26 Colisões bidimensionais

Considere uma situação em que uma partícula de massa m_1 move-se para a direita ao longo do eixo x com velocidade v_{1a} e colide com uma partícula de massa m_2 inicialmente em repouso na origem do sistema de coordenadas. Após a colisão, a partícula 1 passa a mover-se com velocidade v_{1b} fazendo um ângulo θ com a direção de incidência; a partícula 2 passa a mover-se com velocidade v_{2b} fazendo um ângulo φ com a direção de incidência. Numa situação simples em que não existem forças externas atuando no sistema, não há variação da energia interna, as massas são iguais e, ignorando qualquer possível movimento de rotação, a conservação da energia cinética $E = \frac{1}{2}mv^2$ e do momento linear $\mathbf{p} = m\mathbf{v}$ levam a:

$$v_{1a}^2 = v_{1b}^2 + v_{2b}^2$$
$$\mathbf{v}_{1a} = \mathbf{v}_{1b} + \mathbf{v}_{2b}$$

A segunda equação pode ser reescrita como $\mathbf{v}_{2b} = \mathbf{v}_{1a} - \mathbf{v}_{1b}$, que, elevada ao quadrado fica $\mathbf{v}_{2b} \cdot \mathbf{v}_{2b} = (\mathbf{v}_{1a} - \mathbf{v}_{1b}) \cdot (\mathbf{v}_{1a} - \mathbf{v}_{1b})$, ou ainda $v_{2b}^2 = v_{1a}^2 + v_{1b}^2 - 2\, v_{1a} v_{1b} \cos\theta$. Substituindo v_{2b}^2 pela expressão que pode ser obtida da primeira equação, obtemos finalmente:

$$\cos\theta = v_{1b} / v_{1a}$$

A conservação do momento na direção perpendicular à direção de incidência da partícula 1 nos diz que $v_{1b} \sin\theta = v_{2b} \sin\varphi$. Elevando os dois lados desta igualdade ao quadrado e lembrando que $\sin^2\theta = 1 - \cos^2\theta$, chegamos a:

$$\sin\varphi = v_{1b} / v_{1a} = \cos\theta$$

Esta relação expressa o fato de que, em colisões deste tipo, o ângulo entre as duas partículas emergentes é sempre $\theta + \varphi = 90°$.

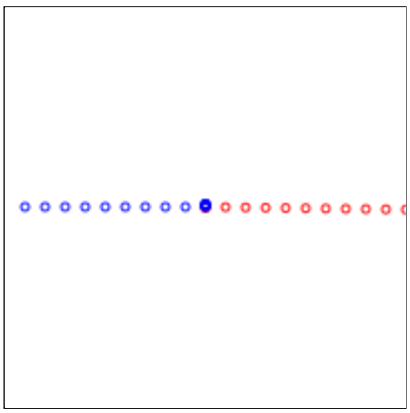
O script a seguir simula uma colisão bidimensional em que é sorteado um ângulo de espalhamento para a partícula 1 e, a partir dele, determinados os valores dos módulos das velocidades das duas partículas além do ângulo de espalhamento da partícula 2.

`exemplo-26-1.html`

```
<p style="text-align:center">
<canvas style="border: 1px solid" id="cnv" width="200" height="200">
</canvas>
</p>

<script>
var cw,ch;
var canvas = document.getElementById("cnv");
var max = canvas.attributes.length;
for (var i = 0; i < max; i++) {
   if (canvas.attributes[i].nodeName == "width") {
      cw = canvas.attributes[i].value;
   } else if (canvas.attributes[i].nodeName == "height") {
   ch = canvas.attributes[i].value;
   }
}
var ctx = canvas.getContext('2d');

var wxi = -1;
var wxf = +1;
var wyi = -1;
var wyf = +1;
var tc = new TransCoord(cw,ch,wxi,wyi,wxf,wyf);

var x1 = wxi;
var y1 = 0;
var x2 = 0;
var y2 = 0;

var vo = 0.001;
var v1x = vo;
var v1y = 0;
var v2x = 0;
var v2y = 0;

var r = (wxf-wxi)*0.01;
var dt = 100;
var intervalo = setInterval('passo()',dt);

var colisao = 0;

function passo() {

   // propaga as posições
   x1 = x1 + v1x * dt;
   y1 = y1 + v1y * dt;
   x2 = x2 + v2x * dt;
   y2 = y2 + v2y * dt;

   // verifica se houve colisão
   if ((Math.abs(x1-x2)<(2*r))&&(colisao==0)) {

      var theta = Math.random() * Math.PI/2;
      var v1a = Math.sqrt(v1x*v1x+v1y*v1y);
      var v1b = v1a * Math.cos(theta);
      var v2b = Math.sqrt(v1a*v1a - v1b*v1b);
      var phi = -(Math.PI/2 - theta);

      v1x = v1b * Math.cos(theta);
      v1y = v1b * Math.sin(theta);
      v2x = v2b * Math.cos(phi);
      v2y = v2b * Math.sin(phi);
```

```
        colisao = 1;
    }

    // desenha partícula 1
    ctx.beginPath();
    ctx.strokeStyle = "blue";
    ctx.lineWidth = 1;
    ctx.arc(tc.cx(x1),tc.cy(y1),2,0,2*Math.PI,1);
    ctx.stroke();

    // desenha partícula 2
    ctx.beginPath();
    ctx.strokeStyle = "red";
    ctx.lineWidth = 1;
    ctx.arc(tc.cx(x2),tc.cy(y2),2,0,2*Math.PI,1);
    ctx.stroke();

    // finaliza simulação quando partícula 1 sai pela direita
    if (x1>wxf) clearInterval(intervalo);

}

</script>
```

Resultado:



MOVE PARA

v_{1a} = 1.00 v_{2a} = 0.00 v_{1b} = 0.01 v_{2b} = 1.00

θ = 89.23 φ = -0.77

O script listado não reproduz totalmente o resultado apresentado, mas apenas o que acontece no canvas, e mesmo assim sem a transformação de coordenadas necessária para que funcione. Ficará como exercício para o leitor a implementar o que falta.

O script inicia logo após a definição do canvas na porção HTML da listagem. A primeira providência é descobrir a largura (`width`) e a altura (`height`) do canvas, o que é feito buscando-os em seus atributos. O leitor interessado pode utilizar os termos apresentados em uma busca se desejar aprofundar seus conhecimentos sobre a estrutura interna do canvas.

Depois de obtidas as dimensões do canvas e a referência para o contexto gráfico, são definidos os limites do "mundo" da simulação (`wxi`, `wxf`, `wyi`, `wyf`) e instanciada a trasnformação de coordenadas. A seguir, são inicializadas as componentes das posições e velocidades das partículas para $t = 0$, bem como o raio do círculo a ser desenhado e o intervalo de tempo entre cada representação das partículas que se movem. Feito isso, é inicializado o temporizador, que chama a função `passo()` a cada 100 milissegundos.

A função passo faz o que dizem os seus comentários: (i) propaga as posições das partículas. (ii) verifica se ocorreu uma colisão; (iii) desenha as partículas nas novas posições e (iv) verifica se é o momento de encerrar a simulação.

Dessas ações, a que talvez mereça algum comentário adicional é a verificação da ocorrência de uma colisão. A primeira condição, que compara a distância entre as duas partículas com a soma dos seus raios, é bastante óbvia: se for satisfeita, terá havido uma colisão e um ângulo de espalhamento é gerado e todos os parâmetros que dele dependem calculados.

Para algumas condições, entretanto, as novas posições das partículas podem ser tais que a distância entre elas continua menor que soma dos raios, o que leva a um novo sorteio indevido. Para impedir isso, foi colocada a segunda condição, que deve somar-se à anterior (note o operador "E" (`&&`) lógico): a variável lógica `colisão` não pode ser verdadeira (ou 1, no caso). Note que a variável `colisao` é inicializada como falsa (ou 0, no caso) e modificada para verdadeira na primeira vez que a condição de colisão é satisfeita.

Exercícios

1. O exemplo apresentado não contém o código que define o objeto `TransCoord`, que faz a transformação do sistema de coordenadas da simulação para o sistema de coordenadas do canvas. Tente implementá-la sem recorrer a outros exemplos que a contém.

2. O exemplo apresentado não contém os botões nem as caixas de texto para os valores das velocidades antes e depois da colisão, nem para os ângulos de espalhamento. Implemente o código necessário para que apareçam e funcionem.

3. O ângulo de espalhamento θ da partícula incidente está relacionado com o parâmetro de impacto da colisão. Pesquise a definição de parâmetro de impacto e sua relação com o ângulo de espalhamento para colisões entre discos rígidos de tamanho finito. Modifique o script de modo que passe a sortear parâmetros de impacto ao invés de ângulos de espalhamento.

4. Utilize o resultado do exercício anterior e obtenha a distribuição do ângulo de espalhamento para alguns milhares de partículas de raio R incidindo sobre outra partícula também de raio R com parâmetros de impacto que obedeçam uma distribuição uniforme entre $-2R$ e $2R$.