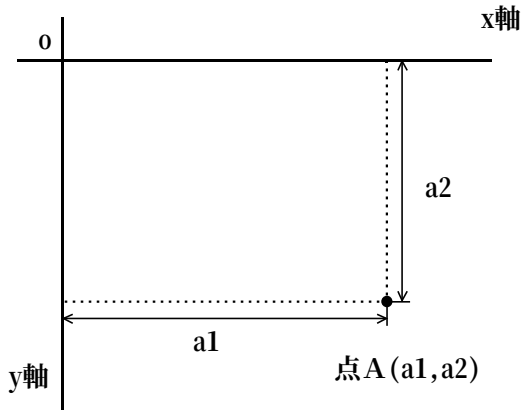
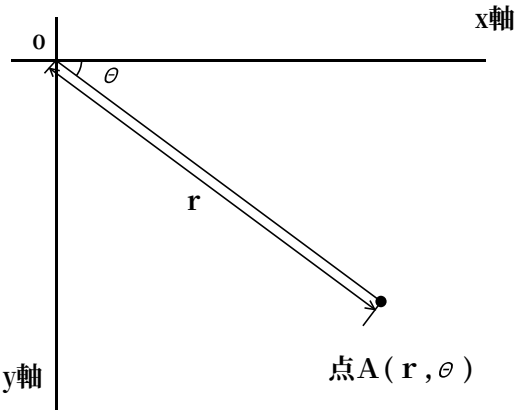
画像処理に必要な数学その１

座標　・・・・・　空間内の位置を数値で表したもの

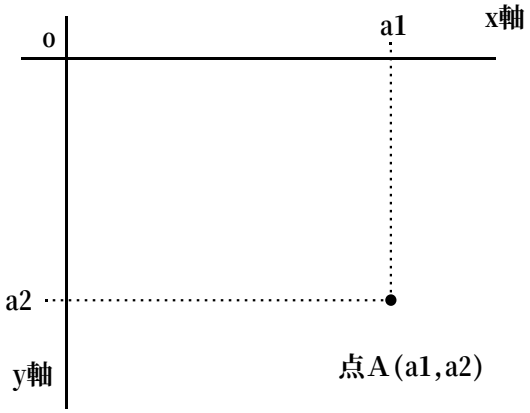
【デカルト座標】



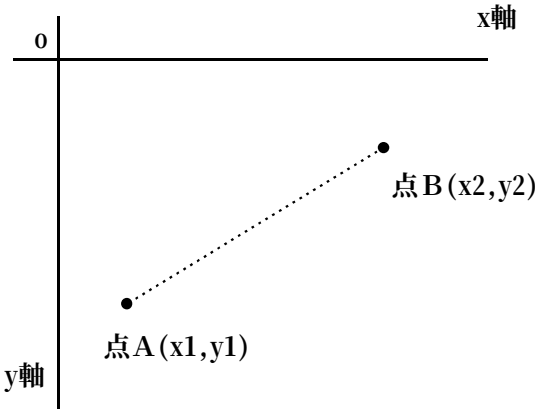
【極座標】



ただ、一般には以下の様に
考えた方が良いかも知れない。



線分　・・・・・・　２点の間をつないだもの

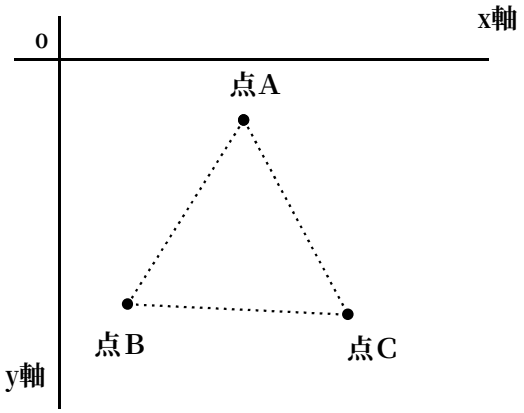


$$y = \frac{(y2 - y1)x + y1x2 - x1y2}{(x2 - x1)}$$

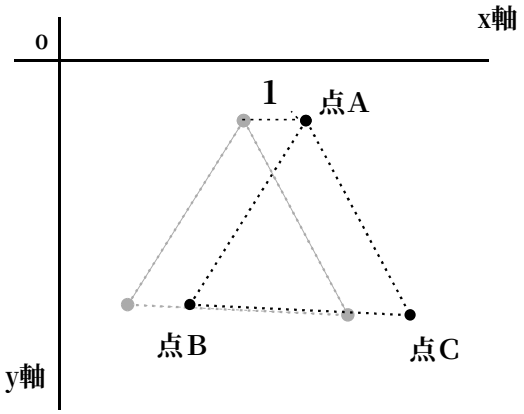
または

$$x = \frac{(x2 - x1)y + x1y2 - y1x2}{(y2 - y1)}$$

ポリゴン　・・・・　三点によって表現される図形



ポリゴンの移動



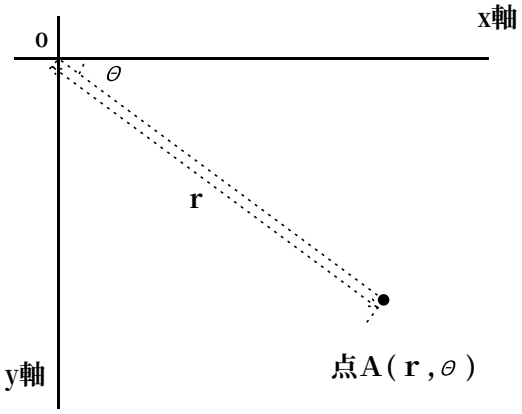
点A(a1,a2)　→　(a1+1,a2)

点B(b1,b2)　→　(b1+1,b2)

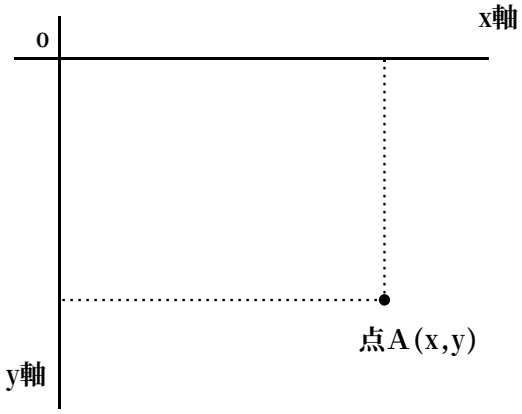
点C(c1,c2)　→　(c1+1,c2)

極座標・・・デカルト座標の変換

【極座標】



【デカルト座標】

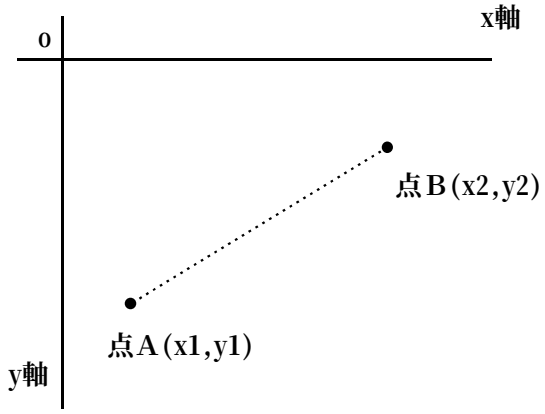


$$x = r\cos(\theta)$$

$$y = r\sin(\theta)$$

画像処理の為の数学２

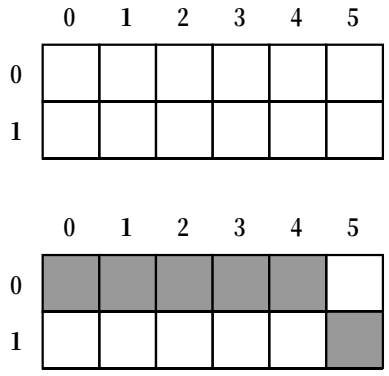
線分のアルゴリズム（プログラム化時の問題点）



$$y = \frac{(y2 - y1)x + y1x2 - x1y2}{(x2 - x1)}$$

または

$$x = \frac{(x2 - x1)y + x1y2 - y1x2}{(y2 - y1)}$$

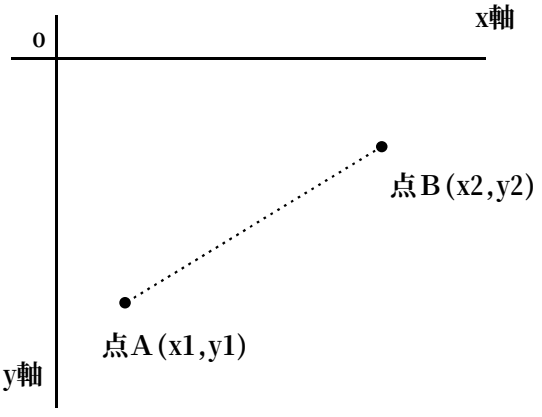


点A(0,0)
点B(5,1)

x:0　0.0
x:1　0.2
x:2　0.4
x:3　0.6
x:4　0.8
x:5　1.0

小数点以下切り捨てなら

線分その２（別の考え方・・・線の描画には使わないが、テクスチャマッピングなどで使用。）



０ ≦ｕ≦１．０　のとき

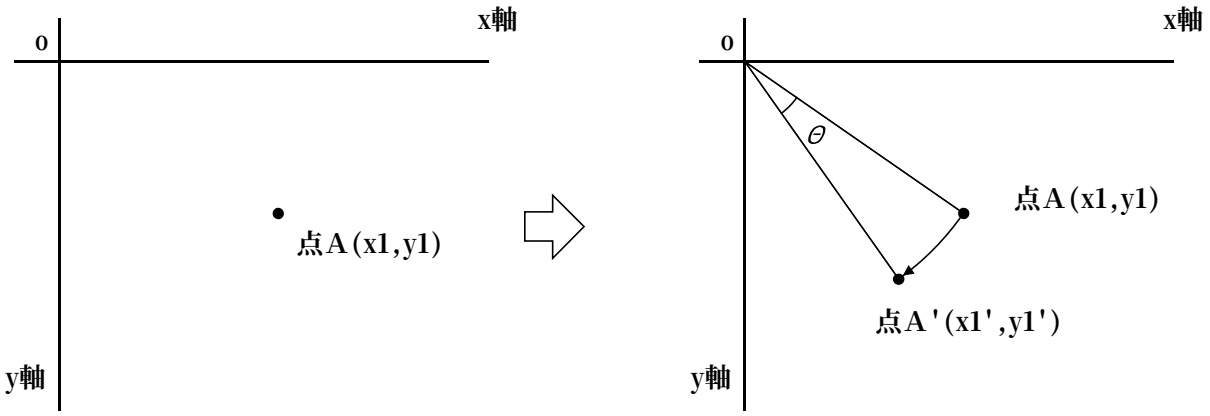
$$\begin{cases} x = x1(1 - u) + x2u \\ y = y1(1 - u) + y2u \end{cases}$$

上の式を　u でまとめると

$$\begin{cases} x = x1 + (x2 - x1)u \\ y = y1 + (y2 - y1)u \end{cases}$$

座標の回転移動

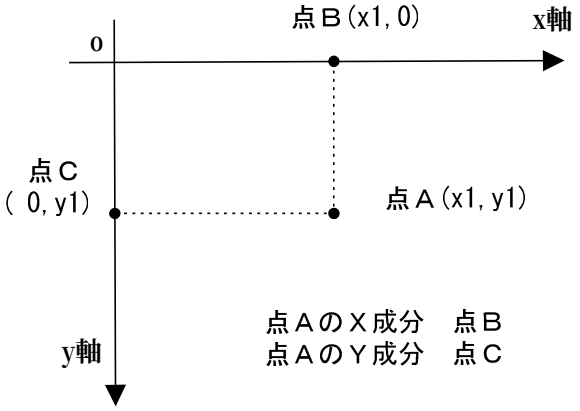
点A(x1,y1)を原点Oを中心として角度Θだけ回転させる。



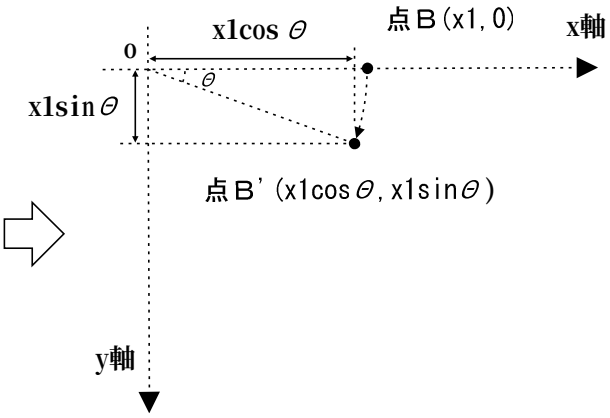
$x1' = x1\cos\theta - y1\sin\theta$

$y1' = x1\sin\theta + y1\cos\theta$

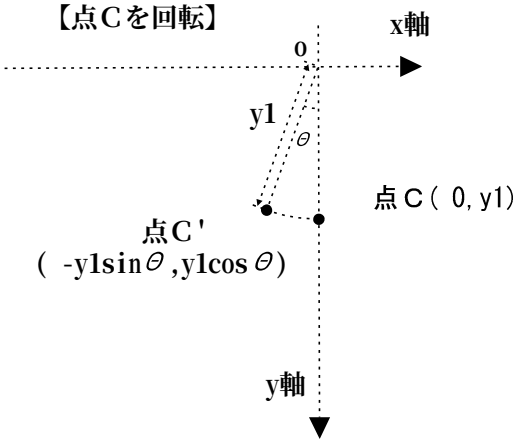
【解説】



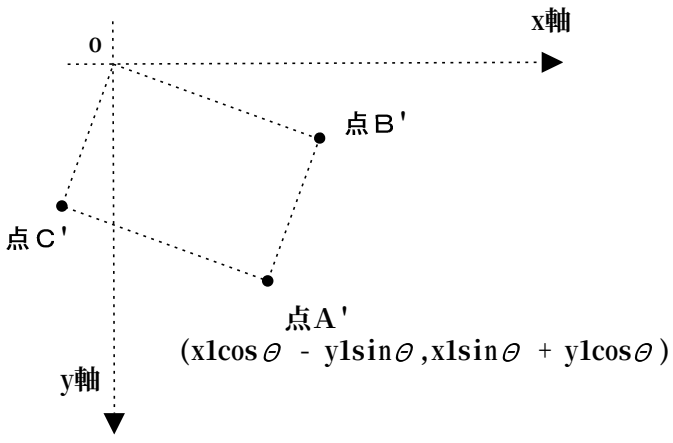
【点Ｂを回転】



【点Ｃを回転】



【結果】



画像処理の為の数学その３

行列(Matrix)について

$$\begin{pmatrix} 5 & 10 & 7 \\ 12 & 9 & 1 \\ 6 & 2 & 8 \end{pmatrix}$$

左のような数字の組み合わせを行列という
その名の通り、行と列とからなっている。

【行列の加算】

$$\begin{pmatrix} 5 & 10 & 7 \\ 12 & 9 & 1 \\ 6 & 2 & 8 \end{pmatrix} + \begin{pmatrix} 3 & 1 & 10 \\ 6 & 8 & 2 \\ 14 & 5 & 11 \end{pmatrix} = \begin{pmatrix} 8 & 11 & 17 \\ 18 & 17 & 3 \\ 20 & 7 & 19 \end{pmatrix}$$

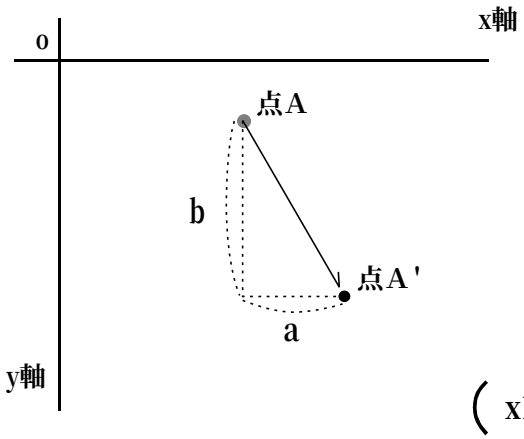
各要素ごとに加算を行い、同じ行同じ列へ結果を返す
減算の場合も同じ、＋記号の左右の行列は、行数・列数ともに同じでなければ
ならない。

【行列の乗算】

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}\begin{pmatrix} j & k & l \\ m & n & o \\ p & q & r \end{pmatrix} = \begin{pmatrix} aj+bm+cp & ak+bn+cq & al+bo+cr \\ dj+em+fp & dk+en+fq & dl+eo+fr \\ gj+hm+ip & gk+hn+iq & gl+ho+ir \end{pmatrix}$$

乗算する２つの行列のうち、左側の行列の一列と右側の行列の一行とで演算を行う。

座標の移動



$x1' = x1 + a$

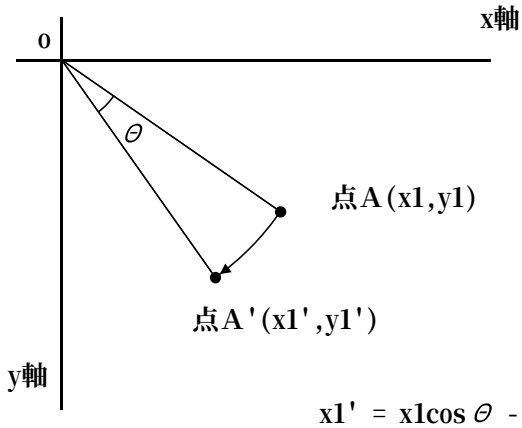
$y1' = y1 + b$

⇩ 行列を使用すると、下のように書き換えられる。

$$\begin{pmatrix} x1' & y1' & 1 \end{pmatrix} = \begin{pmatrix} x1 & y1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{pmatrix}$$

座標の回転移動について

点A(x1,y1)を原点Oを中心として角度Θだけ回転させる。



$$\begin{pmatrix} x1' & y1' & 1 \end{pmatrix} = \begin{pmatrix} x1 & y1 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$x1' = x1\cos\theta - y1\sin$

$y1' = x1\sin\theta + y1\cos\theta$

行列を使用すると、上のように書き換えられる。

回転してから移動

$$\begin{pmatrix} x1' & y1' & 1 \end{pmatrix} = \begin{pmatrix} x1 & y1 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{pmatrix}$$

⇩

$$\begin{pmatrix} x1' & y1' & 1 \end{pmatrix} = \begin{pmatrix} x1 & y1 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ a & b & 1 \end{pmatrix}$$

画像処理の為の数学その４

回転行列について

点A(x1,y1,z1)を原点Oを中心としてZ軸まわりに時計回りで角度Θだけ回転させる行列。

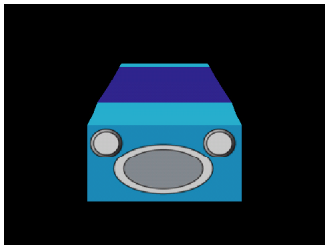
$$\begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

【使用方法】

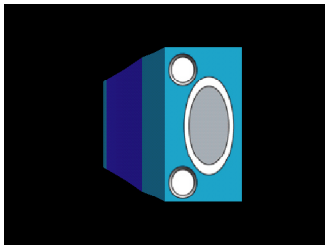
$$\begin{pmatrix} x1' & y1' & z1' & 1 \end{pmatrix} = \begin{pmatrix} x1 & y1 & z1 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

※時計回りというのは、Z軸のプラス側から原点の方を見て、時計回りの意。右の図は半時計回りに見えるが、カメラはZ軸のマイナス側から原点を見ているので、回転が逆方向に見えているのである。

【回転前】



【時計回りに９０度回転】



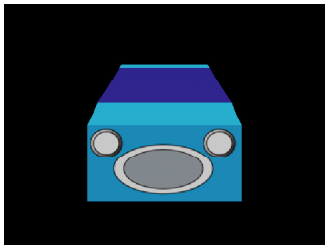
点A(x1,y1,z1)を原点Oを中心としてY軸まわりに時計回りで角度Θだけ回転させる行列。

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

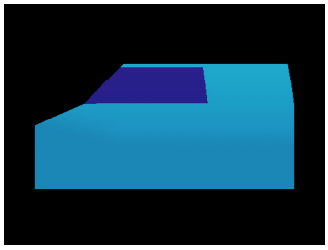
【使用方法】

$$\begin{pmatrix} x1' & y1' & z1' & 1 \end{pmatrix} = \begin{pmatrix} x1 & y1 & z1 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

【回転前】



【時計回りに９０度回転】



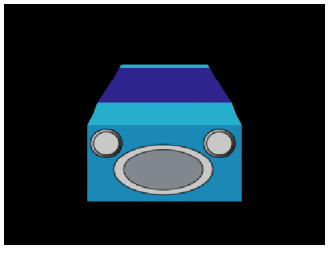
点A(x1,y1,z1)を原点Oを中心としてX軸まわりに時計回りで角度Θだけ回転させる行列。

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

【使用方法】

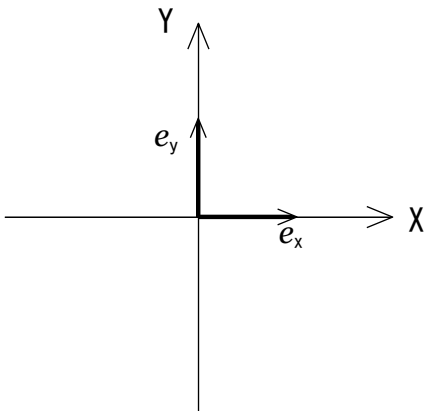
$$\begin{pmatrix} x1' & y1' & z1' & 1 \end{pmatrix} = \begin{pmatrix} x1 & y1 & z1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

【回転前】



【時計回りに９０度回転】

回転の考え方その２



X軸に平行なノルム１のベクトル

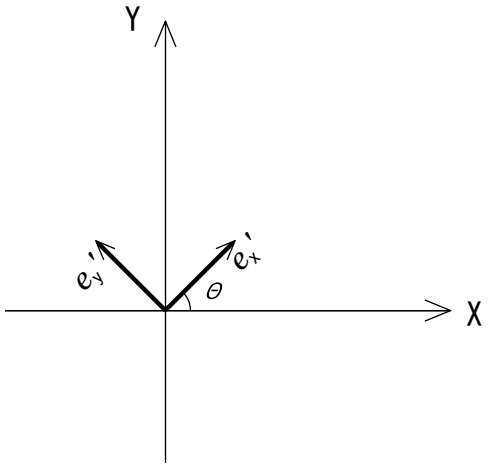
$$e_x=(1, 0)$$

Y軸に平行なノルム１のベクトル

$$e_y=(0, 1)$$

と置くと、任意の位置ベクトル(x_1, y_1) は、
$x_1e_x + y_1e_y$ で示す事ができる。

（ノルム１のベクトルを単位ベクトルと呼ぶ）



e_xを反時計まわりに、θ だけ回転させたベクトル e_x'は、

$$e_x'=(\cos\theta,\ \sin\theta)$$

e_yを反時計まわりに、θ だけ回転させたベクトル e_y'は、

$$e_y'=(-\sin\theta,\ \cos\theta)$$

で表すことができる。

任意の位置ベクトル(x_1, y_1) を反時計まわりに、θ だけ回転させた位置は、以下の式で示す事ができる。

$$(x_1', y_1') = x_1e_x' + y_1e_y'$$

これを行列で表現すると、以下のようになる。

$$\begin{pmatrix} x_1' & y_1' & 1 \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

（e_x'：$\cos\theta$ $\sin\theta$ 0 ／ e_y'：$-\sin\theta$ $\cos\theta$ 0）

３Ｄの場合、

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

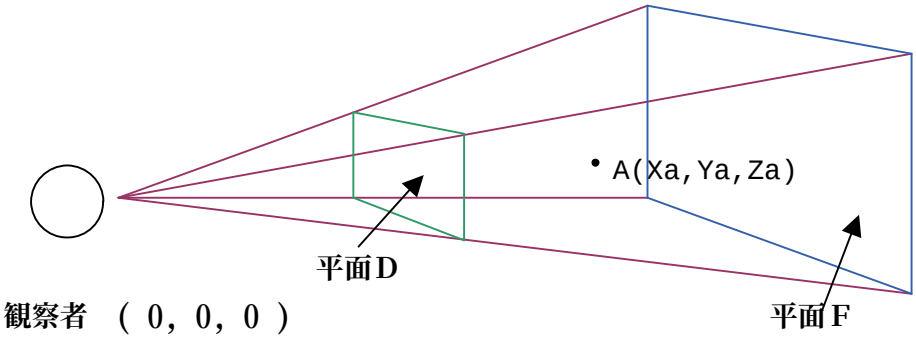
（e_x：1 0 0 0 ／ e_y：0 1 0 0 ／ e_z：0 0 1 0）

回転行列は、座標軸に平行な３本の単位ベクトルを回転させたものを並べた行列である。
これを利用すると、任意の回転を行列に変換する事ができる。

画像処理のための数学その５

３Ｄ座標からスクリーン座標への変換

３ＤＣＧでは、ある点の座標は三次元空間内の位置で表されるが、コンピュータの
スクリーンはあくまで平面しか表現できない為、３ＤＣＧを実際に画面に表示する為には、
三次元空間内の任意の座標を二次元座標に変換する必要がある。
方法はというと、以下の図をみて欲しい。



上記図中の点 A(Xa, Ya, Za) を、スクリーンに表示したいとする。
今回のシステムでは、平面Ｄ(z = Zn)と平面Ｆ(z = Zf)の間に挟まれた点について、
画面表示を行うものと仮定する。また、両平面の中心をＺ軸が通っているものとする。
通常、平面Ｆより遠くにある点および平面Ｄより手前にある点については処理を行わない。

上記四角錐のうち、平面Ｄと平面Ｆで囲まれた立体部分を、ビュー･ボリュームと呼ぶ。
この時､点 A(Xa,Ya,Za)をスクリーン上に投影する時､どのようにすれば良いかというと、
具体的には以下のように行う。

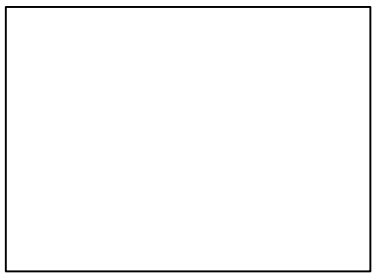
スクリーン上の座標を(Sx,Sy) とすると。

$$Sx = \frac{Xa * w}{Za} \qquad Sy = \frac{Ya * h}{Za}$$

w 横方向の補正係数。
h 縦方向の補正係数。
（画面の縦横比などから計算）

これにより、３Ｄ座標が２Ｄ座標に変換される。

(-1.0,1.0)　　(1.0,1.0)



(-1.0,-1.0)　　(1.0,-1.0)

ただし、ここでの座標は、左図のようなスクリーンを想定している。実際の画面解像度に合わせるためには、別途の変換も必要である。

３Ｄから２Ｄへの変換式は割り算を含んでいる為、ベクトルと行列の掛け算で表す事はできない。
そこで一計を案じる。

一次変換の結果（Xa’, Ya’, Za’, Wa ）に対して、以下の演算を行う事で２Ｄ座標を算出するという事をルールとする。

$$Sx = \frac{Xa'}{Wa} \qquad Sy = \frac{Ya'}{Wa} \qquad Sz = \frac{Za'}{Wa}$$

この時、以下の変換を考える。

$$\begin{pmatrix} x_a' & y_a' & z_a' & w_a \end{pmatrix} = \begin{pmatrix} x_a & y_a & z_a & 1 \end{pmatrix} \begin{pmatrix} w & 0 & 0 & 0 \\ 0 & h & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

すると、

$$x_a' = x_a * w$$
$$y_a' = y_a * h$$
$$z_a' = z_a$$
$$w_a = z_a$$

$$Sx = \frac{Xa'}{Wa} = \frac{Xa*w}{Za}$$

$$Sy = \frac{Ya'}{Wa} = \frac{Ya*h}{Za}$$

この行列をプロジェクション行列と呼ぶ。

z_a' の値はポリゴンの前後判定に使いやすいように加工される事が多い、例えば DirectX の D3DXMatrixPerspectiveFovLH では、

```
D3DXMATRIX *
D3DXMatrixPerspectiveFovLH( D3DXMATRIX *pOut,FLOAT fovY,FLOAT Aspect, FLOAT Zn, FLOAT Zf);
```

に対して、以下のプロジェクション行列を出力する仕様となっている。

$$\begin{pmatrix} w & 0 & 0 & 0 \\ 0 & h & 0 & 0 \\ 0 & 0 & \frac{Zf}{(Zf-Zn)} & 1 \\ 0 & 0 & \frac{-Zn*Zf}{(Zf-Zn)} & 0 \end{pmatrix}$$

以下の計算が追加されている。

$$z_a' = \frac{z_f \quad (z_a - z_n)}{z_f - z_n}$$

ようするに以下の式の範囲が、0～1.0 となる。

$$Sz = \frac{Za'}{Wa}$$

（この式に数値を入れてグラフを出してみると興味深い。近くでの前後判定の精度が高く、遠くの精度は低く設定される。）

※付録１参照

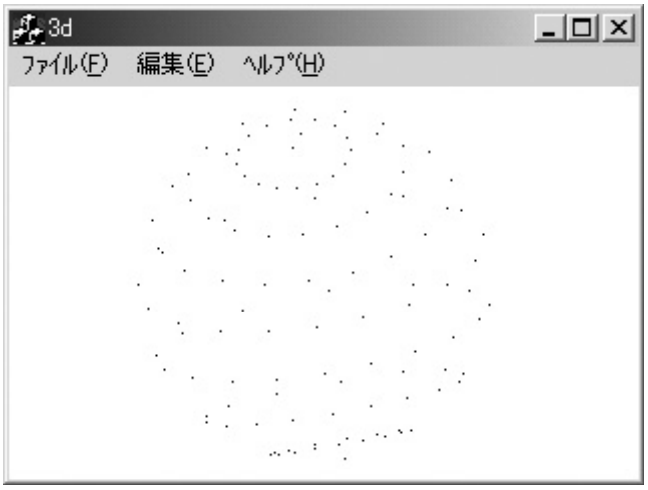
h = cot(fovY/2)

w = h / Aspect

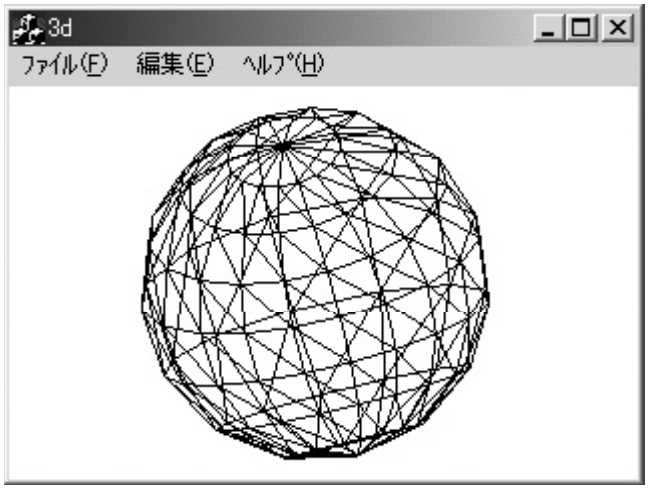
$$\cot\theta = \frac{1}{\tan\theta}$$

画像処理の為の数学その６

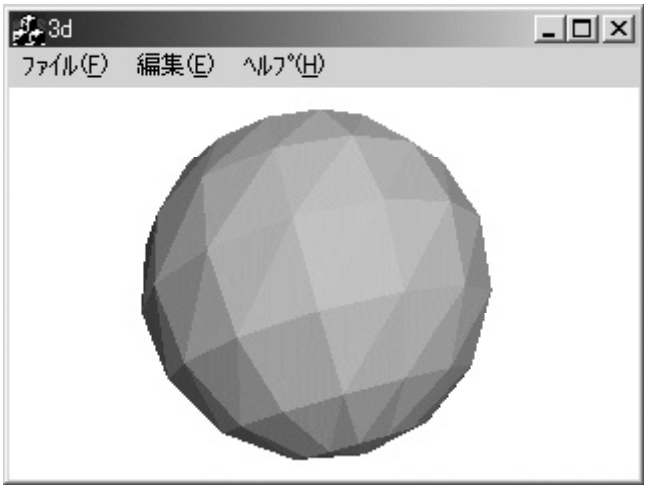
３Ｄモデルについて



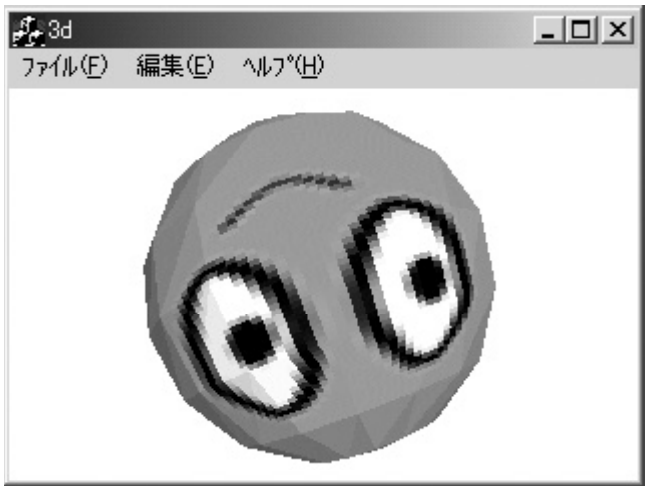
ポイントクラウド



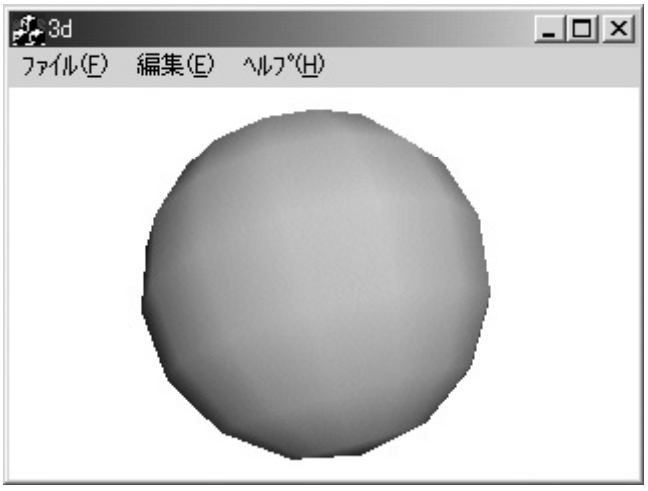
ワイヤーフレーム



ポリゴン



テクスチャ付きポリゴン



ポリゴン（グーローシェーディング）

テクスチャ付きポリゴン

上記はまったく同じモデリングデータを、描画の仕方を変えて表示させたものである。
基本的にデータは「ポイントクラウド」が基本である。

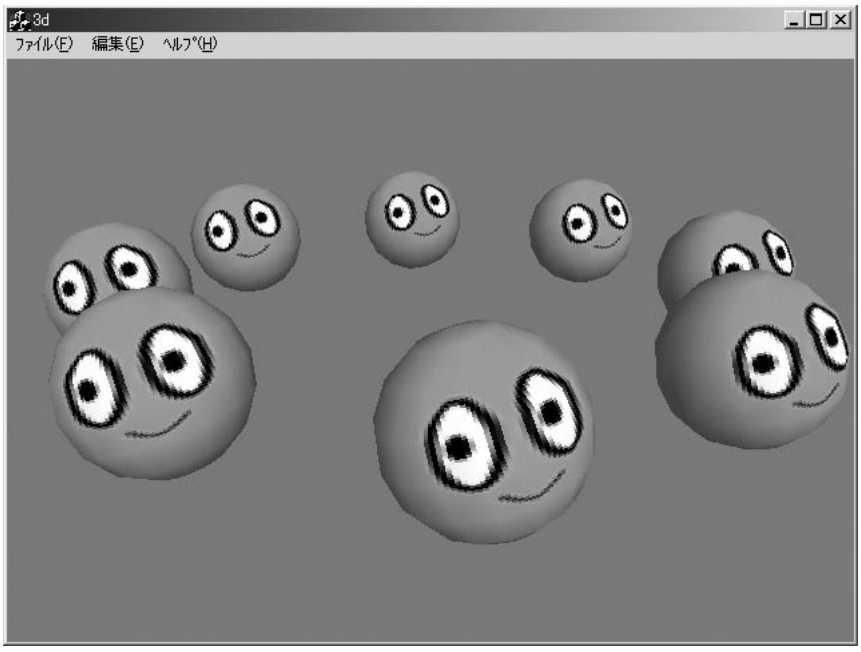
座標系について。
　頂点を数値で表現する為には、基準となる原点が必要である。この原点だが、
３Ｄグラフィックスでは、使用される原点は一つとは限らない。
一つの原点で表現される座標のシステムを座標系という。

よく利用される座標系

物体座標系	モデルデータ内部で頂点の位置を表している座標
世界座標系	複数のモデルからなるシーンの基準座標
視座標系	カメラ位置を原点とした座標
スクリーン座標系	ディスプレイ上の二次元座標

実際にモデルデータがスクリーンに表示される時には、モデルデータを記述している物体座標系で表されている頂点が世界座標系に変換され、次いで視座標系に変換される。最後にスクリーン座標系に変換されて、実際に表示される。

ゲームの場合、下の図の様に同じモデルデータを、さまざまな場所に表示させるケースが多い。　この際、先ほどの考え方が便利である。



モデルデータは本来一つの原点を持っている。この原点を、世界座標系の中に配置してやるという考え方である。　実際、ゲームというものを作る際には、世界座標系を基準にして考える事が多い。

具体的には、一つのモデルデータに対して回転や移動といった座標変換を行ってから表示するのだが、

実際には、表示を行う時にモデルデータと変換行列を用意して描画エンジンに変換行列を渡して、表示を行うという事をする。

この時、モデルデータ内の各頂点の座標は、以下の方法で計算される。

$$\begin{pmatrix} x1' \\ y1' \\ z1' \\ 1 \end{pmatrix} = \begin{pmatrix} \text{変換行列} \end{pmatrix} \begin{pmatrix} x1 \\ y1 \\ z1 \\ 1 \end{pmatrix}$$

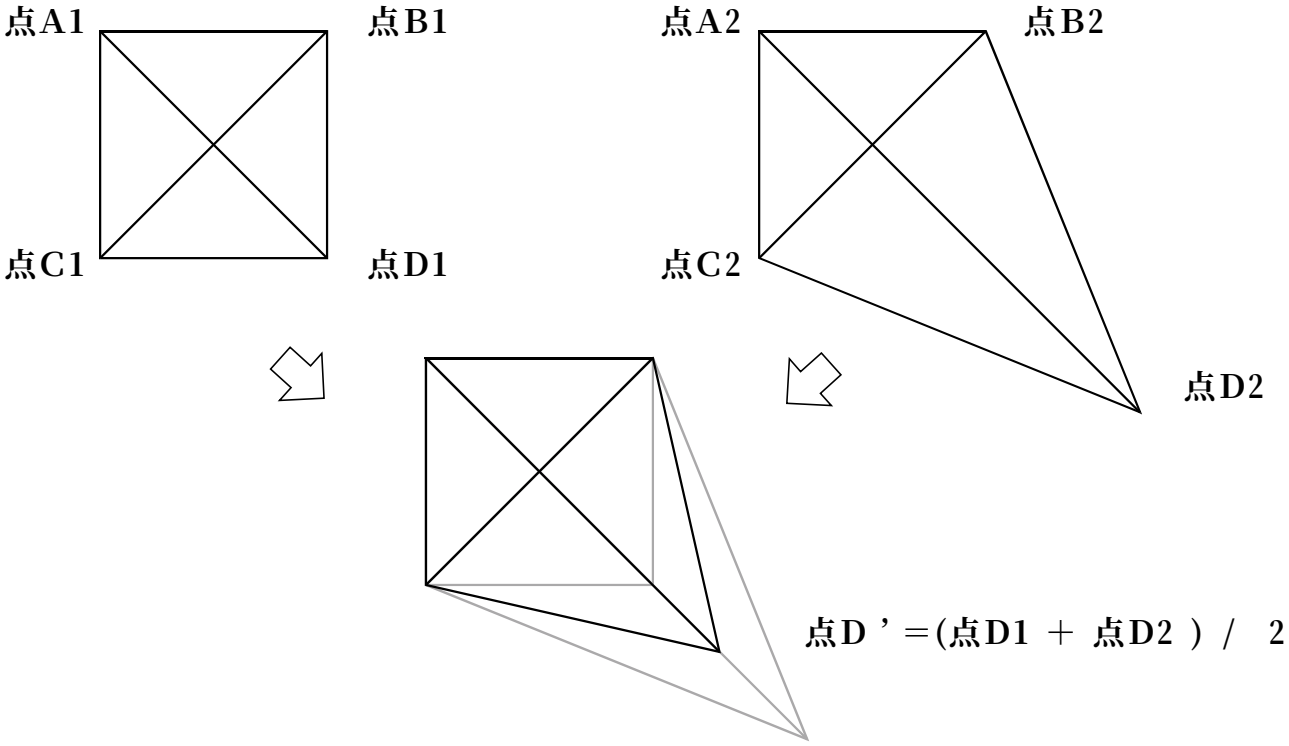
描画エンジンの中で、この計算を行う部分を、ジオメトリエンジンと呼ぶ事が多いようである。

アニメーション３種

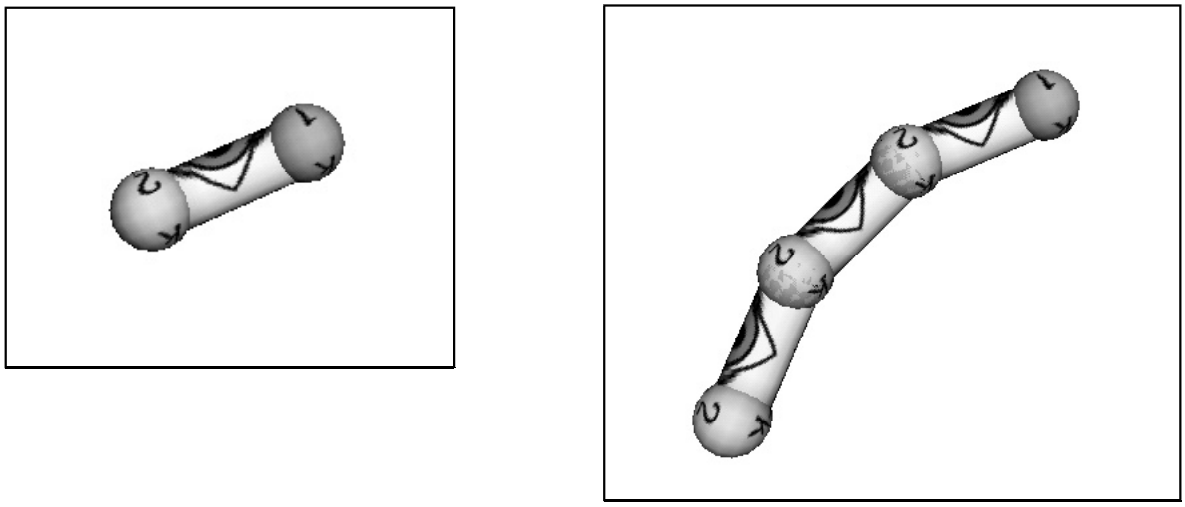
３Ｄモデルの形状を変更させるアニメーションは、主に以下の３種類を用いる事が多い。

頂点アニメーション（処理　軽い、データ　重い、見映え　良好）

２Ｄのアニメーションと同様、さまざまな形の３Ｄモデルを作成し、それぞれの形状の中間の形状をなんらかの補間によって生成するケースが多い。

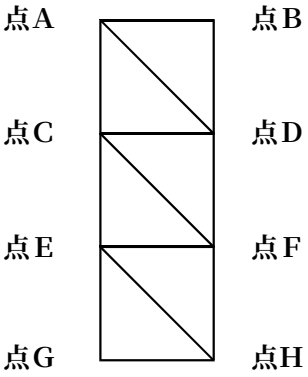


関節アニメーション（処理　軽い、データ　軽い、見映え　劣る）

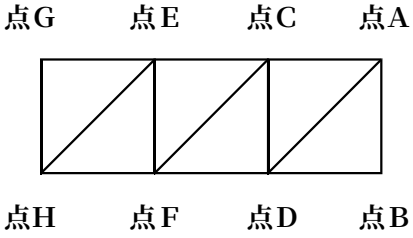


複数の３Ｄモデルを繋がって見えるように配置し、繋がって見えるように保ちながら、別々に動かす方法である。
上の図は、まったく同じものをつないで表示した例である。

ボーンの利用 （処理　重い、データ　軽い、見映え　良好）



元データ

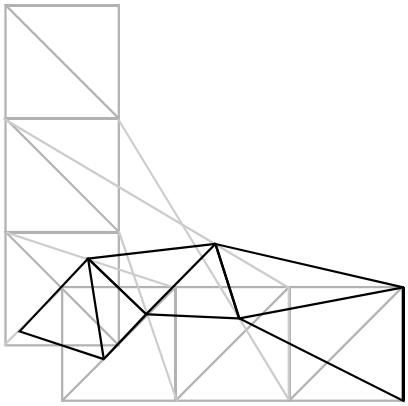


変更後データ（時計方向に９０°回転）

この時、全ての頂点を同じ様に動かすのでは無く、それぞれの頂点で、変更の影響を受ける度合いを変えるとどうなるだろうか？

影響を受ける度合い

点A・B　１００％
点C・D　　９０％
点E・F　　６６％
点G・H　　３３％



こうすると、関節アニメーションのように、カクと曲がるのではなく、やわらかく曲がるものを表現できるため、動物や人間を表現する際に、良く使われるようになって来ている。

ただし、大きく曲げると、形状が破綻する為、破綻しない程度に利用する必要がある。

一般には、関節アニメーションと組み合わせる。関節に近い頂点は、変形の影響をあまり受けないようにし、関節から離れる程、影響を強く受けるようにすると、ちょうど骨の周りに皮膚がついているような動きになる。

この為、この手法を「スキン（皮膚）」とか「ボーン（骨）」とか言う用語で表す事が多い様である。

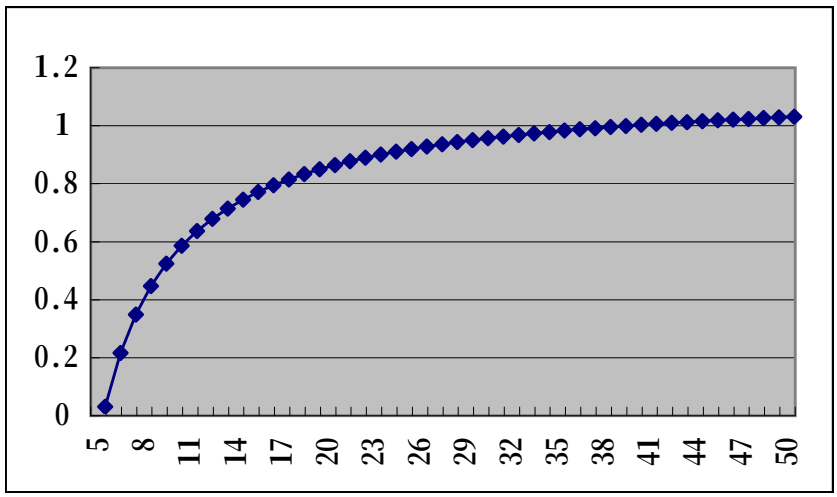
付録１：DirectXにおけるZバッファ値の算出について

Z	Sz
5	0
6	0.185185
7	0.31746
8	0.416667
9	0.493827
10	0.555556
11	0.606061
12	0.648148
13	0.683761
14	0.714286
15	0.740741
16	0.763889
17	0.784314
18	0.802469
19	0.818713
20	0.833333
21	0.846561
22	0.858586
23	0.869565
24	0.87963
25	0.888889
26	0.897436
27	0.90535
28	0.912698
29	0.91954
30	0.925926
31	0.9319
32	0.9375
33	0.942761
34	0.947712
35	0.952381
36	0.95679
37	0.960961
38	0.964912
39	0.968661
40	0.972222
41	0.97561
42	0.978836
43	0.981912
44	0.984848
45	0.987654
46	0.990338
47	0.992908
48	0.99537
49	0.997732
50	1

Zn ： 5.0　Zf:　50.0
とした場合の、変換後の Zの値をプロット。
Sz の算出方法は、

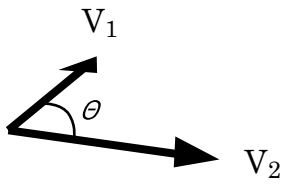
$$Sz = Zf * (Z - Zn) / \{ Z * (Zf - Zn) \}$$

である。このSz はポリゴンの前後判定に用いられる。



付録２．内積と外積

$V_1 = (x_1, y_1, z_1)$　　$V_2 = (x_2, y_2, z_2)$　　それらのなす角をθとしたとき、



内積　$V_1 \cdot V_2$　は、以下の式で求められる。

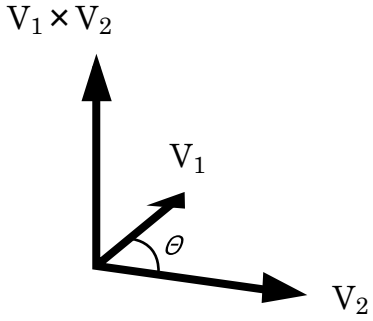
$$(x_1, y_1, z_1) \cdot (x_2, y_2, z_2) = x_1x_2 + y_1y_2 + z_1z_2 = |V_1|\,|V_2| \cos\theta$$

※ベクトルの内積の結果はスカラーである。

また、外積　$V_1 \times V_2$　は、以下の式で求められる。

$$(x_1, y_1, z_1) \times (x_2, y_2, z_2) = (y_1z_2 - z_1y_2,\ z_1x_2 - x_1z_2,\ x_1y_2 - y_1x_2)$$

$$= |V_1|\,|V_2|(\sin\theta)\boldsymbol{e}$$



$\boldsymbol{e}$ は、V_1 および V_2 に直角に交わる（直交する）単位ベクトル。

※ベクトルの外積の結果はベクトルである。

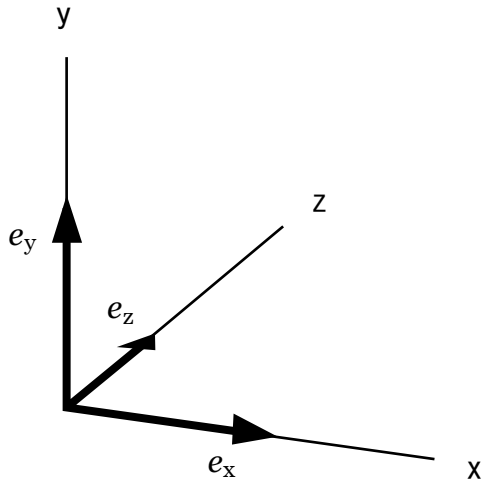
正規直交系と外積

$\boldsymbol{e}_x\,\boldsymbol{e}_y\,\boldsymbol{e}_z$ をそれぞれ、ｘｙｚ軸に沿う単位ベクトルとしたとき、以下の関係がある。

$$\boldsymbol{e}_x \times \boldsymbol{e}_y = \boldsymbol{e}_z$$

$$\boldsymbol{e}_y \times \boldsymbol{e}_z = \boldsymbol{e}_x$$

$$\boldsymbol{e}_z \times \boldsymbol{e}_x = \boldsymbol{e}_y$$



正規直交系