P4 Simulación de Mallas Deformables y Roturas

Autores	Camilo Enguix Nadal, Pablo Gómez
Fecha	@May 12, 2022
Asignatura	L1, Simulación

1 Reflexiones

1 ¿Qué diferencias se observan entre los tipos de disposiciones (structural, shear, etc.) de muelles de la malla? ¿Cambia el comportamiento de la simulación? ¿En qué casos?

Working...

https://web.microsoftstream.com/video/ce97be70-bc10-4dd3-8837-cf3ec53bf0fa

Respuesta 1

Los enlaces entre nodos cambian. En el caso de structural, cada nodo se uno a los nodos que se encuentran en su norte, sud, este y oeste. Con shear conectamos los nodos con sus diagonales, obteniendo una malla entrelazada, y por tanto, más resistente. Al combinarlas, el resultado es un material mucho más preparado para colisiones. Que la distribución cambie, implica que la redistribución de fuerzas también lo haga, y esto produce distintos niveles de resistencia, en función de cuánta tensión debe soportar cada enlace de nuestro material.

2 ¿Son todas las disposiciones igual de costosas a nivel de tiempo de computación?

Working...

https://web.microsoftstream.com/video/d881f109-4d13-4964-873d-191b3e03c73c

Respuesta 2

Ligando con el punto anterior, conseguir una malla más resistente no sale gratis. El entramado de nodos produce una relación más compleja entre el material y la fuerza externa. No por nada, se reparte mejor la energía del impacto. Cuanto más efectiva es la redistribución del golpe, más carga computacional tiene la simulación. Por eso la combinación de las dos es la disposición más costosa computacionalmente multiplicándose por 2.

3 ¿Qué puede ocurrir si la esfera tiene un radio menor que la distancia entre los nodos de la malla? ¿Por qué?

Working...

https://web.microsoftstream.com/video/f39802de-2093-4844-8ac1-6246827abed6

Respuesta 3

Que la esfera atraviese la malla sin rebotar en ella. Esto se produce debido a que el radio es menor a la distancia entre nodos, por tanto, no se detecta la partícula y no se produce el rebote.

4 **¿Qué ocurre si la velocidad de la esfera es extremadamente alta cuando la malla es irrompible? ¿y cuándo no es irrompible? ¿Por qué?**

Working...

https://web.microsoftstream.com/video/a55c7862-7c6a-4d2a-816b-59a4e1c79892

Respuesta 4

En el primer caso, al llevar un velocidad muy alta, la malla no es capaz de generar una fuerza suficiente como para detener esta bola, por lo tanto atravesaría la malla.

Mientras que en el segundo caso, el agujero que dejaría la bola en la malla sería menor, ya que, a cuanta más velocidad más pequeño será el tamaño del agujero.

5 ¿Qué ocurre cuando la masa de la esfera es muy grande en relación a la de los nodos de la malla? ¿Por qué?

Working...

https://web.microsoftstream.com/video/7058f449-b5df-45c2-a13e-aa83264fc526

Respuesta 5

Al colisionar la bola con la malla, los muelles, al no poder soportar un peso tan grande, genera una gran inestabilidad en esta en el momento que la atraviesa. A parte de que el agujero generado por esta bola de mayor masa es más grande.

6 ¿En qué condiciones es la simulación inestable?

Hay varias opciones para que la simulación sea inestable:

Working...

https://web.microsoftstream.com/video/57021b0e-23f1-4bf3-bd55-df76c3f46081

Respuesta 6.1

En caso en el que la masa de los nodos es muy grande, lo que ocurre aquí es que al ser demasiado pesada, se rompe de su borde y cae por la fuerza de la gravedad.

Working...

https://web.microsoftstream.com/video/e967ced2-4ede-42c2-bf8e-bbfa23dc2c8a

Respuesta 6.2

Otro caso es cuando no existe amortiguación en los muelles lo que conlleva a que la malla nunca pueda recuperar su estado original. Al no haber amortiguación los muelles se encuentran en vibración constante.

2 Soluciones implementadas

2.1 En la clase DeformableSurface

- La siguiente función crea una matriz de nodos, cuyo número de filas y de columnas lo obtenemos de constantes externas. La función recibe la profundidad a la que se encuentra la red y la masa de los nodos de la misma. En los extremos de la malla los nodos tendrán una propiedad `clamped`, que los fijará, para resistir a las físicas que afectan al resto de nodos.

```
void createNodes(float surfacePosZ, float nodeMass)
  {
    for (int i = 0; i < _numNodesX; i++){
        for (int j = 0; j < _numNodesY; j++){
          float posx = (i*_lengthX/_numNodesX) - _lengthX/2;
          float posy = (j*_lengthY/_numNodesY) - _lengthY/2;

          PVector s = new PVector (posx, posy, surfacePosZ);
          PVector v = new PVector (0, 0, 0);
          boolean clamped = false;

          if ((i==0) || (i == _numNodesX -1) || (j==0) || (j == _numNodesY-1)) //Cuando se alguno de los nodos de las esquinas no
            clamped = true;

          _nodes[i][j] = new Particle(s, v, nodeMass, clamped);
        }
      }
   }
```

- A continuación, mostramos la función con la que definimos cómo enlazamos unos nodos con otros. En función del tipo de estructura elegido, podremos llegar a una organización u otra:
  - En el modo `STRUCTURAL`, los nodos se unen a sus vecinos cardinales, `[i±1][j]` e `[i][j±1]`.
  - Si se trata de `SHEAR`, los nodos a unir son los oblicuos, `[i+1][j+1]` e `[i-1][j+1]`.
  - Cuando se combinan, se lleva a cabo ambas conexiones.

  Al mismo tiempo, se define los coeficientes de las fuerzas de cada enlace y si son resistentes a roturas.

```
void createSurfaceSprings(float Ke, float Kd, float maxForce, float breakLengthFactor)
  {
    if(_isUnbreakable){
      maxForce = 0;
    }
    switch(_springLayout){
      case STRUCTURAL:
        for (int i = 0; i < _numNodesX; i++){
          for (int j = 0; j < _numNodesY; j++){
            if(i < _numNodesX - 1){//Añadimos muelle entre el nodo [i][j] con el nodo de su derecha.
              DampedSpring muelle1 = new DampedSpring(_nodes[i][j], _nodes[i+1][j], Ke, Kd, false, maxForce, breakLengthFactor);
              _springsSurface.add(muelle1);
            }
            if(j < _numNodesY - 1){//Añadimos muelle entre el nodo [i][j] con el nodo que se encuentra debajo suya.
              DampedSpring muelle2 = new DampedSpring(_nodes[i][j], _nodes[i][j+1], Ke, Kd, false, maxForce, breakLengthFactor);
              _springsSurface.add(muelle2);
            }
          }
        }
        break;

      case SHEAR:
       for (int i = 0; i < _numNodesX; i++){
          for (int j = 0; j < _numNodesY; j++){
            if(j != _numNodesY - 1){
               if(i == 0){
                DampedSpring muelle1 = new DampedSpring(_nodes[i][j], _nodes[i+1][j+1], Ke, Kd, false, maxForce, breakLengthFactor
                _springsSurface.add(muelle1);
              }
              else if(i < _numNodesX - 1){
                DampedSpring muelle1 = new DampedSpring(_nodes[i][j], _nodes[i+1][j+1], Ke, Kd, false, maxForce, breakLengthFactor
                _springsSurface.add(muelle1);
                DampedSpring muelle2 = new DampedSpring(_nodes[i][j], _nodes[i-1][j+1], Ke, Kd, false, maxForce, breakLengthFactor
                _springsSurface.add(muelle2);
              }
              else if(i == _numNodesX - 1){
                DampedSpring muelle1 = new DampedSpring(_nodes[i][j], _nodes[i-1][j+1], Ke, Kd, false, maxForce, breakLengthFactor
                _springsSurface.add(muelle1);
```

```
            }
          }
        }
      }
      break;

    case STRUCTURAL_AND_SHEAR:
      for (int i = 0; i < _numNodesX; i++){
        for (int j = 0; j < _numNodesY; j++){
          //STRUCTURAL
          if(i < _numNodesX - 1){//Añadimos muelle entre el nodo [i][j] con el nodo de su derecha.
            DampedSpring muelle1 = new DampedSpring(_nodes[i][j], _nodes[i+1][j], Ke, Kd, false, maxForce, breakLengthFactor);
            _springsSurface.add(muelle1);
          }
          if(j < _numNodesY - 1){//Añadimos muelle entre el nodo [i][j] con el nodo que se encuentra debajo suya.
            DampedSpring muelle2 = new DampedSpring(_nodes[i][j], _nodes[i][j+1], Ke, Kd, false, maxForce, breakLengthFactor);
            _springsSurface.add(muelle2);
          }
          //SHEAR
          if(j != _numNodesY - 1){
             if(i == 0){
              DampedSpring muelle1 = new DampedSpring(_nodes[i][j], _nodes[i+1][j+1], Ke, Kd, false, maxForce, breakLengthFactor
              _springsSurface.add(muelle1);
            }
            else if(i < _numNodesX - 1){
              DampedSpring muelle1 = new DampedSpring(_nodes[i][j], _nodes[i+1][j+1], Ke, Kd, false, maxForce, breakLengthFactor
              _springsSurface.add(muelle1);
              DampedSpring muelle2 = new DampedSpring(_nodes[i][j], _nodes[i-1][j+1], Ke, Kd, false, maxForce, breakLengthFactor
              _springsSurface.add(muelle2);
            }
            else if(i == _numNodesX - 1){
              DampedSpring muelle1 = new DampedSpring(_nodes[i][j], _nodes[i-1][j+1], Ke, Kd, false, maxForce, breakLengthFactor
              _springsSurface.add(muelle1);
            }
          }
        }
      }
    break;
  }
}
```

- Para actualizar las físicas de la simulación, recorremos todos los nodos, actualizándolos uno a uno, en la función 3.2 de este documento.

```
void update(float simStep)
  {
    int i, j;

    for (i = 0; i < _numNodesX; i++)
      for (j = 0; j < _numNodesY; j++)
        if (_nodes[i][j] != null)
          _nodes[i][j].update(simStep);

    for (DampedSpring s : _springsSurface)
    {
      s.update(simStep);
      s.applyForces();
    }

    for (DampedSpring s : _springsCollision.values())
    {
      s.update(simStep);
      s.applyForces();
    }
  }
```

Primero calculamos el valor de las variables usadas para calcular la fuerza elástica y la fuerza de amortiguación.

- La fuerza elástica sigue la fórmula del muelle, ya vista en clase: Constante elástica por la diferencia entre la elongación actual y la de reposo.
- La fuerza de amortiguación mantiene la dirección de la velocidad de elongación y se multiplica por la constante de amortiguación.

A parte, concretamos el comportamiento que deben tener los nodos, cuando decimos que la malla se ha roto (`isBroken`), donde _F = (0, 0, 0), o que el propio enlace se rompa (`breakIt`), cuando la fuerza y/o la elongación, superen las marcas de tolerancia máxima de la red.

```
void update(float simStep)
  {
    float eanterior = _e.mag();
    _e = PVector.sub(_p2.getPosition(), _p1.getPosition());
    _l = _e.mag();
    _eN = _e.copy();
    _eN.normalize();
    float dist = _l - _lr;

     _v = ( _l - eanterior) / simStep; // elongación actual

     float Fd =  _v * _Kd; // F damping (amortiguación) = direcc. * elongacion_anterior * constante
     float Fe = _Ke * dist; // F eléctrica = direcc. * (_Ke * l_actual - l_reposo)

      if(_repulsionOnly){
        if(_lr < _l){
          _F.set(0,0,0);
        }
      }
      _F = PVector.mult(_eN,Fd + Fe); //Fe = Fe + Fd

     if(_FMax != 0){
       if(_F.mag() > _FMax){
         breakIt();
       }

       if(_l > _lMax * _lr){
         breakIt();
       }

     }
     if(isBroken()){
       _F.set(0,0,0);
     }
  }
```

- La última función recorre todos los nodos y saca la distancia entre cada nodo y la esfera. Si esa distancia es menor que el radio de la pelota, se detecta como una colisión y ese nodo entra dentro del diccionario de nodos que están en colisión con la esfera, `_springsCollison`.

  Si no estamos en el caso en el que la distancia es menor que el radio, entonces se elimina del diccionario.

```
void avoidCollision(Ball b, float Ke, float Kd, float maxForce, float breakLengthFactor)
  {
    for (int i = 0; i < _numNodesX; i++){
          for (int j = 0; j < _numNodesY; j++){
            PVector distancia = PVector.sub(b.getPosition(), _nodes[i][j].getPosition());

            if(distancia.mag() < b.getRadius()){
              if(_springsCollision.get("" + _nodes[i][j].getId()) == null){
                _springsCollision.put("" + _nodes[i][j].getId(), new DampedSpring(b, _nodes[i][j], Ke, Kd, true, maxForce, breakLe
              }
            }
            else{
               _springsCollision.remove("" + _nodes[i][j].getId());
            }
          }
    }
  }
```