Digital 3D Anthropometry

3. Point Class

Sungmin Kim

SEOUL NATIONAL UNIVERSITY

TPoint2D

❖ Class 설계

- 정의
  - ✓ 2차원 공간상에서의 한 점을 정의하는 클래스
- 멤버 함수
  - ✓ 두 점 간의 거리 구하기
  - ✓ 한점에서 다른 점으로의 방향 구하기
  - ✓ 벡터의 절대 각도 구하기
  - ✓ 세 점 사이의 사잇각 구하기
  - ✓ 두 벡터의 외적 구하기
  - ✓ 한점이 삼각형 안에 있는지 판단하기
  - ✓ 한점을 다른 점을 중심으로 일정 각도 회전하기
- 연산자
  - ✓ 사칙연산
  - ✓ 벡터의 내적

TPoint2D

❖ Class 정의

▪ 기본형 정의

✓ 생성자, 복제생성자, 파괴자

TPoint2D.h

```
class TPoint2D
{
public:
        TPoint2D();
        TPoint2D(TPoint2D&);
        ~TPoint2D();

        float x,y;
};
```

TPoint2D.cpp

```
TPoint2D::TPoint2D()
{
x=y=0;
}

TPoint2D::TPoint2D(TPoint2D &P)
{
x=P.x;
y=P.y;
}

TPoint2D::~TPoint2D()
{
}
```


TPoint2D

❖ Class 정의

▪ 추가 기능 정의

✓ 점으로 할 수 있는 다양한 계산함수들을 클래스에 포함

- Prototype을 TPoint2D.h 에 정의
- 본체는 TPoint2D.cpp 에 정의
- Polymorphism을 이용해서 같은 작업을 하는 함수를 여러 버젼으로 생성 가능
- 필요한 함수가 생길 때 마다 계속 추가하여 클래스를 다듬기

```
float     Distance(TPoint2D&);                    // 두 점간의 거리
TPoint2D  Direction(TPoint2D&);                   // 한 점에서 다른 점의로의 방향
float     Angle();                                // 벡터의 절대 각도
float     Angle(TPoint2D&,TPoint2D&);             // 세 점사이의 사잇각
float     CrossProduct(TPoint2D&);                // 두 벡터의 외적 1
float     CrossProduct(TPoint2D&,TPoint2D&);      // 두 벡터의 외적 2
bool      IsInside(TPoint2D&,TPoint2D&,TPoint2D&); // 한 점이 삼각형 안에 있는지 ?
TPoint2D  Rotated(TPoint2D&,float);               // 한점을 다른 점을 중심으로 회전
```

TPoint2D

❖ Class 정의

▪ 추가 기능 정의

```
float           TPoint2D::Distance(TPoint2D &P)
{
return sqrt((x-P.x)*(x-P.x)+(y-P.y)*(y-P.y));
}
```

sqrt 함수를 사용하려면 TPoint2D.h 에서 #include <math.h> 를 해야 함!

```
TPoint2D        TPoint2D::Direction(TPoint2D &P)
{
float           l=Distance(P);
TPoint2D        V;
if (l==0){
        V.x=V.y=0;
        }
else{
        V.x=(P.x-x)/l;
        V.y=(P.y-y)/l;
        }
return V;
}
```

0으로 나누는 등의 기본 검사를 철저히 하는 것이 중요함

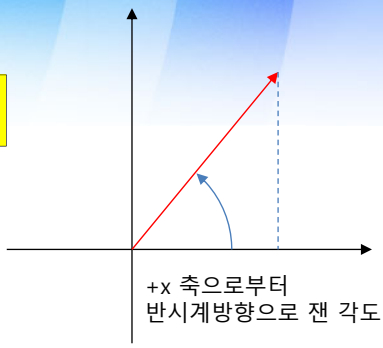
TPoint2D

❖ Class 정의

▪ 추가 기능 정의

```
float           TPoint2D::Angle()
{
Normalize();
if (!x){
        if (y>0) return M_PI/2;
                if (!y) return 0;
                if (y<0) return 3*M_PI/2;
                }
if (!y){
        if (x>0) return 0;
                if (!x) return 0;
                if (x<0) return M_PI;
                }
if (x>=1) return 0;
if (x<=-1) return M_PI;
float  a=acos(fabs(x));
if (x>0 && y>0) return a;
if (x<0 && y>0) return M_PI-a;
if (x<0 && y<0) return M_PI+a;
if (x>0 && y<0) return 2*M_PI-a;
return 0;
}
```

모든 경우의 수를 고려해야 함
각도는 Radian 값으로 구함



+x 축으로부터
반시계방향으로 잰 각도

```
void            TPoint2D::Normalize()
{
float l=sqrt(x*x+y*y);
if (l){
        x/=l;
        y/=l;
        }
}
```

단위벡터로 만드는 함수 추가

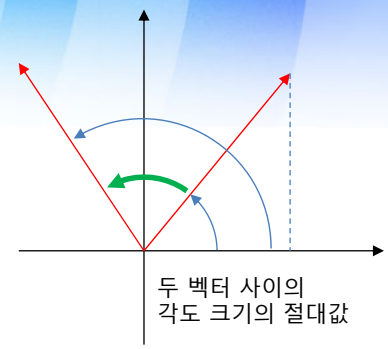
TPoint2D

❖ Class 정의

▪ 추가 기능 정의

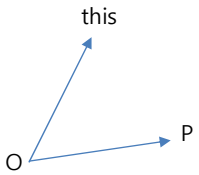
✓ 세 점 사이의 각을 구하기

```
float          TPoint2D::Angle(TPoint2D &P1,TPoint2D &P2)
{
P1=Direction(P1);
P2=Direction(P2);
return fabs(P1.Angle()-P2.Angle());
}
```

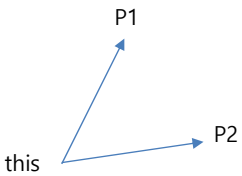


✓ 외적 구하기

```
float          TPoint2D::CrossProduct(TPoint2D &P)
{
return (x*P.y-y*P.x);
}
```




```
float          TPoint2D::CrossProduct(TPoint2D &P1,TPoint2D &P2)
{
P1.x=P1.x-x;
P1.y=P1.y-y;
P2.x=P2.x-x;
P2.y=P2.y-y;
return P1.x*P2.y-P1.y*P2.x;
}
```



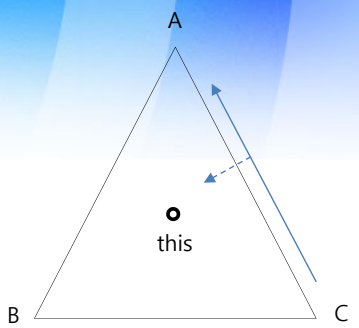
TPoint2D

❖ Class 정의

▪ 추가 기능 정의

✓ 한 점이 삼각형 안에 있는지 구하기

```
bool           TPoint2D::IsInside(TPoint2D &A,TPoint2D &B,TPoint2D &C)
{
return (IsLeft(A,B) && IsLeft(B,C) && IsLeft(C,A));
}
```




```
bool           TPoint2D::IsLeft(TPoint2D &P,TPoint2D &Q)
{
return (SignedArea(P,Q)>=0);
}

float          TPoint2D::SignedArea(TPoint2D &P,TPoint2D &Q)
{
return ((P.x*Q.y-P.y*Q.x)+(P.y*x-P.x*y)+(Q.x*y-x*Q.y))*0.5f;
}
```

1. 한점이 두점으로 이루어지는 벡터의 왼쪽에 있는지 ?

2. 세 점의 좌표로 구한 면적의 부호가 어떻게 되는지 ?

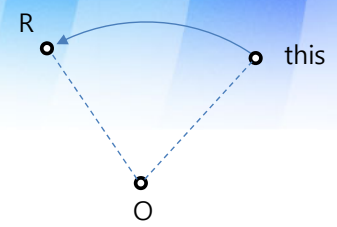
TPoint2D

❖ Class 정의

▪ 추가 기능 정의

✓ 한 점을 다른 점을 중심으로 회전하기

– 회전방향은 반시계 방향



$$\begin{pmatrix} R_x \\ R_y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} P_x \\ P_y \end{pmatrix}$$

```
TPoint2D        TPoint2D::Rotated(TPoint2D &O,float ang)
{
TPoint2D R;
R.x=O.x+(x-O.x)*cos(ang)-(y-O.y)*sin(ang);
R.y=O.y+(x-O.x)*sin(ang)+(y-O.y)*cos(ang);
return R;
}
```

```
TPoint2D        TPoint2D::Rotated(TPoint2D &O,float cc,float ss)
{
TPoint2D R;
R.x=O.x+(x-O.x)*cc-(y-O.y)*ss;
R.y=O.y+(x-O.x)*ss+(y-O.y)*cc;
return R;
}
```

수 많은 점을 회전해야 할 경우에는 ?

TPoint2D

❖ Class 정의

▪ 연산자 오버로딩

✓ 사칙연산

✓ 특수 연산

– 내적 계산

```
TPoint2D    operator+(const TPoint2D& N);
TPoint2D    operator-(const TPoint2D& N);
TPoint2D    operator*(float n);
TPoint2D    operator/(float n);

TPoint2D&   operator+=(const TPoint2D& N);
TPoint2D&   operator-=(const TPoint2D& N);
TPoint2D&   operator*=(float n);
TPoint2D&   operator/=(float n);

bool        operator!=(const TPoint2D &N);
bool        operator==(const TPoint2D &N);
float       operator*(const TPoint2D& N);
```

일반 사칙 연산

C++ 스타일 연산자

특수 연산자, 내적 (Polymorphism)

TPoint2D

❖ Class 정의

- 연산자 오버로딩

```
TPoint2D        TPoint2D::operator+(const TPoint2D& N)
{
return TPoint2D(x+N.x,y+N.y);
}

TPoint2D        TPoint2D::operator-(const TPoint2D& N)
{
return TPoint2D(x-N.x,y-N.y);
}

TPoint2D        TPoint2D::operator*(const float n)
{
return TPoint2D(x*n,y*n);
}

TPoint2D        TPoint2D::operator/(float n)
{
if (n) return TPoint2D(x/n,y/n);
return TPoint2D(x,y);
}
```

```
TPoint2D::TPoint2D(float X,float Y)
{
x=X;
y=Y;
}
```

```
TPoint2D&       TPoint2D::operator+=(const TPoint2D& N)
{
x+=N.x;
y+=N.y;
return *this;
}

TPoint2D&       TPoint2D::operator-=(const TPoint2D& N)
{
x-=N.x;
y-=N.y;
return *this;
}

TPoint2D&       TPoint2D::operator*=(const float n)
{
x*=n;
y*=n;
return *this;
}

TPoint2D&       TPoint2D::operator/=(const float n)
{
x/=n;
y/=n;
return *this;
}
```


TPoint2D

❖ Class 정의

- 연산자 오버로딩

```
bool            TPoint2D::operator!=(const TPoint2D &N)
{
if (x!=N.x || y!=N.y) return true;
return false;
}

bool            TPoint2D::operator==(const TPoint2D &N)
{
if (x==N.x && y==N.y) return true;
return false;
}

float           TPoint2D::operator*(const TPoint2D& N)
{
return x*N.x+y*N.y;
}
```

TPoint3D

❖ Class 설계

- **정의**
  - ✓ 3차원 공간상에서 한 점을 정의하는 클래스
- **멤버함수**
  - ✓ 2D 와 거의 유사하지만 몇 가지 함수를 추가
    - – 한 점을 다른 점을 중심으로 한 벡터에 수직한 방향으로 회전하기 등

TPoint3D

❖ Class 정의

- **기본형 정의**
  - ✓ 생성자, 복제생성자, 파괴자

TPoint3D.h

```
class    TPoint3D
{
public:

        TPoint3D();
        TPoint3D(float,float,float);
        TPoint3D(TPoint3D&);
        ~TPoint3D();

        float    x,y,z;
};
```

TPoint3D.cpp

```
TPoint3D::TPoint3D()
{
x=y=z=0;
}

TPoint3D::TPoint3D(float X,float Y,float Z)
{
x=X;
y=Y;
z=Z;
}

TPoint3D::TPoint3D(TPoint3D &P)
{
x=P.x;
y=P.y;
z=P.z;
}

TPoint3D::~TPoint3D()
{
}
```

TPoint3D

❖ Class 정의

▪ 추가 기능 정의

✓ 점으로 할 수 있는 다양한 계산함수들을 클래스에 포함

– TPoint2D 와 유사한 기능을 3차원으로 확장

```
float       Distance(TPoint3D&);                              // 두 점 간의 거리
TPoint3D    Direction(TPoint3D&);                             // 한 점에서 다른 점으로의 방향
float       Angle(TPoint3D&,TPoint3D&);                       // 세 점 사이의 사잇각
TPoint3D    CrossProduct(TPoint3D&,TPoint3D&);                // 두 벡터의 외적
TPoint3D    Rotated(TPoint3D&,float,float,float);             // 원점에 대한 x,y,z 축 회전
TPoint3D    Rotated(TPoint3D&,float,float,float,float,float,float);  // 많은 점을 회전
TPoint3D    RotatedAbout(TPoint3D&,TPoint3D&,float);          // 원점을 중심으로 한 벡터에 수직으로 회전
```


TPoint3D

❖ Class 정의

▪ 추가 기능 정의

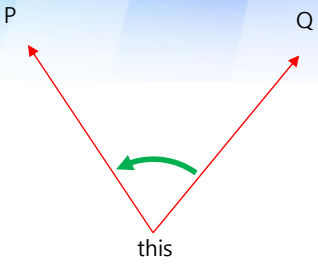
```
float           TPoint3D::Distance(TPoint3D &N)
{
return sqrt((x-N.x)*(x-N.x)+(y-N.y)*(y-N.y)+(z-N.z)*(z-N.z));
}

TPoint3D        TPoint3D::Direction(TPoint3D &N)
{
float   X,Y,Z;
float           l=sqrt((x-N.x)*(x-N.x)+(y-N.y)*(y-N.y)+(z-N.z)*(z-N.z));
X=N.x-x;
Y=N.y-y;
Z=N.z-z;
if (l>0){
        X/=l;
        Y/=l;
        Z/=l;
        }
return TPoint3D(X,Y,Z);
}
```

❖ Class 정의

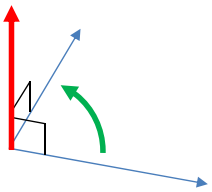
- 추가 기능 정의

```
float           TPoint3D::Angle(TPoint3D &P,TPoint3D &Q)
{
float  l=Distance(P)*Distance(Q);
if (!l) return 0;
float  ang=((P.x-x)*(Q.x-x)+(P.y-y)*(Q.y-y)+(P.z-z)*(Q.z-z))/l;
if (ang>=1) return 0;
if (ang<=-1) return M_PI;
return acos(ang);
}
```



두 벡터 사이의 각도

```
TPoint3D          TPoint3D::CrossProduct(TPoint3D &P,TPoint3D &Q)
{
TPoint3D R;
R.x=(P.z-z)*(Q.y-y)-(P.y-y)*(Q.z-z);
R.y=(P.x-x)*(Q.z-z)-(P.z-z)*(Q.x-x);
R.z=(P.y-y)*(Q.x-x)-(P.x-x)*(Q.y-y);
return R;
}
```



벡터의 외적

❖ Class 정의

- 추가 기능 정의

```
TPoint3D     TPoint3D::Rotated(TPoint3D &O,float rx,float ry,float rz)
{
TPoint3D R=TPoint3D(x-O.x,y-O.y,z-O.z);
float  X,Y,Z;
float  cc,ss;
if (rx){
        cc=cos(rx); ss=sin(rx);
        Y=R.y*cc-R.z*ss;
        Z=R.y*ss+R.z*cc;
        R.y=Y; R.z=Z;
        }
if (ry){
        cc=cos(ry); ss=sin(ry);
        X=R.x*cc-R.z*ss;
        Z=R.x*ss+R.z*cc;
        R.x=X; R.z=Z;
        }
if (rz){
        cc=cos(rz); ss=sin(rz);
        X=R.x*cc-R.y*ss;
        Y=R.x*ss+R.y*cc;
        R.x=X; R.y=Y;
        }
return TPoint3D(R.x+O.x,R.y+O.y,R.z+O.z);
}
```

수 많은 점을 회전해야 할 경우

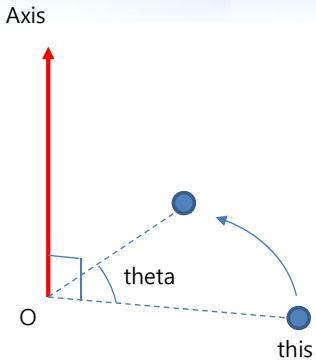
```
TPoint3D     TPoint3D::Rotated(TPoint3D &O,float cx,float sx,float cy,
float sy,float cz,float sz)
{
TPoint3D R=TPoint3D(x-O.x,y-O.y,z-O.z);
float  X,Y,Z;
Y=R.y*cx-R.z*sx;
Z=R.y*sx+R.z*cx;
R.y=Y;
R.z=Z;
X=R.x*cy-R.z*sy;
Z=R.x*sy+R.z*cy;
R.x=X;
R.z=Z;
X=R.x*cz-R.y*sz;
Y=R.x*sz+R.y*cz;
R.x=X;
R.y=Y;
return TPoint3D(R.x+O.x,R.y+O.y,R.z+O.z);
}
```

TPoint3D

❖ Class 정의

- 추가 기능 정의

```
TPoint3D        TPoint3D::RotatedAbout(TPoint3D &O,TPoint3D &Axis,float theta)
{
float           ang1,ang2;
TPoint2D        p1=TPoint2D(Axis.x,Axis.z);
TPoint2D        p2=TPoint2D(sqrt(Axis.x*Axis.x+Axis.z*Axis.z),Axis.y);
ang1=p1.Angle();
ang2=p2.Angle()-M_PI/2;
TPoint3D        P=TPoint3D(x,y,z);
P=P.Rotated(O,0,-ang1,-ang2);
P=P.Rotated(O,0,theta,ang2);
P=P.Rotated(O,0,ang1,0);
return P;
}
```



TPoint3D

❖ Class 정의

- 연산자 오버로딩
  - ✓ 사칙연산
  - ✓ 특수 연산
    - – 벡터의 내적 계산

```
TPoint3D      operator+(const TPoint3D& N);
TPoint3D      operator-(const TPoint3D& N);
TPoint3D      operator*(float n);
TPoint3D      operator/(float n);

TPoint3D&     operator+=(const TPoint3D& N);
TPoint3D&     operator-=(const TPoint3D& N);
TPoint3D&     operator*=(float n);
TPoint3D&     operator/=(float n);

bool          operator==(TPoint3D& N);
bool          operator!=(TPoint3D& N);
float         operator*(const TPoint3D& N);
```

❖ Class 정의

▪ 연산자 오버로딩

```
TPoint3D            TPoint3D::operator+(const TPoint3D& N)
{
return TPoint3D(x+N.x,y+N.y,z+N.z);
}

TPoint3D            TPoint3D::operator-(const TPoint3D& N)
{
return TPoint3D(x-N.x,y-N.y,z-N.z);
}

TPoint3D            TPoint3D::operator*(float n)
{
return TPoint3D(x*n,y*n,z*n);
}

TPoint3D        TPoint3D::operator/(float n)
{
if (n) return TPoint3D(x/n,y/n,z/n);
return TPoint3D(x,y,z);
}

TPoint3D&       TPoint3D::operator+=(const TPoint3D& N)
{
x+=N.x;y+=N.y;z+=N.z;
return *this;
}

TPoint3D&       TPoint3D::operator-=(const TPoint3D& N)
{
x-=N.x;y-=N.y;z-=N.z;
return *this;
}
```

```
TPoint3D&       TPoint3D::operator*=(float v)
{
x*=v;y*=v;z*=v;
return *this;
}

TPoint3D&       TPoint3D::operator/=(float v)
{
if (v){
        x/=v;y/=v;z/=v;
        }
return *this;
}

bool            TPoint3D::operator==(TPoint3D& N)
{
if (x==N.x && y==N.y && z==N.z) return true;
return false;
}

bool            TPoint3D::operator!=(TPoint3D& N)
{
if (x==N.x && y==N.y && z==N.z) return false;
return true;
}

float           TPoint3D::operator*(const TPoint3D& N)
{
return x*N.x+y*N.y+z*N.z;
}
```