[ 게임 알고리즘 ]

**알고리즘 구현을 위한 수학지식**

인하대학교 미래인재개발원 문화콘텐츠프로그래밍

김예슬

**벡터**

- 크기와 방향을 가지고 있는 개념을 수학적으로 표현한 것

- 연산으로 덧셈, 뺄셈, 곱셈, 나눗셈, 외적, 내적 등이 존재

**행렬**

- 수 또는 다항식 등을 직사각형 모양으로 배열한 코드

- 연산으로 덧셈, 스칼라배, 곱셈, 전치 행렬 대각합, 행렬식 등이 존재

- 곱셈의 경우 일반적으로 교환법칙이 성립하지 않음

**벡터를 이용한 기본 연산 프로그램 코드**

VectorClass::VectorClass(float set1, float set2, float set3) : x(set1), y(set2), z(set3)

{ x = set1; y = set2; z = set3; }

VectorClass VectorClass::operator+(VectorClass & vec)

{

VectorClass r;

r.x = x + vec.x;

r.y = y + vec.y;

r.z = z + vec.z;

return r;

}

VectorClass VectorClass::operator-(VectorClass & vec)

{

VectorClass r;

r.x = x - vec.x;

r.y = y - vec.y;

r.z = z - vec.z;

return r;

}

VectorClass VectorClass::operator\*(float f)

{

VectorClass r;

r.x = x \* f;

r.y = y \* f;

r.z = z \* f;

return r;

}

VectorClass VectorClass::operator/(float f)

{

VectorClass r;

r.x = x / f;

r.y = y / f;

r.z = z / f;

return r;

}

float VectorClass::Dot(VectorClass & v1, VectorClass & v2)

{

return v1.x\*v2.x + v1.y\*v2.y + v1.z\*v2.z;

}

VectorClass VectorClass::Cross(VectorClass & v1, VectorClass & v2)

{

VectorClass r;

r.x = v1.y\*v2.z - v1.z\*v2.y;

r.y = v1.z\*v2.x - v1.x\*v2.z;

r.z = v1.x\*v2.y - v1.y\*v2.x;

return r;

}

float VectorClass::Length()

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

VectorClass VectorClass::Normalize()

{

VectorClass r;

float len = Length();

r.x = x / len;

r.y = y / len;

r.z = z / len;

return r;

}

**행렬을 이용한 기본 연산 프로그램 코드**

int cMatrix::Dimension()

{

return (int)colVec.size();

}

cMatrix cMatrix::Identity(int nDimension)

{

cMatrix matRet(nDimension);

for (int i = 0; i < nDimension; i++)

{

for (int j = 0; j < nDimension; j++)

matRet[i][j] = (i == j) ? 1 : 0;

}

return matRet;

}

cMatrix cMatrix::operator+(cMatrix & mat)

{

cMatrix r(Dimension());

for (int i = 0; i < Dimension(); i++)

{

for (int j = 0; j < Dimension(); j++)

{

r[i][j] = (\*this)[i][j] + mat[i][j];

}

}

return r;

}

cMatrix cMatrix::operator-(cMatrix & mat)

{

cMatrix r(Dimension());

for (int i = 0; i < Dimension(); i++)

{

for (int j = 0; j < Dimension(); j++)

{

r[i][j] = (\*this)[i][j] - mat[i][j];

}

}

return r;

}

cMatrix cMatrix::operator\*(cMatrix & mat)

{

cMatrix r(Dimension());

for (int i = 0; i < Dimension(); i++)

{

for (int j = 0; j < Dimension(); j++)

{

r[i][j] = 0.f;

for (int k = 0; k < Dimension(); k++)

{

r[i][j] = (\*this)[i][k] \* mat[k][j];

}

}

}

return r;

}

cMatrix cMatrix::operator\*(float f)

{

cMatrix r(Dimension());

for (int i = 0; i < Dimension(); i++)

{

for (int j = 0; j < Dimension(); j++)

{

r[i][j] = (\*this)[i][j] \* f;

}

}

return r;

}

cMatrix cMatrix::Inverse(OUT float & fDeterminant)

{

cMatrix r = cMatrix::Identity(Dimension());

fDeterminant = Determinant();

if (-dEpslion < fDeterminant && fDeterminant < dEpslion)

{

cout << "역행렬이 존재하지 x" << endl;

return r;

}

cMatrix adjoint = Adjoint();

r = adjoint \* (1 / fDeterminant);

return r;

}

float cMatrix::Determinant()

{

if (Dimension() == 1)

{

return (\*this)[0][0];

}

if (Dimension() == 2)

{

return (\*this)[0][0] \* (\*this)[1][1] - (\*this)[1][0] \* (\*this)[0][1];

}

float fDeterminant = 0.0f;

for (int i = 0; i < Dimension(); i++)

{

fDeterminant += ((\*this)[i][0] \* Cofactor(i, 0));

}

return 0;

}

cMatrix cMatrix::Adjoint()

{

cMatrix r(Dimension());

for (int i = 0; i < Dimension(); i++)

{

for (int j = 0; j < Dimension(); j++)

{

r[i][j] = Cofactor(j, i);

}

}

return r;

}

cMatrix cMatrix::Transpose()

{

cMatrix r(Dimension());

for (int i = 0; i < Dimension(); i++)

{

for (int j = 0; j < Dimension(); j++)

{

r[i][j] = (\*this)[j][i];

}

}

return r;

}

float cMatrix::Cofactor(int nRow, int nCol)

{

int nConst = 0;

if ((nRow + nCol) % 2 != 0)

nConst = -1;

else

nConst = 1;

// >> 부호 결정

return nConst \* Minor(nRow, nCol);

}

float cMatrix::Minor(int nRow, int nCol)

{

cMatrix rMinor(Dimension() - 1);

int nMinorRow = 0;

int nMinorCol = 0;

for (int i = 0; i < Dimension(); i++)

{

if (nRow == i)

continue;

nMinorCol = 0;

for (int j = 0; j < Dimension(); j++)

{

if (nCol == j)

continue;

rMinor[nMinorRow][nMinorCol] = (\*this)[i][j];

nMinorCol++;

}

nMinorRow++;

}

return rMinor.Determinant();

}