Connecting Objects

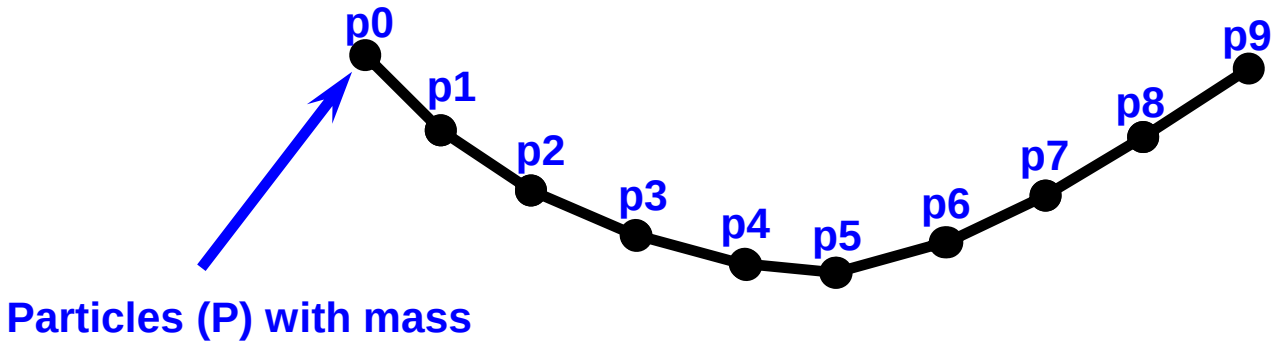
CS-116B: Computer Graphics Algorithms
Spring 2018

Connecting objects

Connecting objects involves rigid body particles to simulate rope or cloth using the following:

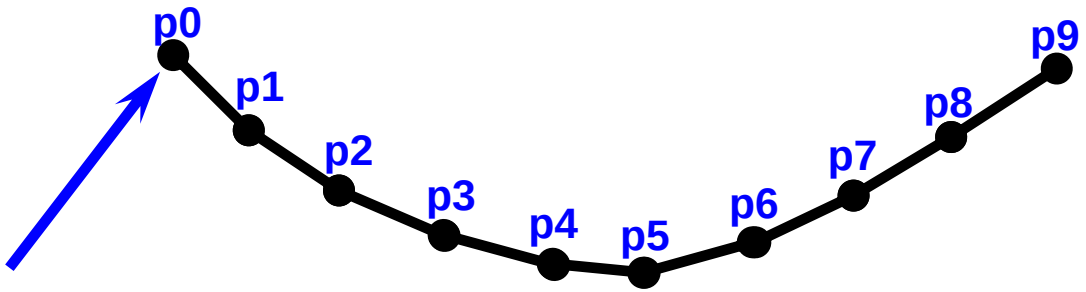
- Hooke’s law, F=kx
- gravity
- rigid body particles (with mass)
- springs (no mass)

Connecting objects



Source: *Physics for Game Developers*, p. 261

Connecting objects

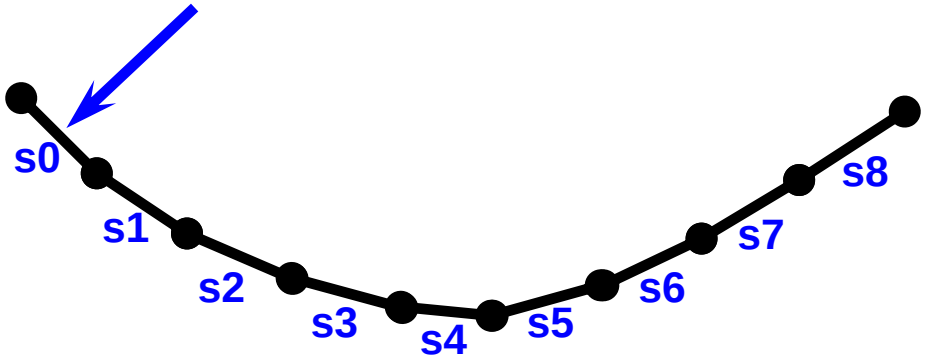


Each particle is pinned to the next particle but allows rotation. This is called a “Pinned joint”.

Source: *Physics for Game Developers*, p. 275

Connecting objects

Weightless, invisible springs (S) connect the particles together.



Source: *Physics for Game Developers*, p. 261

Particle code for rope simulation: Constants and Array definitions

```
#define _NUM_OBJECTS      10
#define _NUM_SPRINGS      9
#define _SPRING_K         1000
#define _SPRING_D         100

Particle  Objects[_NUM_OBJECTS];
Spring    Springs[_NUM_SPRINGS];
```

Source: *Physics for Game Developers*, pp. 260

Particle code for rope simulation: structure definition

```
typedef struct _Spring
{
  int    End1;
  int    End2;
  float  k;
  float  d;
  float  InitalLength;
} Spring, *pSpring;
```

End1:
“A reference to the first particle to which the spring is connected.”

End2:
“A reference to the second particle to which the spring is connected.”

k:
“The spring constant.”

d:
“The damping constant.”

InitialLength:
“The unstretched length of the spring.”

Source: *Physics for Game Developers*, pp. 260

Particle code for rope simulation: Initialize

```
bool Initialize (void)
{
  Vector  r;
  int i;
  Objects[0].bLocked = true;
// Initialize particle locations from left to right.
  for (i=0; i < _NUM_OBJECTS;  i++)
  {
    Objects[i].vPosition.x = _WINWIDTH / 2 + Objects[0].fLength * i;
    Objects[i].vPosition.y = _WINHEIGHT / 8;
  }
// Initialize springs connecting particles from left to right.
  for (i = 0; i <= _NUM_SPRINGS; i++)
  {
    Spring[i].End1 = i;
    Springs[i].End2 = i + 1;
    r = OBjects[i+1],vPosition - Objects[i].vPosition;
    Springs[i].InitialLength = r.Magnitude();
    Springs[i].k = _SPRING_K;
    Springs[i].d = _SPRING_D;
  }
  return true;
}
```

Source: *Physics for Game Developers*, pp. 261-262

Particle code: Update Simulation

```
bool UpdateSimulation (void)
{
  double  dt = _TIMESTEP;
  int   i;
  double  f, dl;
  Vector  pt1, pt2;
  int   j;
  Vector  r;
  Vector  F;
  Vector  v1, v2, vr;
// Initialize the spring forces on each object to zero
  for (i = 0; i <_NUM_OBJECTS; i++)
  {
    Objects[i].vSprings.x = 0;
    Objects[i].vSprings.y = 0;
    Objects[i].vSprings.z = 0;
  }
```

Source: *Physics for Game Developers*, pp. 261-262

Particle code for rope simulation: Update Simulation (continued)

```
// Calculate all spring forces based on positions of connected objects.
  for (i = 0; i < _NUM_SPRINGS; i++)
  {
    j = Springs[i].End1;
    pt1 = Objects[j].Vposition;
    v1 = Objects[j].vVelocity;
    j = Springs[i].End2;
    pt2 = Objects[j].Vposition;
    v2 = Objects[j].vVelocity;
    vr = v2 - v1;
    r = pt2 - pt1;
    dl = r.Magnitude() - Springs[i].InitialLength;
    f = Springs[i].k * dl; // - means compression, + means tension
    r.Normalize();
    F = (r * f) + (Springs[i].d * (vr * r)) * r;
    j = Springs[i].End1;
    Objects[j].vSprings += F;
    j = Springs[i].End2;
   Objects[j].vSprings -= F;
  }
// [...] Integrate equations of motion as usual
// [...] Render the scene as usual
}
```

Source: *Physics for Game Developers*, pp. 261-262