#include"Matrix3x3.h"

#include <cassert>

#include<cmath>

Matrix3x3 Matrix3x3::operator\*(const Matrix3x3& aOther)const noexcept

{

Matrix3x3 lMat= Matrix3x3(operator\*(aOther.column(0)),

operator\*(aOther.column(1)),

operator\*(aOther.column(2)));

return lMat.transpose();

}

float Matrix3x3::det()const noexcept

{

float lDet=0;

size\_t lIn[2];

for (size\_t i = 0; i < 3; i++)

{

Vector2D lVec[2];

size\_t p = 0;

for (size\_t k = 0; k < 2; k++) {

for (size\_t j = 0; j < 3; j++)

{

if (j != i) {

lIn[p] = j;

p++;

}

}

lVec[k] = Vector2D(row(k+1)[lIn[0]], row(k+1)[lIn[1]]);

p = 0;

}

lDet += row(0)[i] \* lVec[0].cross(lVec[1])\*(static\_cast<float>(pow(-1,i)));

}

return lDet;

}

Matrix3x3 Matrix3x3::transpose()const noexcept

{

return Matrix3x3(column(0),column(1),column(2));

}

Matrix3x3 Matrix3x3::inverse()const

{

assert(det() != 0);

float lVal[9];

Vector2D lVec[2];

size\_t lIn[2];

size\_t k = 0;

size\_t lInd = 0;

for (size\_t iRow = 0; iRow < 3; iRow++)

{

for(size\_t jCol=0; jCol<3;jCol++)

{

k = 0;

lIn[0] = 4;

size\_t p = 0;

for (size\_t i = 0; i < 3; i++)

{

if (i != iRow)

{

if (p == 0) {

for (size\_t j = 0; j < 3; j++)

{

if (j != jCol && lIn[0] != j)

{

lIn[p] = j;

p++;

}

}

}

lVec[k]=Vector2D(row(i)[lIn[0]],row(i)[lIn[1]]);

k++;

}

}

lVal[lInd] = lVec[0].cross(lVec[1])\*(static\_cast<float>(pow(-1,iRow+jCol)));

lInd++;

}

}

Matrix3x3 lMat = Matrix3x3(Vector3D(lVal[0], lVal[1], lVal[2]),

Vector3D(lVal[3], lVal[4], lVal[5]),

Vector3D(lVal[6], lVal[7], lVal[8])

);

return lMat.transpose()\* (1 / det());

}

bool Matrix3x3::hasInverse() const noexcept

{

if (det() == 0) {

return false;

}

else {

return true;

}

}

std::ostream& operator<<(std::ostream& aOStream, const Matrix3x3& aMatrix)

{

return aOStream << "[" << aMatrix.row(0) << std::endl

<< aMatrix.row(1) << std::endl

<< aMatrix.row(2) << std::endl << "]";

}