-S^2}) + BS - 1 = 0$ を解くことになる. S が得られたら arcsin で θ が得られる.
判別式が負のときは, 全周見えるか見えないので, 360, 360 を返す.

```
/thcalc {22 dict begin /zz exch def /r exch def
/alpha -45 def /beta -15 def /gamma -30 def
/a alpha cos def /b alpha sin def /c beta cos def /d beta sin def
/e gamma cos def /f gamma sin def
/A e c mul r mul def
/B e d mul b mul f a mul add r mul def
/C e d mul a mul f b mul sub zz mul def
/aa A A mul B B mul add def
/bb A C mul neg def
/cc C C mul B B mul sub def
/D bb bb mul aa cc mul sub def %discriminator
D 0 ge {/D D sqrt def
/sinth0 bb neg D add aa div def
/sinth1 bb neg D sub aa div def

/th0 sinth0 1 sinth0 sinth0 mul sub sqrt atan def
/th1 sinth1 1 sinth1 sinth1 mul sub sqrt atan def

/thtest {1 dict begin /th exch def
[r th sin mul r th cos mul zz transxyz2] 0 get abs 1.0 lt
{th} {th 180 lt {180 th sub} {540 th sub} ifelse} ifelse end} def

/th0 th0 thtest def /th1 th1 thtest def th0 th1}
{360 360} ifelse end} def
```

hiddenline はこの値を 2 つ貰う. xplus, xminus に保存. 見える点, 見えなくなる点は対称なので, 中央からの距離 xdist を求める. xsign は原点で見える, 見えないの指標. それが分かれば全周一括の場合は 0 360 見えないかどうか plotcirc を呼ぶ. 一部見えるときは 3 回にわけて円周を描く
-90 〜 $90 -$ xdist 見える
$90 -$ xdist 〜 $90 +$ xdist 見えない
$90 +$ xdist 〜 270 見える

```
/hiddenline {10 dict begin /zz exch def /r exch def
/zaxis exch def /yaxis exch def /xaxis exch def
/xsign 0 r zz trans x 0 lt {-1}{1}ifelse def
r zz thcalc /xminus exch def /xplus exch def
xminus 360 eq xplus 360 eq and{0 360 xsign 0 lt plotcirc}
{/xdist xplus xminus sub 2 div def
xdist 0 lt {/xdist xdist 180 add def} if
-90 90 xdist sub false plotcirc
90 xdist sub 90 xdist add true plotcirc
90 xdist add 270 false plotcirc} ifelse end} def  %hiddenline
```

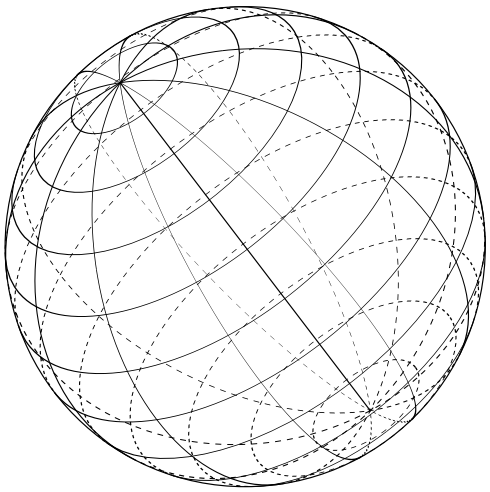
plotcirc は開始角度, 終了角度, 見えるかどうかを引数として受取り, 楕円を描く.

```
/plotcirc {/dash exch def /endang exch def /startang exch def
xaxis yaxis zaxis yaxis ph sin mul 200 add zaxis ph sin mul 200 add
200 ph cos mul startang endang dash ellipse} def
```

plotcirc の下請の ellipse は x, y, z, y0, z0, r, ang0, ang1, dash の引数を 9 個とる．dash は実線を引くか破線を引くかで setdash に使う．y0,z0 を中心とし, 半径 r の楕円を ang0 から ang1 まで描く．x,y,z はこの楕円の法線のベクトル.

```
/ellipse {11 dict begin
/dash exch def /ang1 exch def /ang0 exch def /r exch def
/z0 exch def /y0 exch def /z exch def /y exch def /x exch def
gsave
y0 z0 translate z y atan 90 sub rotate 1 x 200 div scale
dash {[5 5]0 }{[]0}ifelse setdash
r ang0 cos mul r ang0 sin mul moveto 0 0 r
ang0 ang1 arc stroke grestore end} def
```

楕円は y,z で長軸の方向が分かるので, y0,z0 に中心を移動した後, その分画面を回転する．また x が楕円の扁平度を示すので, x 方向は 1, y 方向を x/200 に scale する．そして出発点に移動した後, 0 0 r ang0 ang1 arc stroke で楕円を描く．扁平な楕円だと, 線の太さに影響がでるが, 経線, 緯線は細く描くので問題にならない.
後で回線ルートを大圏コースで描くときは, 太さに影響が及ぶので, 別の方法で描いている.



地図の描画

Mercator 図法では緯度 ϕ を $y = \ln(\tan(\pi/4 + \phi/2))$ に置く. 逆変換は latc で行う
地図は google で探し, 国境のない Mercator の世界地図を得た. この地図データは

```
fffffffffffffffffffffffffffffffffffffffffbfffffffffeefffeffffffffffef6
effffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffff
```

のような列が並んでいる. それを

```
<000000000000000000000000000000000000000000006000006000000000
 400000000004000000000000000000000000000000000000000000000000
 000000000000000000000000000000000000000000000000000000000000>
```

のように変換して world なる配列に持っている. <, >の間は十六進なので, この 2 桁で 1 文字, つまり 8 ビット文字が 90 文字並んでいる. 縦 560 ビット, 横 720 ビットの地図になっている.

```
/drawmap{  %used in globe
0 1 559
{/y exch def /line world y get def
/lat y latc def
0 1 89
{/x exch def /char line x get def
char 0 gt {
char 128 and 0 gt {0 plot} if
char 64 and 0 gt {1 plot} if
char 32 and 0 gt {2 plot} if
char 16 and 0 gt {3 plot} if
char 8 and 0 gt {4 plot} if
char 4 and 0 gt {5 plot} if
char 2 and 0 gt {6 plot} if
char 1 and 0 gt {7 plot} if} if
} for} for}def
```

y を 0 から 559 まで変えながら, world から y 番目の行をとり line に置く. この y がどの緯度に対応するかを latc で計算し lat へ置く. 次に x を 0 から 89 まで変えながら, line から x 番目の文字をとり char に置く. char が 0 なら描くべき点はない. char0 なら上の bit から順に 1 かどうかをみ, 1 なら何ビット目を引数にして plot を呼び, 地球上に点を描く.

```
/plot {1 dict begin /n exch def %used in drawmap
x 8 mul n add lonc lat globec
{moveto 1 0 rlineto 0 1 rlineto -1 0 rlineto closepath fill}{pop pop}
ifelse end}def
```

plot は引数を n とし, 8x+n を lonc で経度にし, lat とともに globec を呼ぶ. globec は緯度経度を x,y,z の座標系に変換し, trans をかけて描画すべき位置に変換する.

```
/globec {2 dict begin /la exch def /lo exch def
200 lo sin mul la cos mul 200 lo cos mul la cos mul 200 la sin mul
trans y 200 add z 200 add x 0 gt end} def
```

plot に戻り, その位置に 1×1 ピクセルの黒点を描く. lonc, latc は以下の通り

```
/lonc {1 dict begin
/x exch def x 327 sub 180 mul 335 div neg lonbase add
end} def
/latc {2 dict begin  /y exch def /y1 315 y sub 2.03 mul 218 div def
y1 0 lt{2.718282 y1 exp 1 atan 45 sub 2 mul}
       {2.718282 y1 neg exp 1 atan 45 sub 2 mul neg} ifelse end} def
```

大圏コース

地球上にインターネットのルートを描くのがこのプログラムの最後の目的である. A 点から B 点への線は常識的には大圏 (great circle) にそって描くことになる.
このプログラムで tokyo から chicago へルートを各部分は次のようになっている.

```
/tokyo [140 36] def
/chicago [360 88 sub 42] def
chicago conv tokyo conv great
```

最初の 2 行は tokyo, chicago の経度緯度を与える. chicago は西経にあるので, 360-88 で東経に変換している.
conv は x, y, z 座標への変換で

```
/conv {4 dict begin /lonlat exch def /lon lonlat 0 get def
/lat lonlat 1 get def /th lon lonbase sub 90 add def
[200 lat cos mul th cos mul 200 lat cos mul th sin mul  200 lat sin mul] end}
def
```

となっている. [x,y,z] の形のリストが返る. それを 2 つ置いて great を起動する

```
/great {10 dict begin /vect1 exch def /vect0 exch def %[unit 0 0][0 unit 0]
vect0 0 get vect0 1 get vect0 2 get trans /y0 y def /z0 z def
vect1 0 get vect1 1 get vect1 2 get trans /y1 y def /z1 z def
vect0 vect1 vectorprod  trans
/ang z y atan 90 sub def /ratio x unit div def
/ang0 y0 z0 ang neg rot ratio div exch atan def
/ang1 y1 z1 ang neg rot ratio div exch atan def
x y z 0 0 unit ang0 ang1 false ellipse2 end} def
```

大圏コースの描き方は, 地球の中心 O から A 点と B 点への 2 つのベクトル $\vec{OA}$, $\vec{OB}$ を用意し, そのベクトル積を計算する. これは $\vec{OA}$, $\vec{OB}$ を含む面と垂直な方向を持つはずで, それを長さ 200 に正規化する.

立体視

anaglyph, つまり赤とシアンの目がねによる立体視は, 多少の視差をもって描いた赤とシアンの 2 つの図を用意しなければならない.
このプログラムでは最後の方に,

```
1 0 0 setrgbcolor
globe route                              %left eye
5 0 translate /xyzturn {-45 xturn -15 yturn -35 zturn} def
0 1 1 setrgbcolor globe route                              %right eye
```

という部分があり, まず rgbcolor を 1 0 0(赤) にし, globe で地球を描く. 次に描く位置を 5 だけ右にずらす (5 0 translate), また z 軸の回転を 30 度から 35 度に変更した xyzturn を定義し, rgbcolor を 0 1 1(シアン) に設定してもう一度 globe を呼ぶ.
これで重なってはいるが, 立体視用の絵ができた.

回転アニメ用の図の連続生成

アニメ用には多くの原図が必要である. 今回は地球の自転なので, 3 度おきに 120 枚の絵を作ることにした. その制御は以下の utilisp のプログラムによる.

```
(defun gengif ()
(do ((deg 0 (+ deg 3))) ((= deg 360))
(setq outfile (string-append "gl" (number-image deg) ".ps"))
(setq s0 (inopen (stream "anaglyph8.ps")))
(setq s1 (outopen (stream outfile)))
(princ "%!" s1) (terpri s1)
(princ (string-append "/lonbase " (number-image deg) " def") s1) (terpri s1)
(readline s0) (readline s0)
(let ((err:end-of-file '(lambda (x (y)) (throw 'loop))))
(catch 'loop
 (loop (princ (readline s0) s1) (terpri s1))))
(close s1)
(call (string-append "./pstogif gl" (number-image deg)))
(call (string-append "rm " outfile))
(call (string-append "rm gl" (number-image deg) ".ppm"))
))
(gengif)
```

2 行目の do は deg を 0 から 3 度おきに, 360 度になるまで反復する指令. 次に出力ファイル名 gl0.ps のようなものを用意.
s0, s1 は入力と出力のポート名で, オープンしておく. 出力に

```
%!
/lonbase 0 def
```

と経度の基準の角度を設定する. この角度が 0 から 3 おきで 357 までになる.
いまの 2 行に対応する部分を入力から読み飛ばすのが次の readline(2 回) で, その後, end-of-file が'loop に throw するように設定し, catch 'loop としてコピーのループに入る. あとは最後の列までコピーするだけである.
end-of-file を検出すると catch でつかまり, 入力は自動的にクローズされる. s1, つまり出力をクローズしたあと, call を使い, shell のコマンドを呼ぶ.

まず .pstogif gl0 で出力画像の gif 版を作る.
次に rm gl0.ps でコピーで作った ps ファイルを棄てる.
また rm gl0.ppm で途中で作った ppm ファイルも棄てる.
こうして gl.gif のファイルが 120 個出来る. これはあとで rotation.html が利用するので, おなじ URL に移転しておく.

rotation.html

```
<html>
<head>
<title>rotating globe animation</title>
</head>
<body>
<img src="anaglyph/gl0.gif" name="animation">

<script language="javascript">
var images = new Array(120);
for(var i = 0; i< 120; i++)
{var j=357-i*3;
 images[i]= new Image();
 images[i].src = "anaglyph/gl" + j + ".gif";}


function animate()
{document.animation.src = images[i].src;
 i=(i+1)%120;
 timeout_id = setTimeout("animate()", 100);}

var i=0;
var timeout_id=null;
</script>

<form>
<input type=button value="Start"
       onClick="if (timeout_id == null) animate();">
<input type=button value="Stop"
       onClick="if (timeout_id) clearTimeout(timeout_id); timeout_id=null;">
</form>
</body>
</html>
```

大きさ 120 個の images に今作った gif ファイルを読込み, setTimeout(”animate()”, 100); で 100 単位ごとに, 画像を次々と読み出す. これで地球の自転する様子が見えるようになった.