

DanguDanawa



Crazy Brother

An Jun Young

Lim Chung Ryeol

DanguDanawa

Contents

1

• Introduction

2

• System Architecture

3

• 3D Graphic & Simulation

4

• Image Processing

5

• Billiards Algorithm & Physical Engine

Introduction

■ Goal

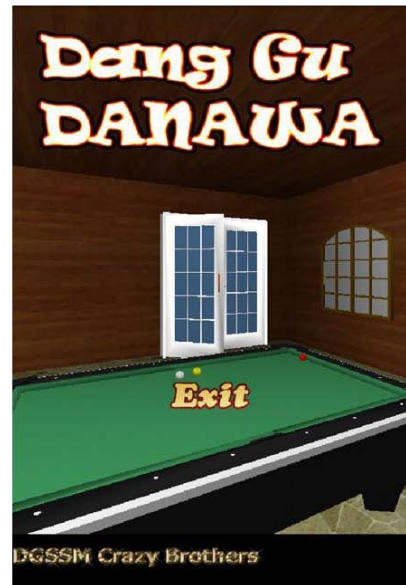
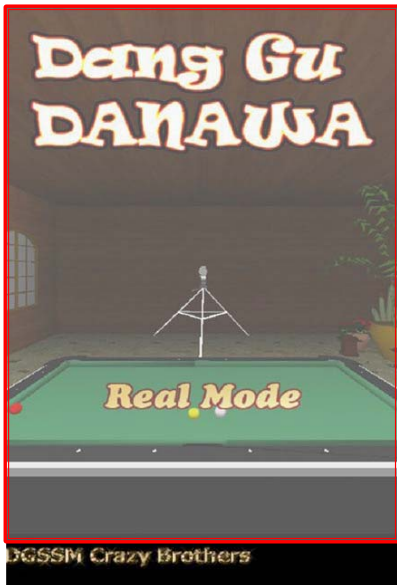
- 스마트폰 기반에서 영상처리를 통한 당구 득점 경로 제공



System Architecture

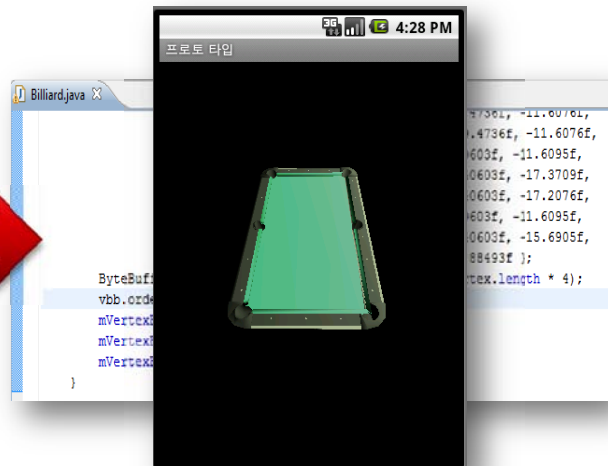


User Interface



■ FrameLayout + LinearLayout

Java code Generator



Java code Generator

```

+MESH 1
+TIMEVALUE 0
+MESH_NUMVERTEX 14
+MESH_NUMFACES 12
+MESH_VERTEX_LIST 1
-MESH_VERTEX 0 1.0000 -1.7299 40.3429
-MESH_VERTEX 1 0.1955 1.7534 37.6788
-MESH_VERTEX 2 -0.6966 0.7463 41.2498
-MESH_VERTEX 3 1.0523 0.5899 37.2527
-MESH_VERTEX 4 -0.1698 -4.5446 42.7179
-MESH_VERTEX 5 -2.9938 -0.8359 44.0751
-MESH_VERTEX 6 -8.3829 -10.5766 49.0606
-MESH_VERTEX 7 -12.7938 -10.9870 48.5559
-MESH_VERTEX 8 -11.8624 -12.1933 48.5940
-MESH_VERTEX 9 -10.3649 -8.0115 48.9796
-MESH_VERTEX 10 -4.9795 -8.9982 48.1361
-MESH_VERTEX 11 -7.9462 -5.1564 48.0149
-MESH_VERTEX 12 -5.8522 -2.7939 45.7874
-MESH_VERTEX 13 -2.0961 -7.6576 45.9408
}
+MESH_FACE_LIST 1
-MESH_FACE 0: A: 0 B: 1 C: 2 AB: 0 BC: 1 CA: 1
-MESH_FACE 1: A: 1 B: 0 C: 3 AB: 0 BC: 1 CA: 1
-MESH_FACE 2: A: 4 B: 2 C: 5 AB: 0 BC: 1 CA: 1
-MESH_FACE 3: A: 2 B: 4 C: 6 AB: 0 BC: 1 CA: 1
-MESH_FACE 4: A: 6 B: 7 C: 8 AB: 0 BC: 1 CA: 1
-MESH_FACE 5: A: 7 B: 6 C: 9 AB: 0 BC: 1 CA: 1
-MESH_FACE 6: A: 10 B: 9 C: 6 AB: 0 BC: 1 CA: 1
-MESH_FACE 7: A: 9 B: 10 C: 11 AB: 0 BC: 1 CA: 1
-MESH_FACE 8: A: 10 B: 12 C: 11 AB: 0 BC: 1 CA: 1
-MESH_FACE 9: A: 12 B: 10 C: 13 AB: 0 BC: 1 CA: 1
-MESH_FACE 10: A: 5 B: 12 C: 13 AB: 1 BC: 1 CA: 1
-MESH_FACE 11: A: 4 B: 5 C: 13 AB: 1 BC: 1 CA: 1
    
```

ASE to Java Source

```

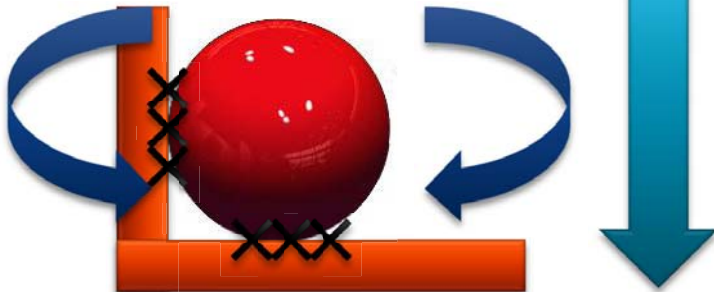
ByteBuffer nbb = ByteBuffer.allocateDirect(valueNormal.length * 4);
nbb.order(ByteOrder.nativeOrder());
mNormalBuffer(114) = nbb.asFloatBuffer();
mNormalBuffer(114).put(valueNormal);
mNormalBuffer(114).position(0);
}

public void setNormal115() {
    float valueNormal[] = { 0f, 0f, 0f, 0f, 0f, 0f, 0f, 0f, 0f, 0f, 0f, 0f };
    ByteBuffer nbb = ByteBuffer.allocateDirect(valueNormal.length * 4);
    nbb.order(ByteOrder.nativeOrder());
    mNormalBuffer(115) = nbb.asFloatBuffer();
    mNormalBuffer(115).put(valueNormal);
    mNormalBuffer(115).position(0);
}

public void setNormal116() {
    float valueNormal[] = { 0f, 0f, 0f, 0f, -2f, 0f, 0f, -1f, 0f, 0.0042621f, -2.99999f, 0f, 0.000503f, -3.99999f, 0f, 0.000503f, -4.99999f, 0f, 0.0042621f, -2.99999f, 0f, 0.000503f, -1.99999f, 0f, 0.000503f, -3.99999f, 0f, 0f, -1f, 0f, 0f, -1f, 0f, 0f, -3f, 0f, 0f, -1f, 0f, 0f, -2f, 0f, 0.0042621f, -2.99999f, 0f, 0.000503f, -3.99999f, 0f };
    ByteBuffer nbb = ByteBuffer.allocateDirect(valueNormal.length * 4);
    nbb.order(ByteOrder.nativeOrder());
    mNormalBuffer(116) = nbb.asFloatBuffer();
    mNormalBuffer(116).put(valueNormal);
    mNormalBuffer(116).position(0);
}

public void setNormal117() {
    float valueNormal[] = { 0.003737f, -5.99913f, 0.0500893f, 0.0028871f, -2.99999f, 0.014435f, 0f, -2f, 0f, 0.0477965f, -1.99912f, -0.0006012f, 0.002724f, -1.99999f, 0.15573f, 0.082415f, -7.99439f, 0.068784f, 0.0193226f, -1.99978f, 0.0000012f, 0.0000002f, -1.99912f, 0.0135311f, 0.0000002f, -1.99742f, 0.027296f, 0f, -1.99912f, 0.124759f, 0.0000002f, -2.99739f, 0.005139f, 0.0106596f, -1.99954f, 0.14542f, 0.017224f, -1.91909f, 0.403228f, 0.0029871f, -2.91904f, 0.408154f, 0.0006099f, -5.9993f, 0.022992f, 0f, -2f, 0f, 0.0000002f, -1.99912f, 0.0118131f };
    ByteBuffer nbb = ByteBuffer.allocateDirect(valueNormal.length * 4);
    nbb.order(ByteOrder.nativeOrder());
    mNormalBuffer(117) = nbb.asFloatBuffer();
    mNormalBuffer(117).put(valueNormal);
    mNormalBuffer(117).position(0);
}
    
```

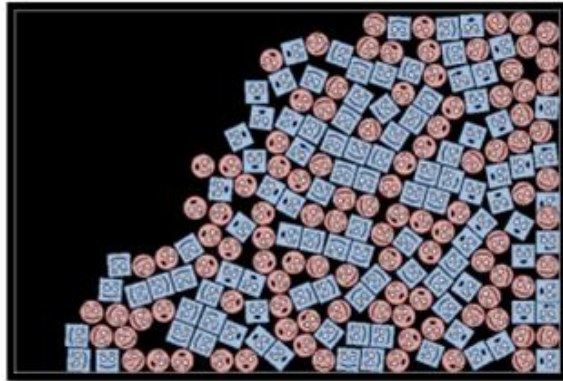
Physics Attribute



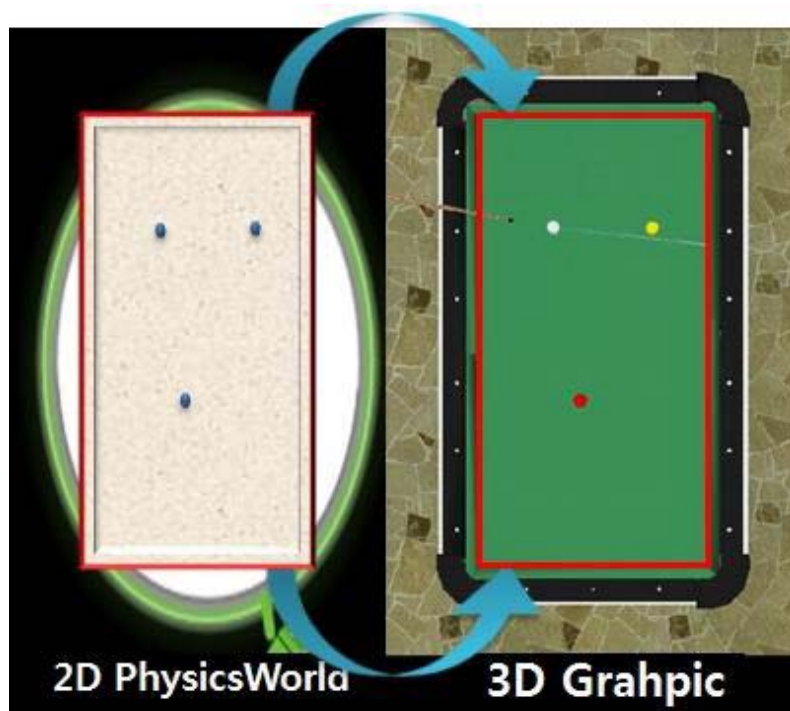
- Attribute
- Angular Velocity
- Linear Velocity
- Gravity (Damping)
- Density
- Elasticity
- Friction

Apply Physics Engine

- AndEngine is a free Android 2D OpenGL Game Engine.



Apply Physics Engine



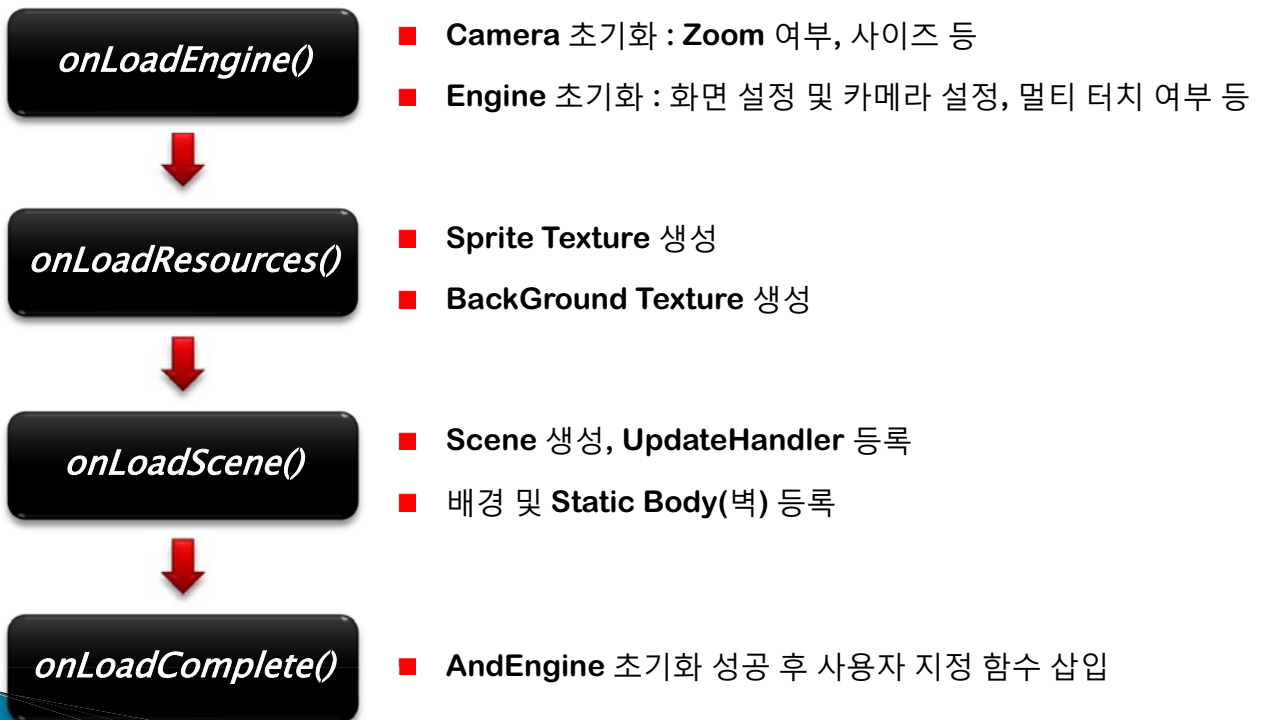
2D PhysicsWorld

3D Grahpic

AndEngine Function

- Drawing Object
- Multi Touch
- Particle Systems
- Network
- Physics
- Text
- Audio
- Backgrounds
- Using Menus, Sub Menu, ZoomCamera, Using ImageFormats(PNG, JPG, GIF, BMP), etc...

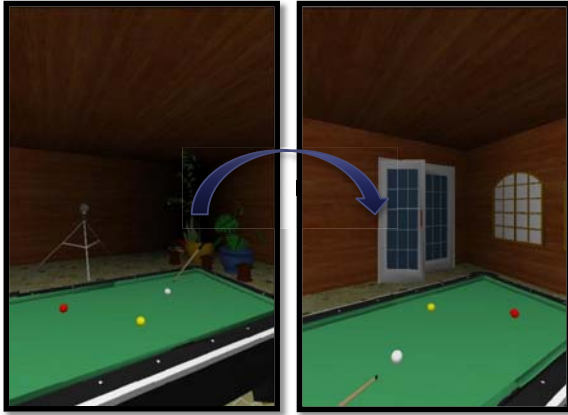
AndEngine Flow Chart



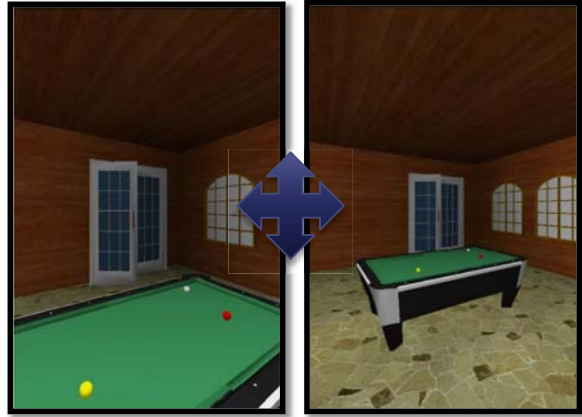
Simulation Function

■ Run, Reset, Help

■ Rotate

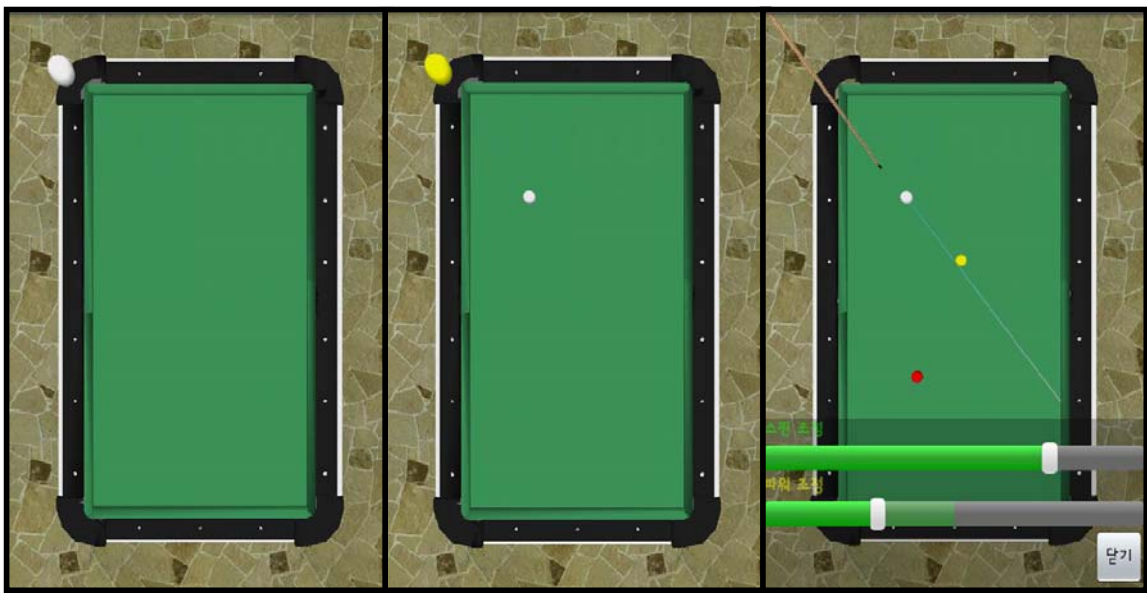


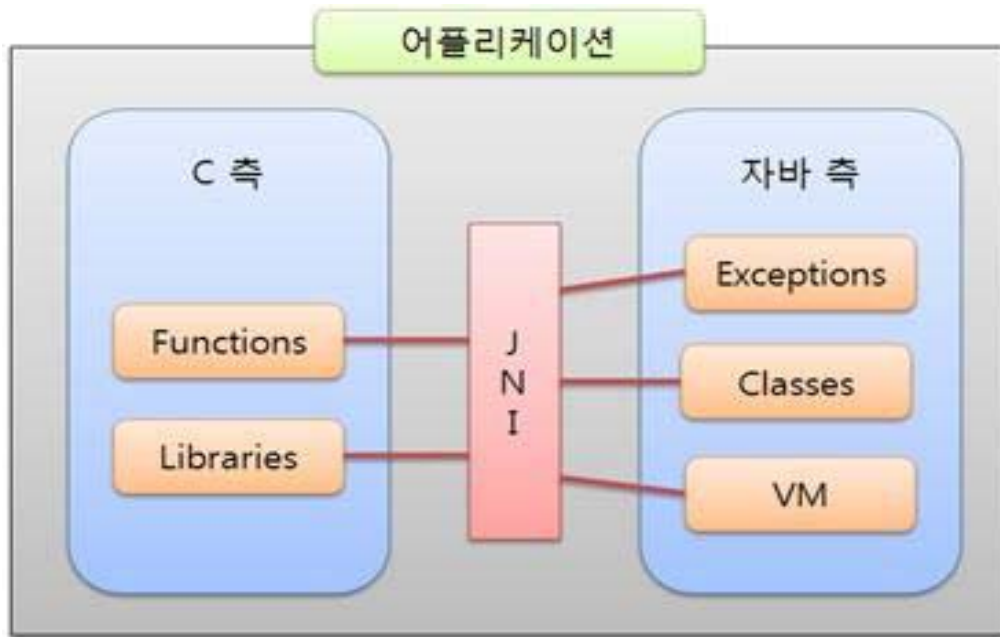
■ Zoom in/out



Free Mode Function

■ Put a Ball, Control, Shot, Reset, Help





-색 검출을 통해
필요한 당구대의
영역 확보

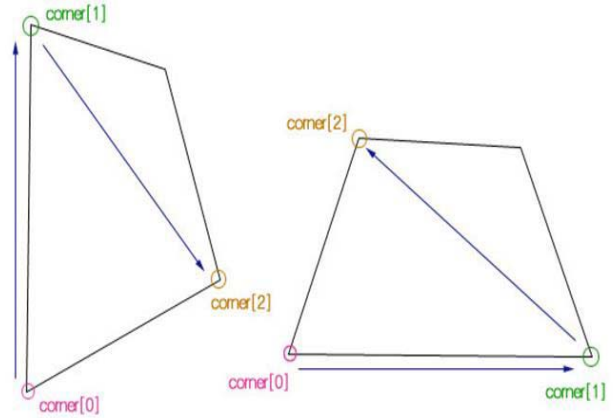
-꼭지점 추출을
위한 윤곽선 추출



❖ 꼭지점 추출

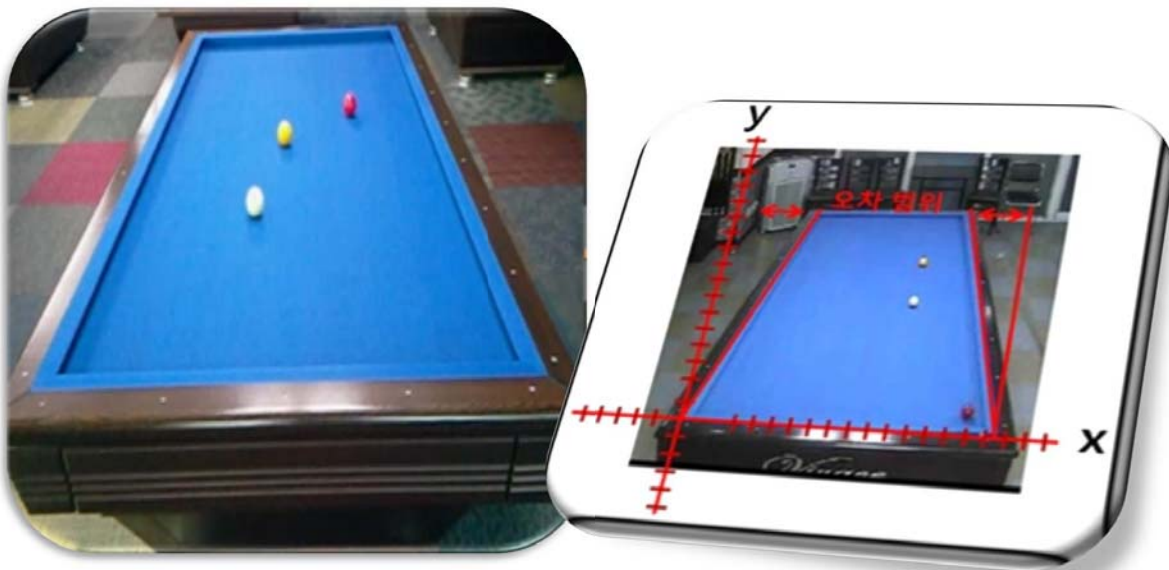
$$A = \frac{1}{2} (x_1y_2 - x_2y_1 + x_2y_3 - x_3y_2 + \dots + x_ny_1 - x_1y_n)$$

$$= \frac{1}{2} \{x_1(y_2 - y_n) + x_2(y_3 - y_1) + \dots + x_n(y_1 - y_{n-1})\}$$



- 임의의 점에서 가장 먼 점을 첫 번째 꼭지점
- 같은 방법으로 두 번째 꼭지점
- 첫 번째와 두 번째 점에서 가장 먼 점이 세 번째 꼭지점
- 사각형의 넓이 정보로 네 번째 꼭지점

❖ 영상 보정

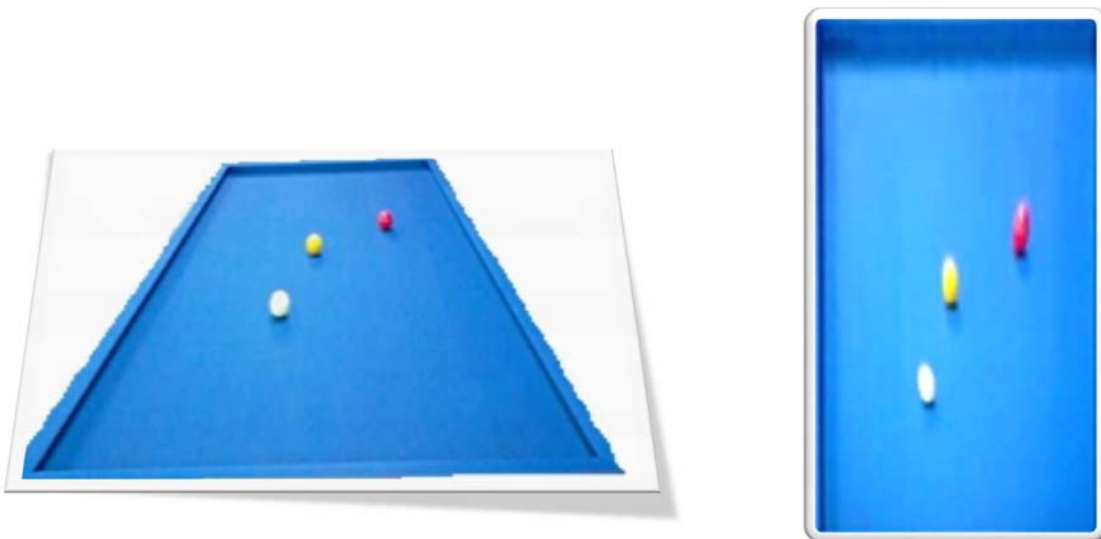


실제 당구대를 데이터로 활용하기 위해서 발생하는 문제점

영상 보정

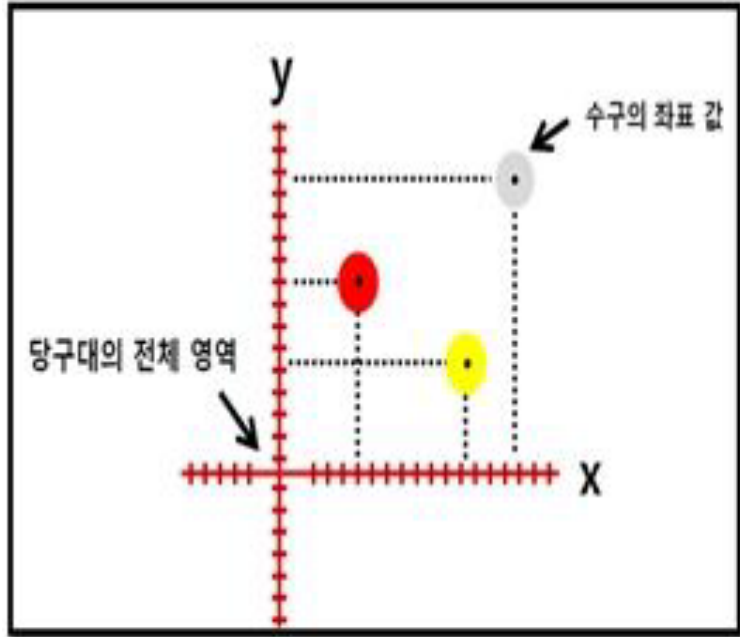


영상 보정

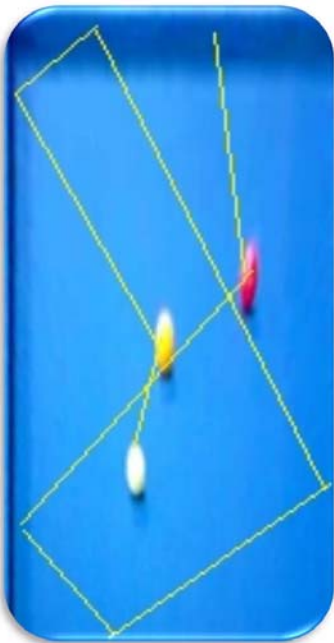


-추출된 당구대를 실제 당구대 비율과 동일한 변환 매트릭스를 만들어 원본이미지를 구하고자 하는 형태의 이미지로 매칭

수구의 좌표 값 전달



수구의 득점 경로 표시



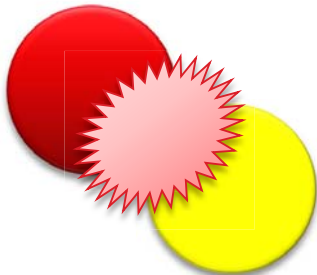
-영상보정의 반대 방법으로 수구의 득점 경로를 입력하여 원 영상으로 병합

Billiards Algorithm

- 목표
 - 당구공의 위치를 이용하여 그 위치에서 3쿠션이 되는 최적의 길을 찾아 주는 것
- 제약사항
 - 당구대의 크기는 당구장에서 가장 많이 사용하는 일반적인 당구대의 크기인
 - 단 쿠션방향 : 1224mm
 - 장 쿠션방향 : 2448mm
 - 당구공의 사이즈는 3구 공의 크기가 아닌 4구 공의 크기인 65.5m

Billiards Algorithm

- 당구에서 충돌의 종류
 - 공과 공



- 공과 큐



- 벽과 공

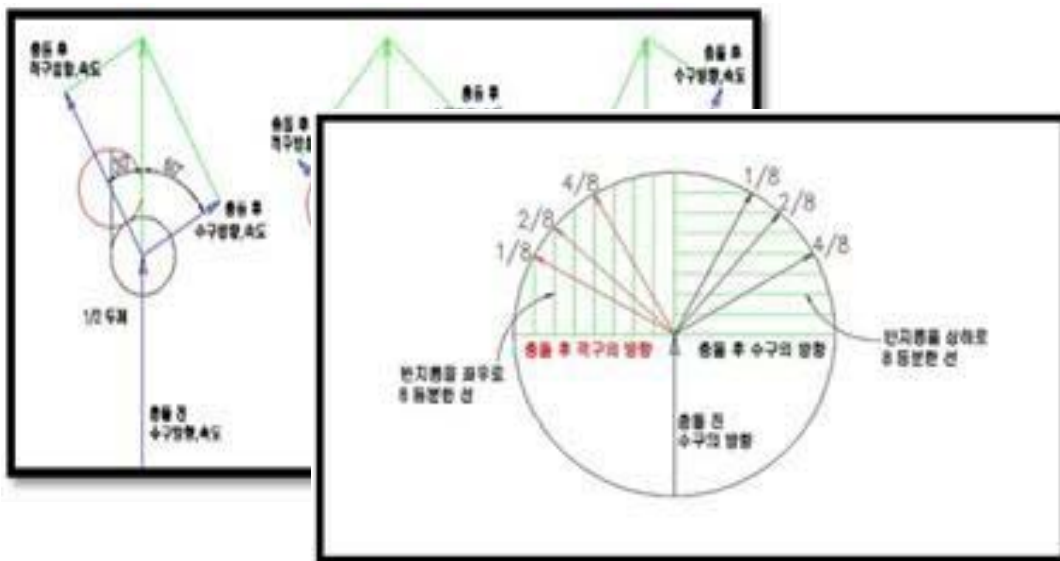


Billiards Algorithm

- 발생하는 물리현상
 - 쿠션의 역할
 - 당구공의 운동에너지를 잠시 담았다 반대 방향으로 되돌려 줌
 - 당구공에 회전이 있으면, 회전방향과 충돌방향이 같으면 반사각이 더 작아지고 반대의 경우이면, 커짐
- 큐의 역할
 - 운동에너지를 가해서 당구공을 움직이게 하는 요소
- 당구대의 역할
 - 마찰력을 발생시켜 공을 멈추게 하는 역할

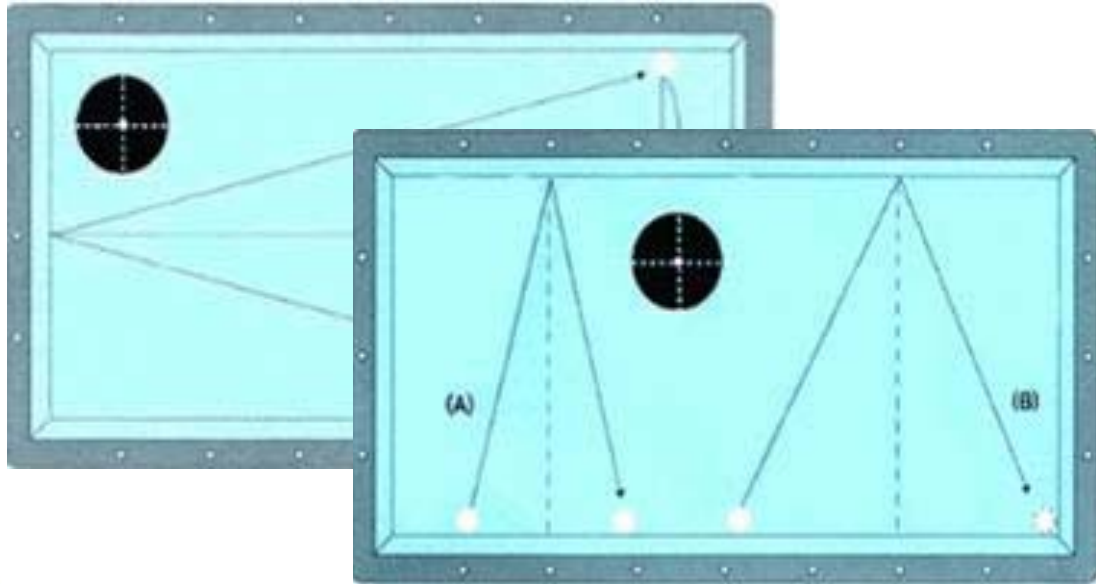
Billiards Algorithm

- 물리 법칙
 - 입사각과 반사각 법칙



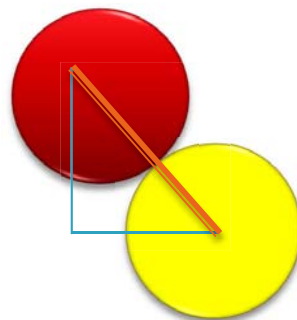
Billiards Algorithm

- 물리 법칙
 - 입사각과 반사각 법칙



Physical Engine

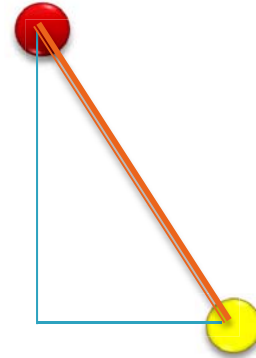
- 수구와 적구의 충돌 확인
 - 피타고라스의 정리를 이용하여 충돌 확인
 - BALLSIZE가 빗변의 길이보다 같아지게 되면 충돌 확인
 - $(X2 - X1)^2 + (Y2 - Y1)^2 \leq \text{BALLSIZE} * \text{BALLSIZE}$



Physical Engine

- 수구와 적구 사이의 각도
 - 피타고라스의 정리를 이용하여 밑변과 높이를 이용하여 빗변을 구하고 그것의 호도각을 이용
 - 빗변을 c , 밑변을 a 라고, 각도를 angle 라고 하면

$$\cos(\text{angle}) = a/c \quad \text{즉} \quad \text{angle} = \arccos(a/c)$$
 - $\text{atan2}(y_2 - y_1, x_2 - x_1) * 180 / \pi$



Physical Engine

- 완전 탄성 충돌에서 사용 가능한 법칙

- 1. 운동에너지 보존법칙

$$\frac{1}{2}m_1v_1^2 + \frac{1}{2}m_2v_2^2 = \frac{1}{2}m_1v_1'^2 + \frac{1}{2}m_2v_2'^2$$

- 2. 운동량 보존법칙

$$m_1v_1 + m_2v_2 = m_1v_1' + m_2v_2'$$

- 3. 반발계수 = 1

$$1 = -\frac{(v_1' - v_2')}{(v_1 - v_2)}$$

Physical Engine

- 반발계수 식에서

$$v_1 - v_2 = -(v'_1 - v'_2), \quad v'_2 = v_1 - v_2 + v'_1 \text{ 을}$$

- 운동량 보존법칙에 대입

$$m_1 v_1 + m_2 v_2 = m_1 v'_1 + m_2 (v_1 - v_2 + v'_1)$$

- v'_1 에 대해 정리하면,

$$(m_1 + m_2)v'_1 = m_1 v_1 - m_2 v_1 + 2m_2 v_2$$

- 따라서

$$\therefore v'_1 = \frac{(m_1 - m_2)v_1}{m_1 + m_2} + \frac{2m_2 v_2}{m_1 + m_2}$$

Billiards Algorithm

- 각도를 이용하여 충돌할 곳을 예측 하여 길을 찾는 방법

- 장점 : 충돌 지점만을 검사하면 예측이 가능하기 때문에 알고리즘을 계산하는 속도가 빠름
- 단점 : 키스를 예측하기 힘들고 키스와 모든 상황을 고려하면 장점이 단점으로 변함

- 수구를 시뮬레이션처럼 미리 굴러보고 길을 찾는 방법

- 장점 : 키스와 같은 모든 상황을 대처할 수 있고 키스를 이용한 길 또한 찾을 수 있음
- 단점 : 공의 움직이는 모든 경우를 예측해야 하기 때문에 알고리즘을 계산하는 속도가 느림