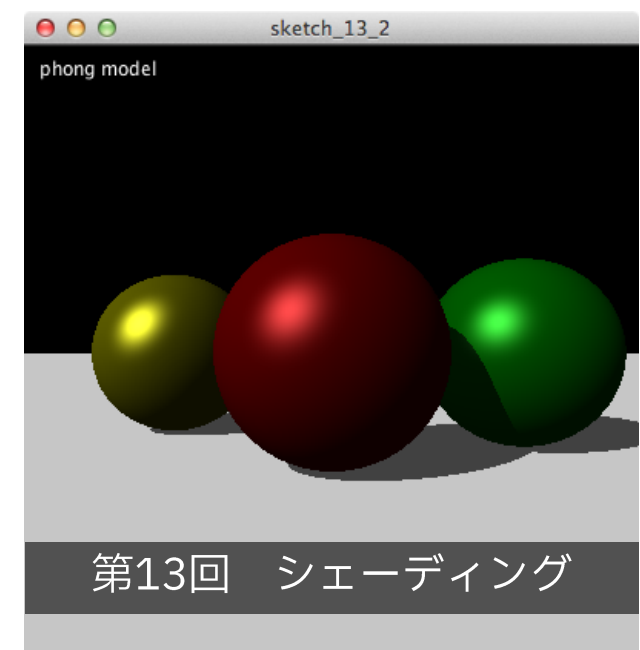
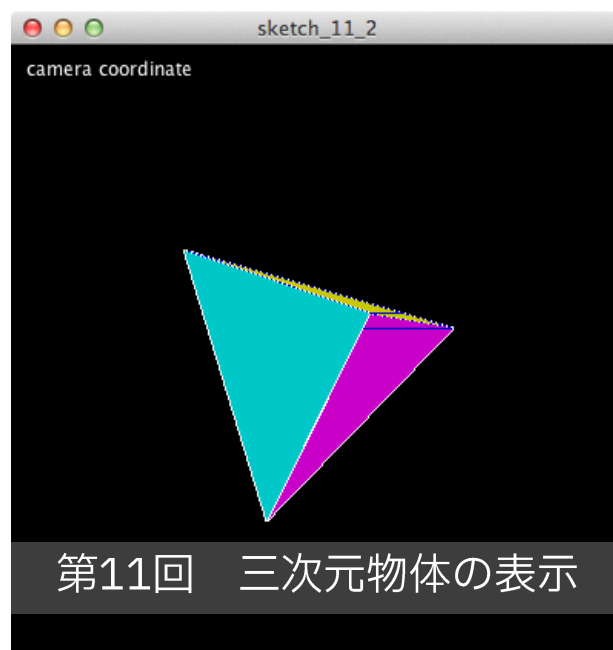
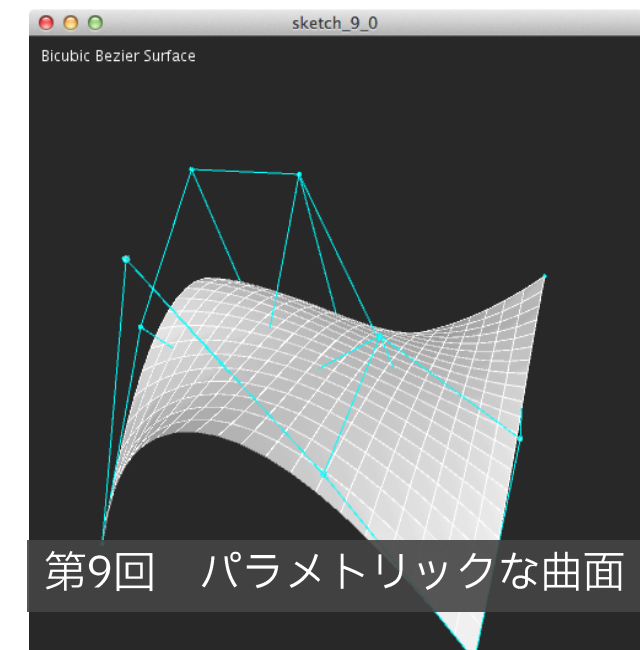
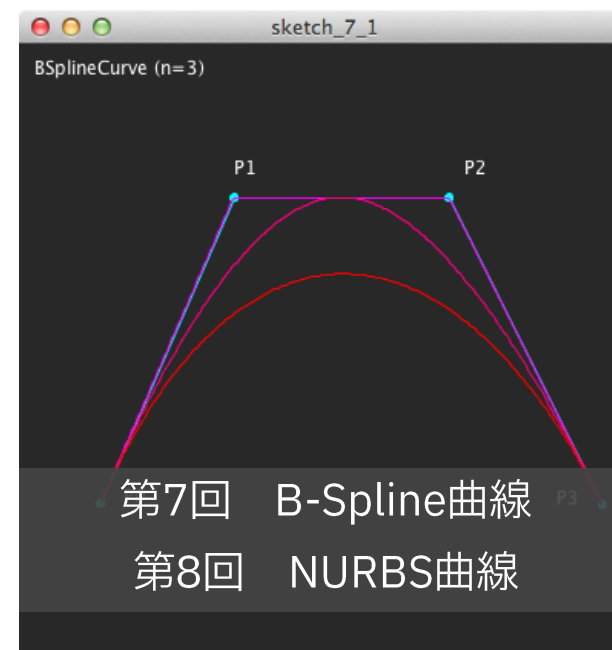
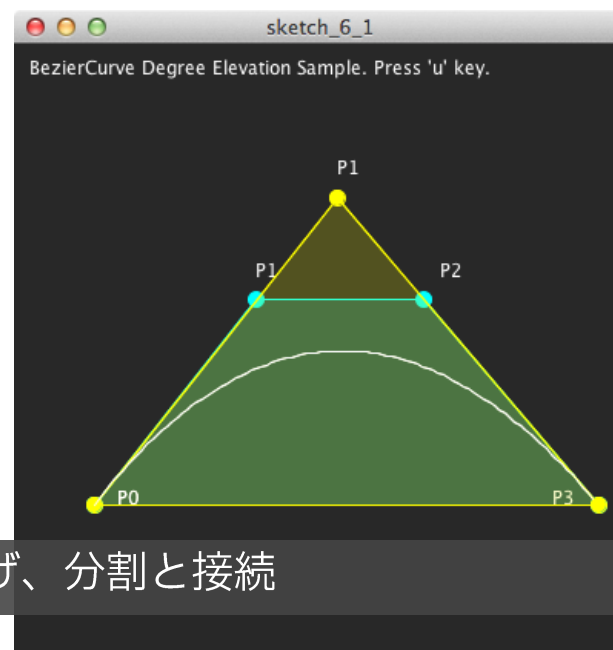
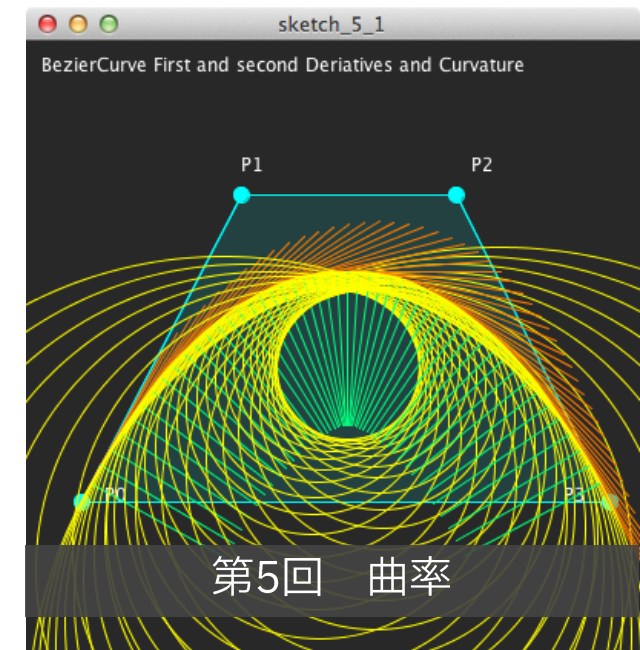
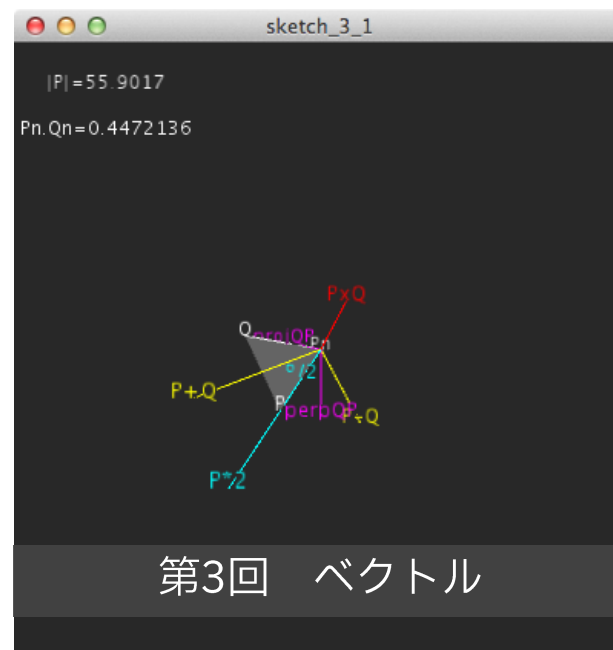
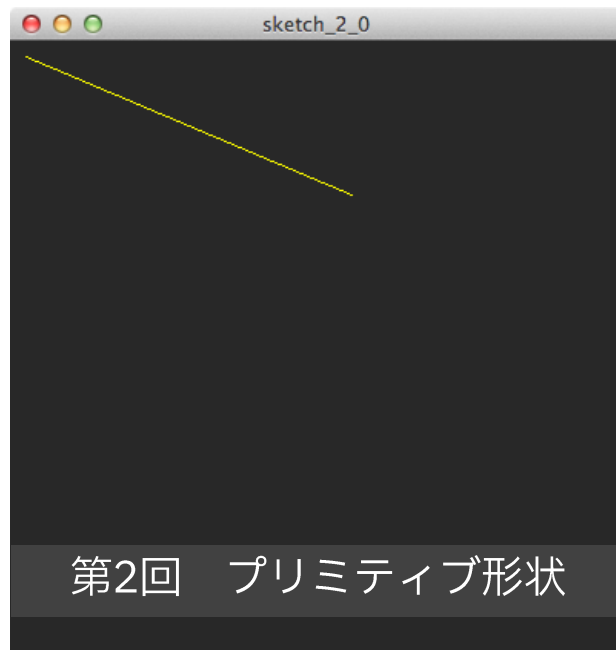


CGとCADの数理

GEOMETRIC MODELING AND COMPUTER GRAPHICS

第03回 ベクトル

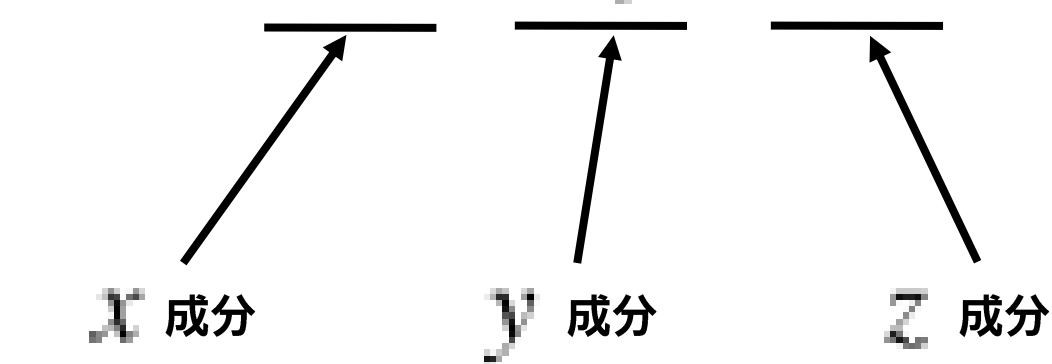


SOLからsketch_3_0.zipをダウンロードしてください。

ベクトル

Vector

デカルト座標系中の三次元ベクトル V は、以下のように表すことにします。

$$V = (V_x, V_y, V_z)$$


x 成分 y 成分 z 成分

デカルト座標系で示される位置 x 、 y 、 z のユニット（固まり）

数学的な厳密な定義とは離れるかもしれませんが、差し当たり講義ではベクトルは、[三次元空間中の座標を示す三つの実数の固まり](#)だと考えて頂いて構いません。また、中高時代にベクトルを学んだ方はそのイメージで構いません。

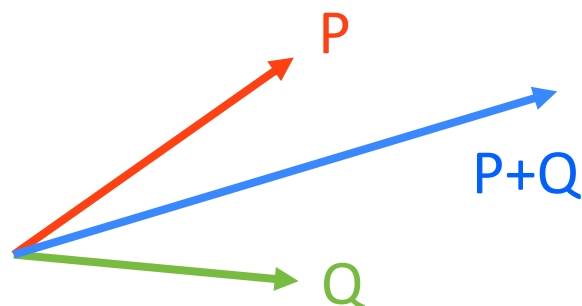
中高でベクトルを勉強していない方もいらっしゃるかと思いますので、簡単なことのように思われるかもしれませんが、今回はベクトルの四則演算とその応用を中心に進めさせていただきます。

ベクトルの四則演算

The four vector operations: addition, subtraction, multiplication and division

足し算

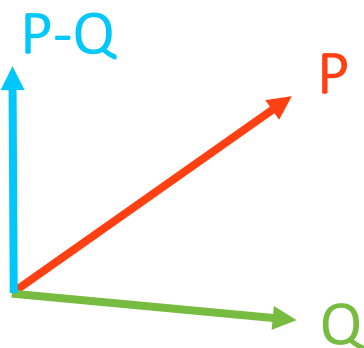
Addition



```
void add( CVector Q, CVector result ) {  
    result.x = x + Q.x;  
    result.y = y + Q.y;  
    result.z = z + Q.z;  
}
```

引き算

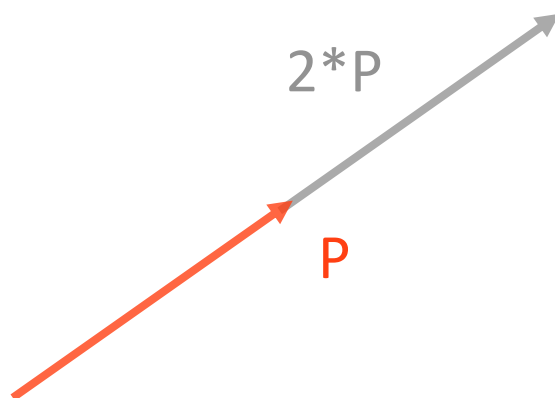
Subtraction



```
void sub( CVector Q, CVector result ) {  
    result.x = x - Q.x;  
    result.y = y - Q.y;  
    result.z = z - Q.z;  
}
```

スカラー乗算

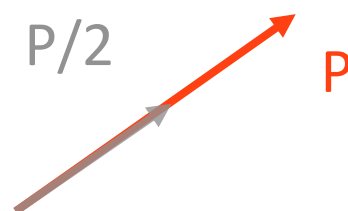
Multiplication



```
void mult( float n, CVector result ) {  
    result.x = x * n;  
    result.y = y * n;  
    result.z = z * n;  
}
```

スカラー除算

Division



```
void div( float n, CVector result ) {  
    result.x = x / n;  
    result.y = y / n;  
    result.z = z / n;  
}
```

ベクトルの四則演算の描画

`v0.add(v1, v2);` $// \quad V_2 \leftarrow V_0 + V_1$

`v0.sub(v1, v3);` $// \quad V_3 \leftarrow V_0 - V_1$

`v0.mult(2, v4);` $// \quad V_4 \leftarrow 2 \cdot V_0$

`v0.div(2, v5);` $// \quad V_5 \leftarrow \frac{V_0}{2}$

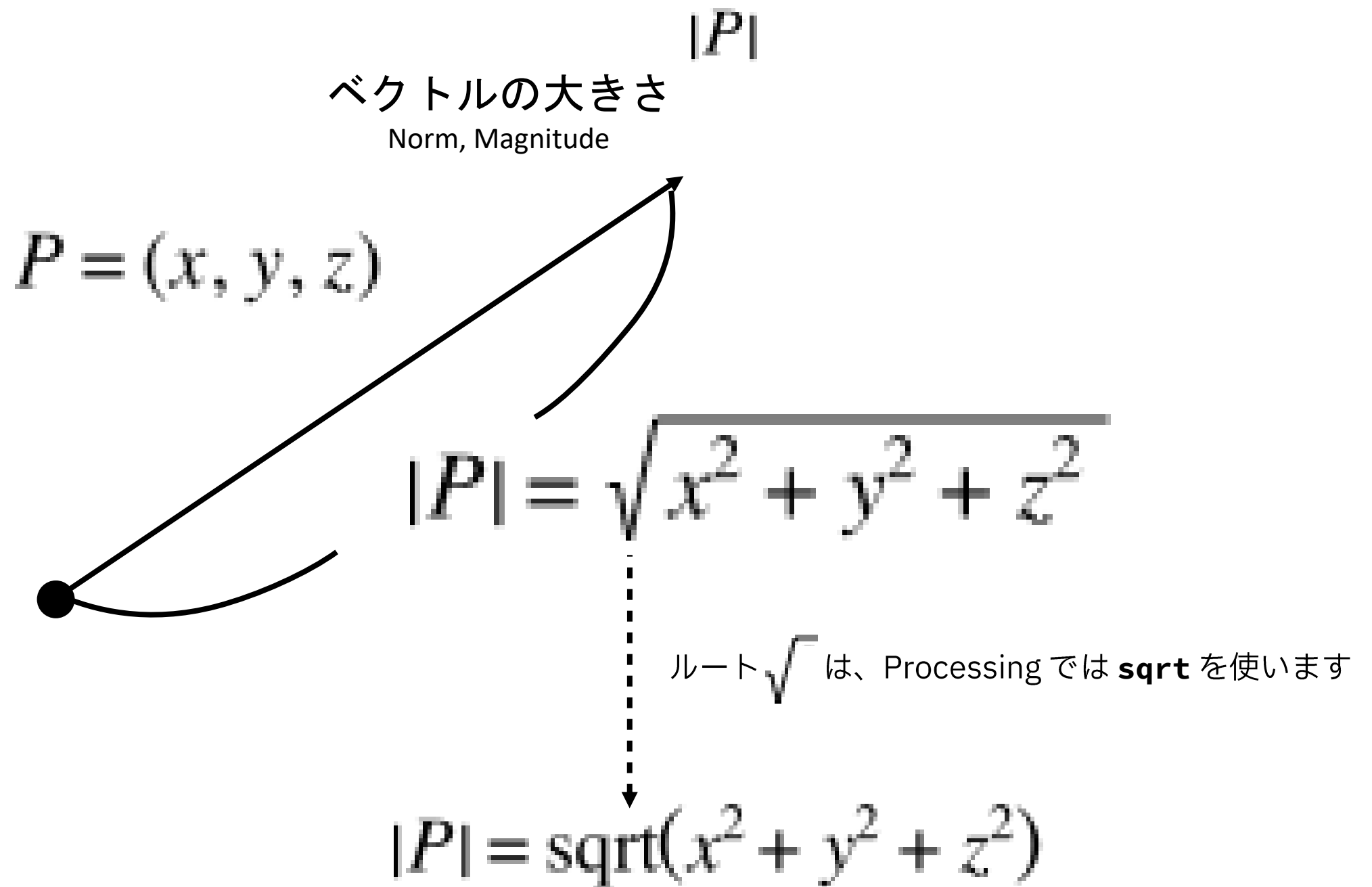
…省略

`v2.draw("v0+v1", color(0, 0, 255));`

`v3.draw("v0-v1", color(0, 255, 255));`

`v4.draw("v0*2", color(255, 255, 255));`

`v5.draw("v0/2", color(255, 255, 255));`



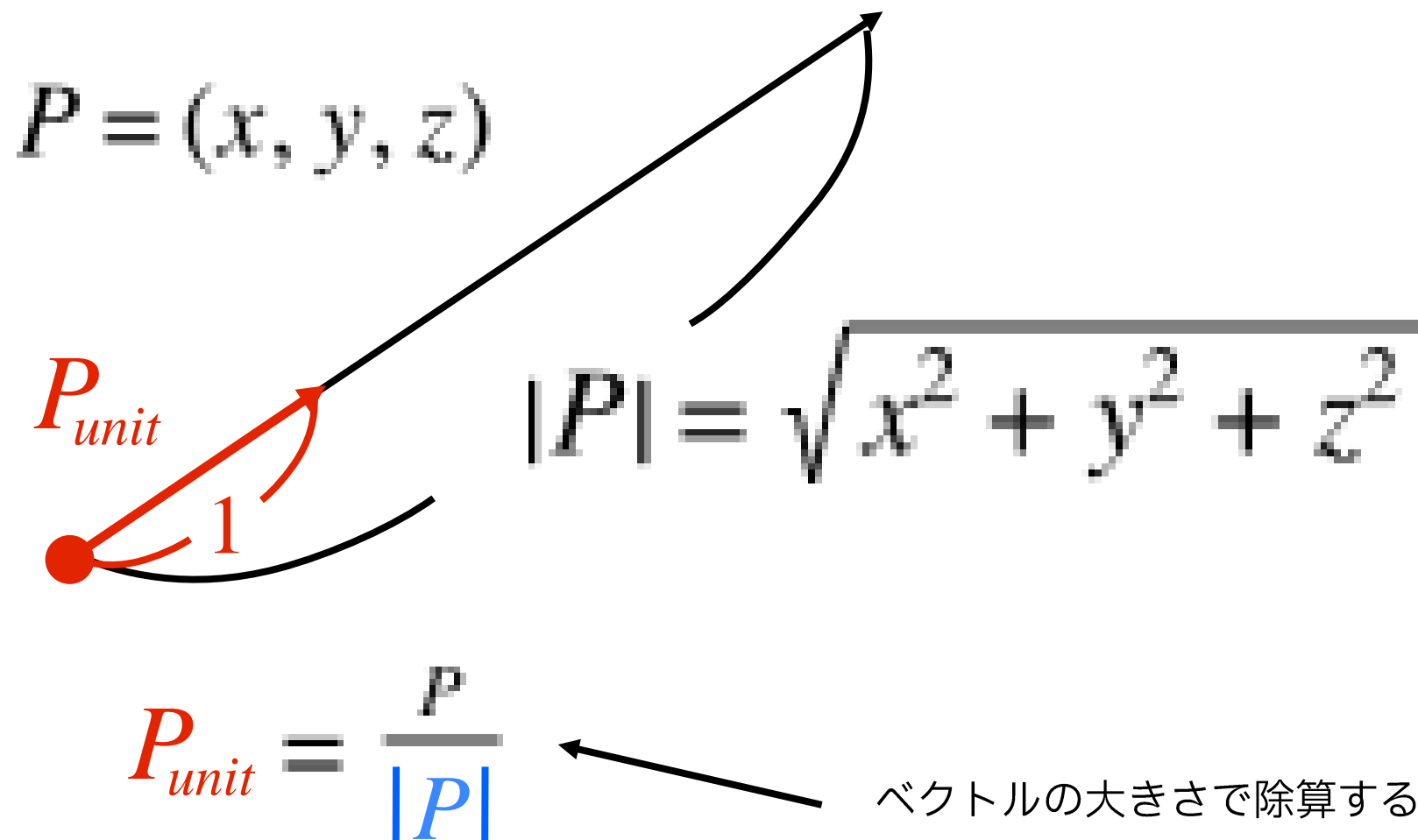
$$= \text{sqrt}(x * x + y * y + z * z)$$

```
float mag() {  
    return (float)sqrt( x * x + y * y + z * z );  
}
```

前回の直線の長さの計算と同様です。ベクトルの大きさは、ベクトルの長さと言ったり、ノルムと言ったりします。

単位ベクトルと正規化

Unit vector and normalization



```
void normalize(CVector result) {  
    float m = mag();  
    if ( m != 0 ) div(m, result);  
}
```

大きさが **1** のベクトルのことを「単位長のベクトル」、あるいは「**単位ベクトル (Unit Vector)**」と言います。あるベクトルを単位ベクトルにする演算を「**正規化 (normalize)**」と言い、CGでは頻繁に用いられます。

単位ベクトルの描画

```
v0.normalize(v6); //  $V_6 \leftarrow \frac{v_0}{|v_0|}$ 
```

```
v1.normalize(v7); //  $V_7 \leftarrow \frac{v_1}{|v_1|}$ 
```

...省略

```
v6.draw("v0/|v0|", color(0,255,255));
```

ベクトルの内積

Dot Product

$$P \cdot Q = \sum_{i=0}^N P_i Q_i$$

2次元の場合 $P \cdot Q = P_x Q_x + P_y Q_y$

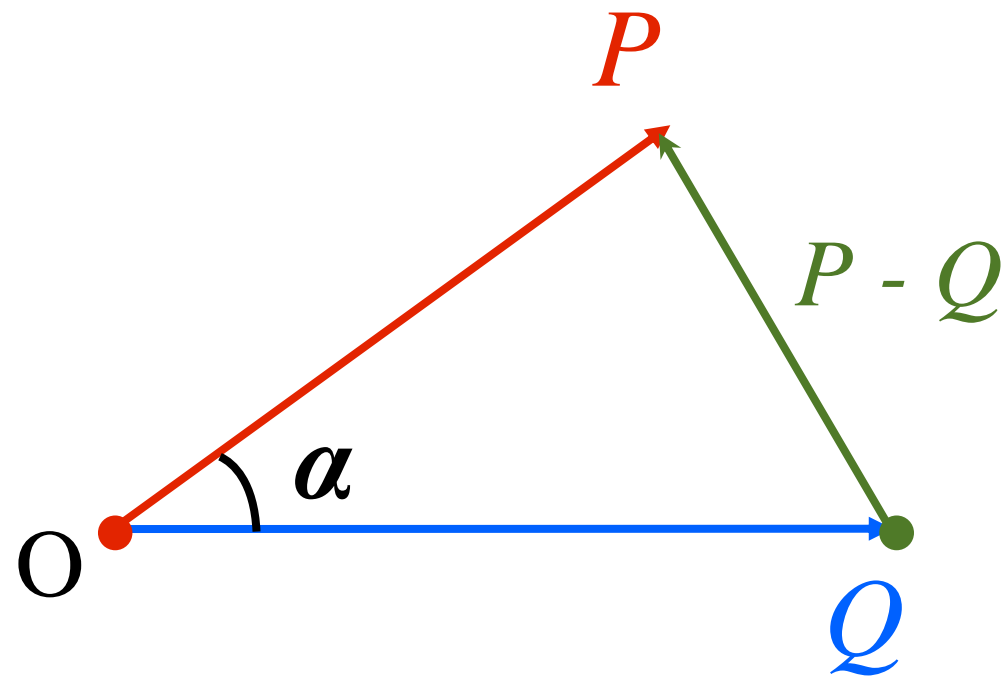
3次元の場合 $P \cdot Q = P_x Q_x + P_y Q_y + P_z Q_z$

```
float dot( CVector v ) {  
    return x*v.x + y*v.y + z*v.z;  
}
```

内積の計算自体は難しくありません。それぞれの成分の積の和（スカラー）です。CGでは内積の計算がたくさん登場します。その理由の一端を示すため、少し時間を割いて内積についてもう少し掘り下げてみたいと思います。

ベクトルの類似度 1

similarity between 2 vectors



左図を考えます。原点 **O** と点 **P**、点 **Q** があり、線分 **OP** と線分 **OQ** の成す角度を α とします。

この状況で、**余弦定理**を適用すると、以下の式が導けます。

$$|P - Q|^2 = |P|^2 + |Q|^2 - 2|P||Q|\cos\alpha$$

左辺を展開すると、

$$|P - Q|^2 = |P|^2 + |Q|^2 - 2P \cdot Q$$

なので、

$$-2P \cdot Q = -2|P||Q|\cos\alpha$$

さらに整理して、

$$P \cdot Q = |P||Q|\cos\alpha$$

$$|P| = 1, |Q| = 1$$

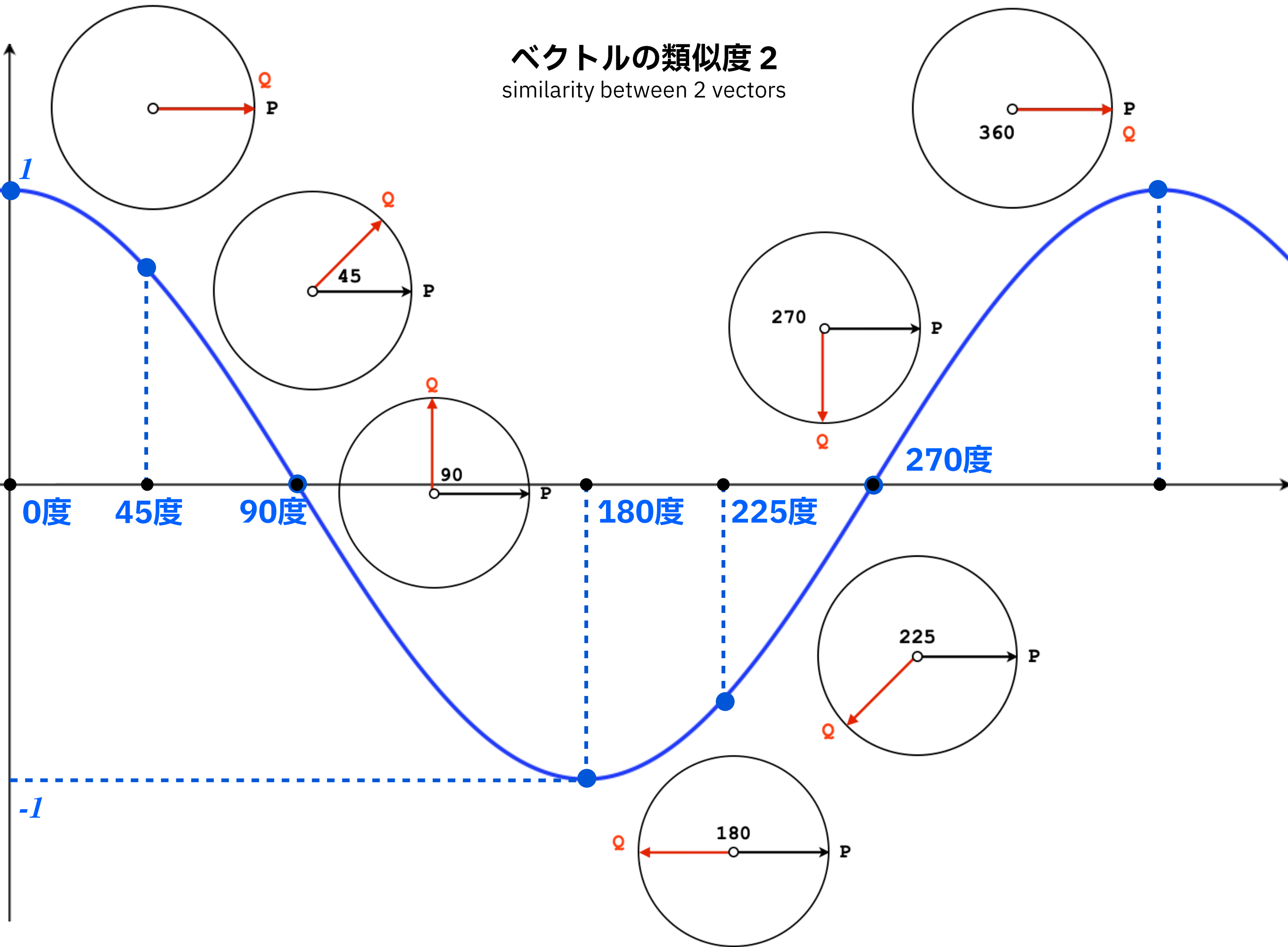
つまり、**P** も **Q** も単位ベクトルだと仮定します。すると内積は、以下の式になります。

$$P \cdot Q = \cos\alpha$$

という計算になりました。これだけだと何の役に立つのか分かりにくいので話を進めます。

ベクトルの類似度 2

similarity between 2 vectors



ベクトルの類似度


`v0.normalize(v6);` // $V_6 \leftarrow \frac{v_0}{|v_0|} \rightarrow |V_6|=1$

`v1.normalize(v7);` // $V_7 \leftarrow \frac{v_1}{|v_1|} \rightarrow |V_7|=1$

`fill(255);`

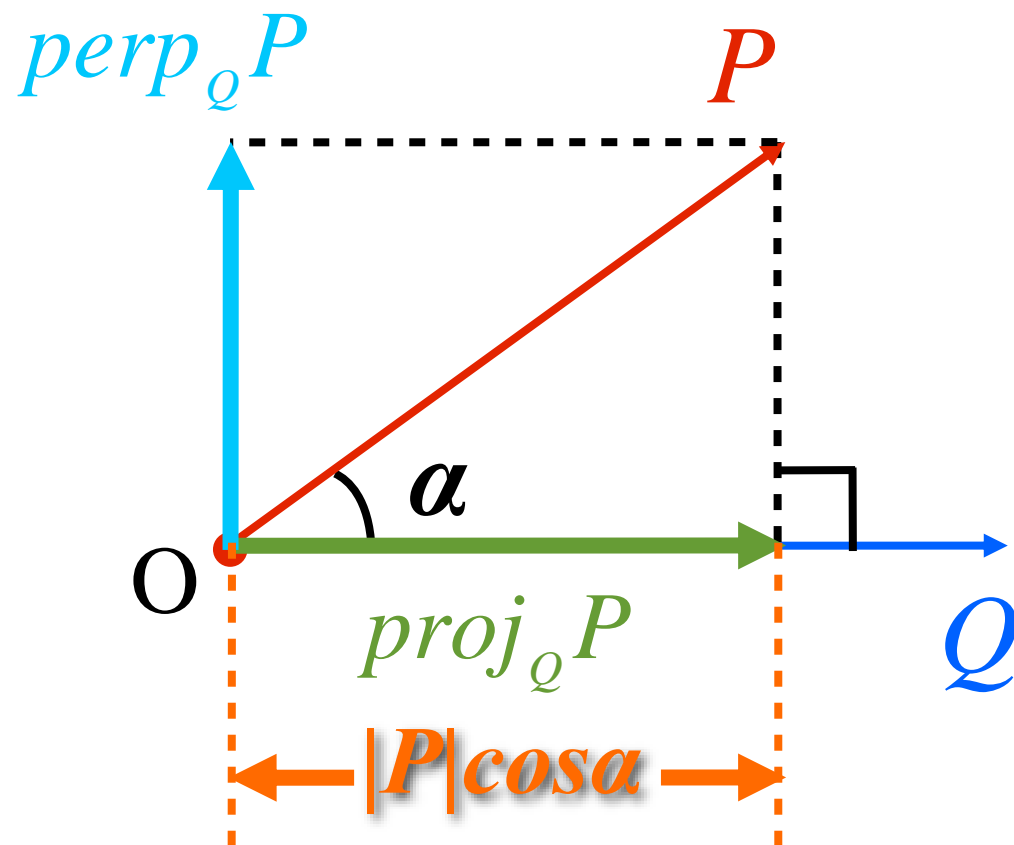
`text("|P|="+v0.mag(), 30, 30);`

`text("P/|P| dot Q/|Q|="+v6.dot(v7), 30, 50);`


$$V_6 \cdot V_7$$

射影ベクトル

Projection



$$\mathbf{P} \cdot \mathbf{Q} = |\mathbf{P}| |\mathbf{Q}| \cos\alpha$$

両辺を $|\mathbf{Q}|$ で割ります。

$$\frac{\mathbf{P} \cdot \mathbf{Q}}{|\mathbf{Q}|} = \frac{|\mathbf{P}| |\mathbf{Q}| \cos\alpha}{|\mathbf{Q}|}$$

$$\frac{\mathbf{P} \cdot \mathbf{Q}}{|\mathbf{Q}|} = |\mathbf{P}| \cos\alpha$$

再び左図を考えます。しばしば、 \mathbf{P} を別のベクトル \mathbf{Q} に対して、平行な成分 $proj_Q P$ と垂直な成分 $perp_Q P$ に分解する状況に出くわします。

$$\mathbf{P} = proj_Q P + perp_Q P$$

$proj_Q P$ は \mathbf{Q} と平行で、大きさは $|\mathbf{P}| \cos\alpha$ です。

$$proj_Q P = |\mathbf{P}| \cos\alpha \cdot \frac{\mathbf{Q}}{|\mathbf{Q}|}$$

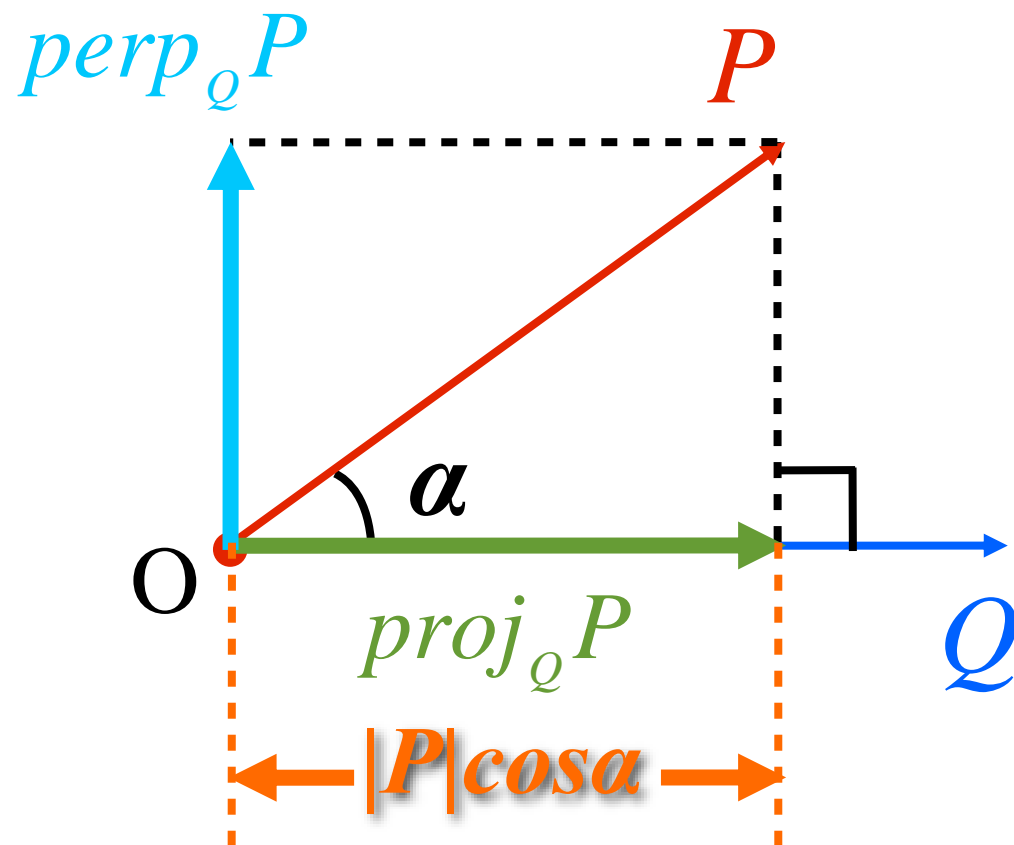
\mathbf{Q} の単位ベクトル

$$= \frac{\mathbf{P} \cdot \mathbf{Q}}{|\mathbf{Q}|^2} \mathbf{Q}$$

$$perp_Q P = \mathbf{P} - proj_Q P$$

射影ベクトル

Projection



$$proj_Q P = \frac{P \cdot Q}{|Q|^2} Q$$

```
void proj(CVector Q, CVector result) {  
    float d = dot(Q);  
    float Q2 = Q.x*Q.x + Q.y*Q.y + Q.z*Q.z;  
    Q.mult(d/Q2, result);  
}
```

$$perp_Q P = P - proj_Q P$$

```
void perp(CVector Q, CVector result) {  
    proj(Q, result);  
    sub(result, result);  
}
```

射影ベクトルの描画

`v0.proj(v1, v8); // $V_8 \leftarrow \text{proj}_Q P$`

`v0.perp(v1, v9); // $V_9 \leftarrow \text{perp}_Q P$`

…省略

`v8.draw("projqP", color(255, 0, 255));`
`v9.draw("perpqP", color(255, 0, 255));`

ベクトルの外積

Cross Product

$$P = (P_x, P_y, P_z)$$

$$Q = (Q_x, Q_y, Q_z)$$

という2つのベクトルの外積 $P \times Q$ は、その二つのベクトルに対し垂直なベクトルを計算します。

$$P \times Q = (P_y Q_z - P_z Q_y, P_z Q_x - P_x Q_z, P_x Q_y - P_y Q_x)$$

```
CVector cross( CVector v ) {  
    CVector r = new CVector();  
    r.x = y * v.z - v.y * z;  
    r.y = z * v.x - v.z * x;  
    r.z = x * v.y - v.x * y;  
    return r;  
}
```

興味のある方は、SOLからサンプルプログラムをダウンロードして下さい。