한국컴퓨터정보학회 동계학술대회 논문집 제25권 제1호 (2017. 1)

현실감 높은 게임 구현을 위한 2차원 공간상의 원형 물체의 탄성충돌 모델 연구

고재협*, 방정원O
O*청강문화산업대학교 게임콘텐츠스쿨
e-mail: wja04003@naver.com*, jwbang@ck.ac.krO

A Study of Elastic Collision Model of a Circular Object in a 2 Dimensional Space for Realistic Game Implementation

Jae-Hyeop Ko*, Jung-Won BangO
O*School of Game, Chungkang College of Cultural Industries

● 요 약 ●

본 논문에서는 공을 쳐서 구멍에 넣는 당구 게임을 제작하면서 실제 물리 현상과 같은 충돌 구현을 위해 사용한 여러 가지 상황에서의 원형 물체의 2차원 탄성 충돌 모델 구현 방법에 대하여 연구하였다.

키워드: Elastic collision, Collision model, Gravity

I. Introduction

게임에 대한 사용자의 요구사항이 높아지면서 현실성 있는 게임을 구현하는 것은 게임 제작에서 있어서 주요한 요소로 작용하고 있다. 공의 충돌이 중심이 된 게임을 제작하면서 원형 물체의 2차원 탄성충돌에 대하여 연구하였다. 물체끼리의 충돌이 중심이 되는 게임의 내용상 충돌을 최대한 자연스럽게 보이도록 하는 것이 중요했다. 이 논문에서는 게임에 사용된 원형 물체간의 탄성충돌과 원형 물체의 벽과의 탄성충돌, 그리고 난간에 걸친 상태에 있는 원형물체에 작용하는 힘의 계산을 살펴보고자 한다.

II. Preliminaries

1. Related works

물체의 충돌은 물체의 모양이나 강도에 따라 다양하게 나타날 수 있다. 충돌 후 물체의 운동에너지 변화에 따라 탄성충돌과 비탄성충돌로도 나눌수 있다. 본 학술지에서는 부서지지 않는 원형 물체의 탄성충돌에 대해 다룬다. 다음과 같은 세 가지 형태의 충돌 연산이 게임에서 사용되었다.

1.1. 물체간의 충돌두 물체가 충돌하면 충돌한 면으로 힘이 작용한다. 그러므로 물체에 작용한 힘을 계산하면 물체의 충돌 이후 속도를 알아낼수 있다.[1]

Fig 1.은 2차원 공간에서 두 물체가 탄성충돌을 하는 경우를 보여준다. 그림의 오른쪽 위와 같이 충돌면에 수직하는 새로운 기준축을 설정하고 기준축에 대해 1차원 탄성충돌을 계산한 뒤 새로운 속도를 구한다. 식 1.은 1차원 탄성충돌시의 속도 계산공식이다.[2]

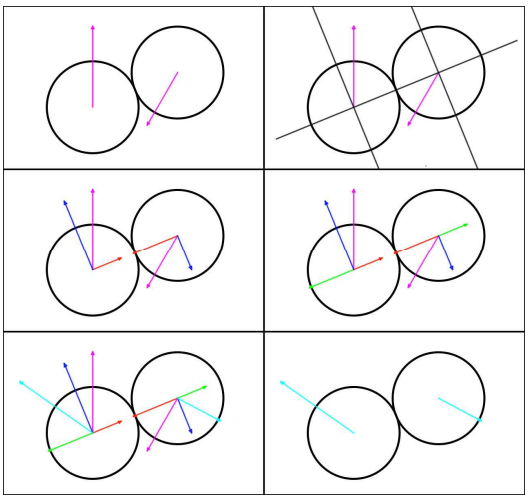

Fig 1. 2차원 공간에서 원형 물체가 탄성 충돌시 발생하는 속도의 변화

$$v_A' = v_A\left(\frac{m_A - m_B}{m_A + m_B}\right) + v_B\left(\frac{2m_B}{m_A + m_B}\right)$$
$$v_B' = v_A\left(\frac{2m_A}{m_A + m_B}\right) + v_B\left(\frac{m_B - m_A}{m_A + m_B}\right)$$

식 1. 1차원 탄성충돌 속도 계산식

1.2 물체와 선의 충돌

탄성충돌을 하는 물체가 벽에 부딪히게 되면 부딪힌 면에 대한 속도의 방향이 반대가 된다. 즉, 충돌 면에 대한 입사각과 반사각이 같다.

1.3 중력 모사

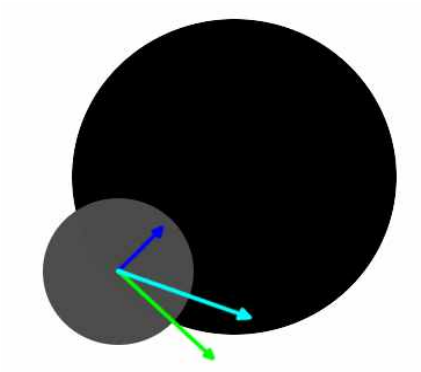

Fig 2. 중력으로 난간에서 일어나는 속도의 변화

이 게임에서는 공을 쳐서 구멍에 넣게 된다. 물체가 난간에 걸치게 되면 중력으로 인해 Fig 2와 같이 난간방향으로 힘을 받게 된다. 때문에 이 게임에서는 이 힘을 계산해 주어여 한다. 새로운 속도를 계산하기 위해서는 Fig 2의 파란색에 해당하는 중력으로 발생하는 속도를 계산해주어야 한다. Fig 3부터 Fig 6까지는 이를 계산하는 방법이다.

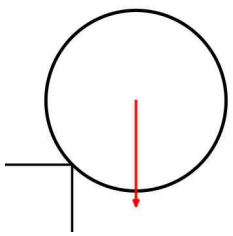

Fig 3. 난간에 걸쳐있는 상황

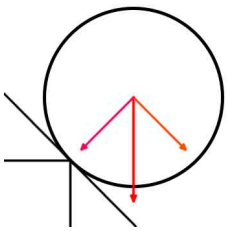

Fig 4. 접선에 대하여 중력을 분해

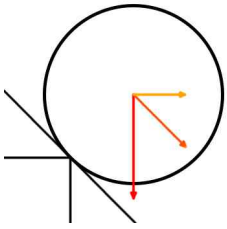

Fig 5. 접선에 대한 힘을 x축으로 분해

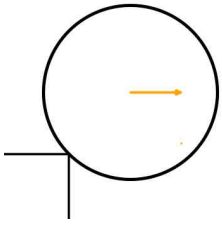

Fig 6. 공에 작용하는 힘

Fig 3에서 2차원 공간을 중력을 고려하기 위해 3차원 공간으로 잠시 확장해서 생각한다. 그림에서 x축은 2차원 평면이 되고 y축은 깊이가 된다. Fig 4에서 접선으로 작용하는 힘은 물체의 속도에 영향을 주지 못하므로 무시된다. 그러므로 접선과 평행하는 힘을 고려해야 한다. 이렇게 계산된 힘이 Fig 5의 노란색 힘이다. 하지만 물체는 2차원 공간속에서 움직이므로 힘을 2차원에 대한 힘으로 변환해주어야 한다. 그 과정이 Fig 6이다.

이렇게 계산된 힘에서 가속도를 계산해 새로운 속도를 계산해 내면 된다.

III. Test Result

1 물체끼리의 충돌

게임에서 사용하는 구형물체는 Fig 과와 같이 속력과 방향 값을 가진다.

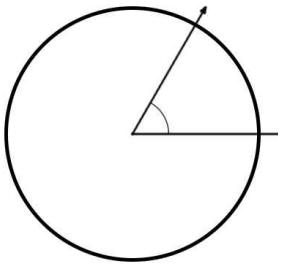

Fig 7. 게임 상에서 사용하는 구형물체가 가진 정보

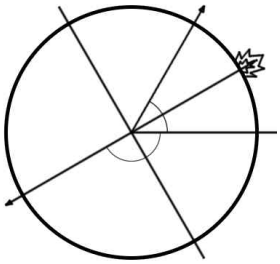

Fig 8. 충돌상황

Fig 8과 같이 충돌이 발생하면 충돌 면에 수직하는 새로운 축을 설정한다. 본 게임에서는 편의를 위해 축을 충돌 면을 향하는 방향이 아닌 반대 방향으로 설정하였다.

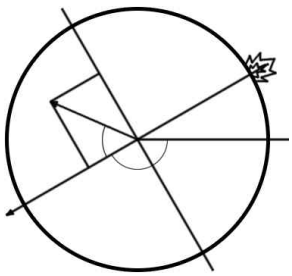

Fig 9. 충돌 후 새로운 속력과 각의 계산

충돌 후 새로운 속력이 계산되면 속력의 방향을 구해야 한다. 게임에서는 속력의 방향을 구하기 위해 충돌 축의 각(Fig 9의 오른쪽 아래 각)과 충돌 축에 대한 새로운 속력의 각(Fig 9의 왼쪽 각)을 합하였다.

Table 1. 물체끼리의 충돌 코드

```
int index1 = -1, index2 = -1;

while (isBallCollision(index1 + 1, index1, index2))
{
float degree = CalculateDegree((*m_balls)[index1]->GetX() -
(*m_balls)[index2]->GetX(),(*m_balls)[index1]->GetY() -
(*m_balls)[index2]->GetY());

float deltaDegree;
deltaDegree = (*m_balls)[index1]->GetDegree() - degree;
float vax = cosf(deltaDegree)*(*m_balls)[index1]->GetPower();
float vay = sinf(deltaDegree)*(*m_balls)[index1]->GetPower();

deltaDegree = (*m_balls)[index2]->GetDegree() - degree +
PI;
float vbx = cosf(deltaDegree)*(*m_balls)[index2]->GetPower();
float vby = sinf(deltaDegree)*(*m_balls)[index2]->GetPower();

float newvax;
float newvbx;
float ma = (*m_balls)[index1]->GetWeight();
float mb = (*m_balls)[index2]->GetWeight();

newvax = (ma - mb) / (ma + mb)*vax + (2.0f*mb) / (ma +
mb)*vbx;
newvbx = (2.0f*ma) / (ma + mb)*vax + (mb - ma) / (ma +
mb)*vbx;

(*m_balls)[index1]->SetPower(sqrtf(newvax*newvax +
vay*vay));
(*m_balls)[index2]->SetPower(sqrtf(newvbx*newvbx +
vby*vby));

if ((*m_balls)[index1]->GetPower() != 0)
(*m_balls)[index1]->SetDegree(asinf(vay /
(*m_balls)[index1]->GetPower()) + degree);
if ((*m_balls)[index1]->GetDegree() < PI)

(*m_balls)[index1]->SetDegree((*m_balls)[index1]->GetDegree
() + 2 * PI);
if ((*m_balls)[index2]->GetPower() != 0)
(*m_balls)[index2]->SetDegree(asinf(vby /
(*m_balls)[index2]->GetPower()) + degree + PI);
if ((*m_balls)[index2]->GetDegree() < PI)
(*m_balls)[index2]->SetDegree((*m_balls)[index2]->GetDegree
() + 2 * PI);
}
```

2 물체와 선의 충돌

다음은 게임상에서 구현한 방법을 그림으로 나타낸 것이다.

Fig 10과 같이 충돌한 상황은 Fig 11과 같이 충돌면에 대해 탄성충돌을 했다고 생각할 수 있다. 위와 같은 경우로 생각하면 물체끼리의 충돌상황과 같은 계산 방식으로 새로운 각을 구할 수 있다.

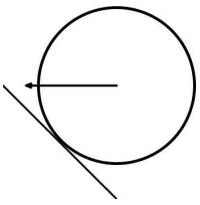

Fig 10. 선에 충돌할 때의 상태

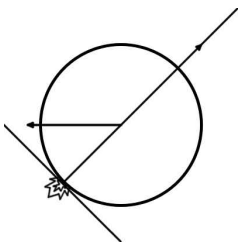

Fig 11. 선에 충돌한 상황

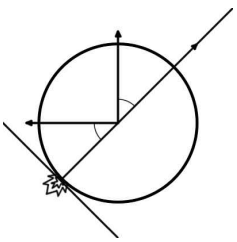

Fig 12. 새로운 각을 계산하는 과정, 두 각의 크기는 같다.

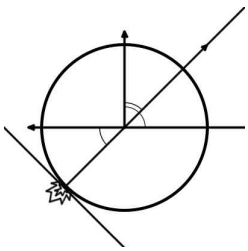

Fig 13. 새로운 각을 계산하는 과정

Table 2. 물체와 선의 충돌 코드

```
int index1 = -1, index2 = -1;
float x, y;
while (isLineCollision(index1 + 1, index2 + 1, index1,
index2, x, y))
{
float degree = CalculateDegree((*m_balls)[index1]->GetX()
- x, (*m_balls)[index1]->GetY() - y);

float deltaDegree;
deltaDegree = (*m_balls)[index1]->GetDegree() - degree;
float vay =
sinf(deltaDegree)*(*m_balls)[index1]->GetPower();

if ((*m_balls)[index1]->GetPower() != 0)
(*m_balls)[index1]->SetDegree(asinf(vay /
(*m_balls)[index1]->GetPower()) + degree);
if ((*m_balls)[index1]->GetDegree() < PI)

(*m_balls)[index1]->SetDegree((*m_balls)[index1]->GetDegr
ee() + 2 * PI);

(*m_balls)[index1]->SetX(x +
(float)(((*m_balls)[index1]->GetRadius() + 1)*cosf(degree)));
(*m_balls)[index1]->SetY(y +
(float)(((*m_balls)[index1]->GetRadius() + 1)*sinf(degree)));
}
```

3. 중력모사

Table3 은 중력모사를 구현한 게임 코드이다.

Table 3. 중력모사 코드

```
int index1 = -1, index2 = -1;
float x, y;
while (isBallInGoal(index1 + 1, index2 + 1, index1, index2))
{
if ((*m_balls)[index1]->GetRadius() != 0)
{
float degree = CalculateDegree((*m_goals)[index2]->GetX()
- (*m_balls)[index1]->GetX(),
(*m_goals)[index2]->GetY() - (*m_balls)[index1]->GetY());

float deltaDegree;
deltaDegree = (*m_balls)[index1]->GetDegree() - degree;
float DeltaPerRadius = ((*m_goals)[index2]->GetRad() -
sqrtf(((*m_goals)[index2]->GetX() -
(*m_balls)[index1]->GetX())*
((*m_goals)[index2]->GetX() - (*m_balls)[index1]->GetX())
+ ((*m_goals)[index2]->GetY() -
(*m_balls)[index1]->GetY()) *
((*m_goals)[index2]->GetY() -
(*m_balls)[index1]->GetY()))) /
(*m_balls)[index1]->GetRadius();
float vax =
cosf(deltaDegree)*(*m_balls)[index1]->GetPower() +
(*m_balls)[index1]->GetWeight() * 10 * DeltaPerRadius *
sin(acos(DeltaPerRadius));
float vay =
sinf(deltaDegree)*(*m_balls)[index1]->GetPower();

(*m_balls)[index1]->SetPower(sqrtf(vax*vax + vay*vay));

if ((*m_balls)[index1]->GetPower() != 0)
(*m_balls)[index1]->SetDegree(asinf(vay /
(*m_balls)[index1]->GetPower()) + degree);
if ((*m_balls)[index1]->GetDegree() < PI)

(*m_balls)[index1]->SetDegree((*m_balls)[index1]->GetDegr
ee() + 2 * PI);
}
}
```

다음은 실제 게임의 구현 모습이다.

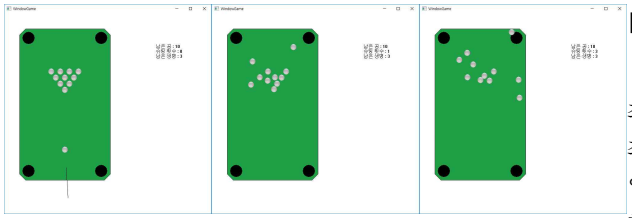

Fig 17. 스테이지 1

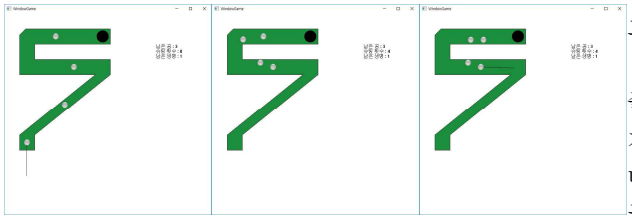

Fig 18. 스테이지 2

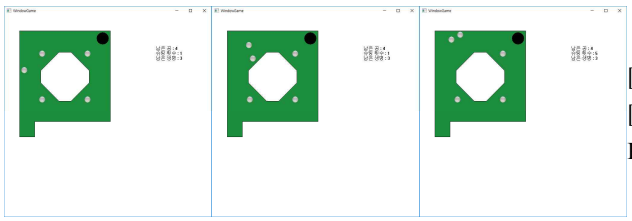

Fig 19. 스테이지 3

IV. Conclusions

이 게임에서는 제작의 간단함을 위하여 충돌이 일어나는 공간의 좌표계를 프로그램 화면상의 좌표로 처리했다. 하지만 이런 방법의 좌표계은 실제 물리작용을 모사하기에는 부족하다. 때문에 충돌이 일어나는 공간을 미터를 단위로 하는 가상의 공간으로 하고 이 공간을 화면에 투영해 보여주는 방식이 나을 것으로 생각된다. 이렇게 하면 중력가속도등, 가속도에 대한 계산과 속도의 계산이 보다 정확해질 것으로 생각된다. 또한, 3개의 물체가 동시에 충돌하는 경우에 대한 고려가 되어야 한다.

게임 상에서는 공이 멈춰야 하기 때문에 공의 속력을 일정하게 줄여주었다. 하지만 공의 속도가 줄어드는 것은 다양한 원인이 작용하기 때문에 일정하게 속력을 줄이는 방식으로는 불충분하다. 공은 대기라는 유체 속에서 움직이기 때문에 유체 속에서 받는 항력과 공의 회전으로 인한 관성모멘트를 고려해 주어야 한다.

References

[1] https://en.wikipedia.org/wiki/Elastic_collision

[2] Wendy Styler, "Beginning Math and Physics for Game Programmers", Jeu Media, pp354-363,2004