[ 게임 알고리즘 ]

**게임 공간에 수학적 물리적 개념 적용**

인하대학교 미래인재개발원 문화콘텐츠프로그래밍

김예슬

**벡터와 행렬을 이용한 공간 변환에 대한 코드**

cVector3 cVector3::TransformCoord(cVector3 & v, cMatrix & mat)

{

float a = mat[0][0] \* v.GetVectorX() + mat[1][0]

\* v.GetVectorY() + mat[2][0] \* v.GetVectorZ() + mat[3][0] \* 1;

float b = mat[0][1] \* v.GetVectorX() + mat[1][1]

\* v.GetVectorY() + mat[2][1] \* v.GetVectorZ() + mat[3][1] \* 1;

float c = mat[0][2] \* v.GetVectorX() + mat[1][2]

\* v.GetVectorY() + mat[2][2] \* v.GetVectorZ() + mat[3][2] \* 1;

float w = mat[0][3] \* v.GetVectorX() + mat[1][3]

\* v.GetVectorY() + mat[2][3] \* v.GetVectorZ() + mat[3][3] \* 1;

if (-dEpsilon > w || w > dEpsilon)

{

a /= w;

b /= w;

c /= w;

}

cVector3 retVec(a, b, c);

return retVec;

}

cVector3 cVector3::TransformNormal(cVector3 & v, cMatrix & mat)

{

float a = mat[0][0] \* v.GetVectorX() + mat[1][0]

\* v.GetVectorY() + mat[2][0] \* v.GetVectorZ();

float b = mat[0][1] \* v.GetVectorX() + mat[1][1]

\* v.GetVectorY() + mat[2][1] \* v.GetVectorZ();

float c = mat[0][2] \* v.GetVectorX() + mat[1][2]

\* v.GetVectorY() + mat[2][2] \* v.GetVectorZ();

cVector3 retVec(a, b, c);

return retVec;

}

cMatrix cMatrix::View(cVector3 & vEye, cVector3 & vLookAt, cVector3 & vUp)

{

cVector3 sight = vLookAt - vEye;

sight = sight.Normalize();

cVector3 right = cVector3::Cross(vUp, sight);

right = right.Normalize();

cVector3 up = cVector3::Cross(sight, right);

up = up.Normalize();

cMatrix resMat = cMatrix::Identity(dArrSize);

SetVal(right, resMat, 0);

SetVal(up, resMat, 1);

SetVal(sight, resMat, 2);

resMat[3][0] = -cVector3::Dot(right, vEye);

resMat[3][1] = -cVector3::Dot(up, vEye);

resMat[3][2] = -cVector3::Dot(sight, vEye);

return resMat;

}

cMatrix cMatrix::Projection(float fFovY, float fAspect, float fNearZ, float fFalZ)

{

float sy = 1.0f / tanf(fFovY / 2.0f);

float sx = sy / fAspect;

cMatrix resMat(dArrSize);

resMat.SetZero();

resMat[0][0] = sx;

resMat[1][1] = sy;

resMat[2][2] = fFalZ / (fFalZ - fNearZ);

resMat[2][3] = 1;

resMat[3][2] = -fFalZ \* fNearZ / (fFalZ - fNearZ);

return resMat;

}

cMatrix cMatrix::ViewPort(float x, float y, float w, float h, float minZ, float maxZ)

{

cMatrix resMat = cMatrix::Identity(dArrSize);

resMat[0][0] = w/2.0f;

resMat[1][1] = -h/2.0f;

resMat[2][2] = maxZ - minZ;

resMat[3][0] = x + (w / 2.0f);

resMat[3][1] = y + (h / 2.0f);

resMat[3][2] = minZ;

return resMat;

}

**물리 연산 수행 코드**

- 이동에 따른 충돌 계산

if (GetAsyncKeyState(VK\_LEFT) & 0x8000 || GetAsyncKeyState(VK\_LEFT) & 0x8001)

{

moveDirection = eMoveLeft;

isRightSight = false;

MovePlayer(playerPos, moveDirection, eMoveSpeed, -1, 0);

int diffNum = 0;

if (CollisionMap(eMoveLeft, diffNum))

MovePlayer(playerPos, moveDirection, diffNum, -1, 0);

else

MovePlayer(playerPos, moveDirection, diffNum, 1, 0); CheckOut(playerPos, moveDirection);

}

bool Player::CollisionMap(int direction, int & lengthDiff)

{

RECT areaRect;

vector<TileMap> checkBtm = dMap->GetMapPos();

RECT checkRect = ConversionRect(playerPos);

if (direction == eMoveLeft)

{

for (int i = 0; i < checkBtm.size(); i++)

{

if (IntersectRect(&areaRect, &checkBtm[i].pos, &checkRect)

&& CheckTileMap(checkBtm[i]))

{

lengthDiff = checkBtm[i].pos.right - checkRect.left;

return false;

}

else if (IntersectRect(&areaRect, &checkBtm[i].pos,

&checkRect) && checkBtm[i].type == eMapSpike)

{

SetIsPlayerDead(true);

return true;

}

else if (IntersectRect(&areaRect, &checkBtm[i].pos, &checkRect)

&& checkBtm[i].type == eMapItem)

{

isGetItem = true;

return false;

}

}

return true;

}

}