Computer Graphics: Animation - 2

Table of contents

Introduction

Particle systems

Collision

Mass-spring system

Fluid simulation

OpenGL example

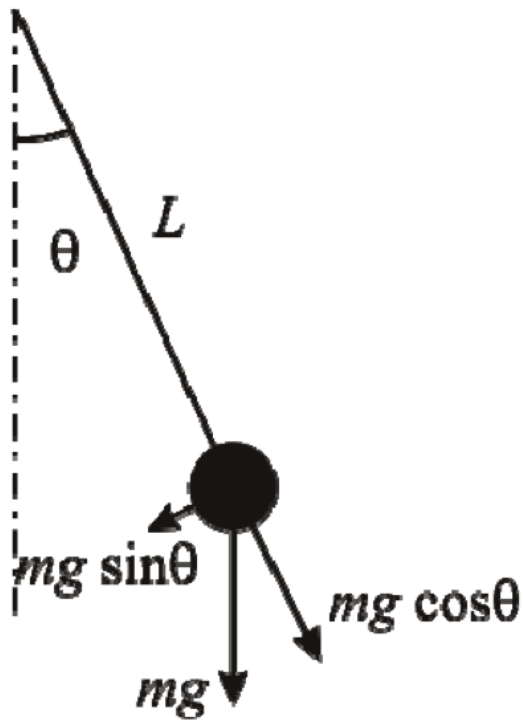
Physically based animation

Physically based animation is suitable for inanimate objects and effects when there is a numerically tractable physical model.
Examples:

- Particle systems for simulating cloud, fire, water, smoke
- Meshes of mass connected by springs for animating deformable objects
- Simulation of fluid by solving differential equation (e.g. Navier Stokes for incompressible flows)

Introductory example: the simple pendulum

A pendulum consists of a mass m hanging from a string of length L and fixed at a pivot point. When displaced to an initial angle and released, the pendulum will swing back and forth with periodic motion.



The simple pendulum

Applying Newton's second law:

$$F = -mg \sin\theta = ma$$

The acceleration a is related to the change in angle θ by:

$$a = \frac{d^2 s}{dt^2} = L\frac{d^2\theta}{dt^2}$$

This gives the equation of motion for the pendulum:

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\sin\theta = 0$$

This equation is an Ordinary Differential Equation (ODE). It can be solved by a numerical algorithm.

The simple pendulum

The equation of motion for the pendulum is:

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\sin\theta = 0$$

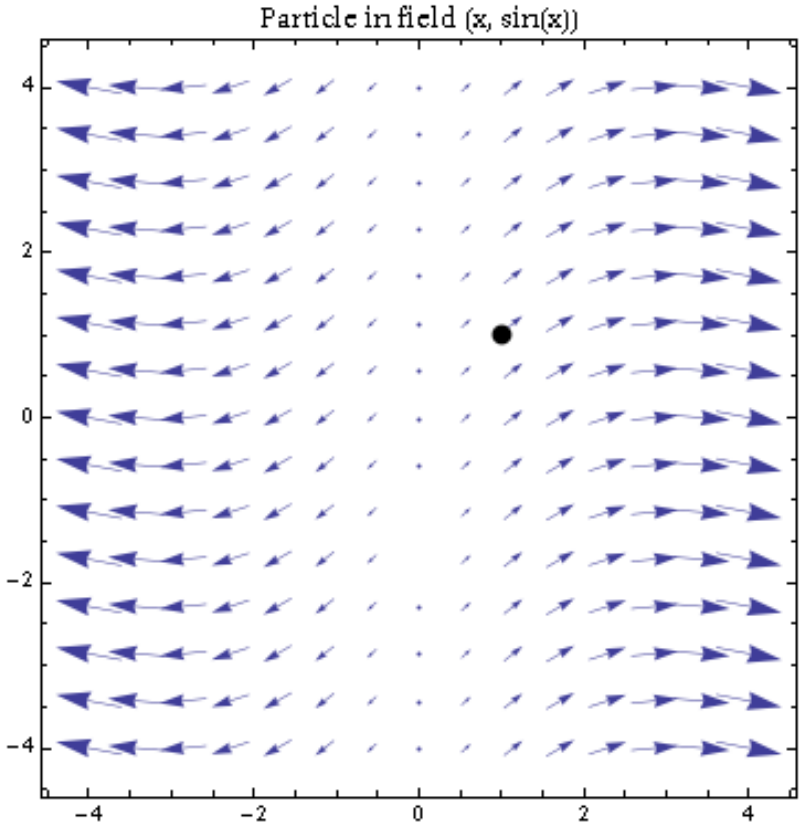
An alternative to the numerical solution is to simplify this equation by making the assumption that angular displacement are small, therefore: $\sin\theta = \theta + O(\theta^3)$. The ODE is re-arranged in:

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\theta = 0$$

The simple harmonic solution is: $\theta(t) = \theta_0 \cos(\omega t)$, where $\omega = \sqrt{\frac{g}{L}}$ is the natural frequency of the motion.

Particle in velocity field

Let us consider a single particle in two dimensions moving in a flow field. The particle is characterized by its position $\mathbf{p} = (x, y)$ and its speed (or velocity) $\mathbf{v} = (v_x, v_y) = \frac{d\mathbf{p}}{dt}$.
Let us assume that the particle is in a flow field $\mathbf{g}(\mathbf{p}, t)$ that dictates the particle velocity, i.e. $\mathbf{v} = \mathbf{g}(\mathbf{p}, t)$



Particle in velocity field

The particle velocity is dictated by the flow field:
$\mathbf{v} = \frac{d\mathbf{p}}{dt} = \mathbf{g}(\mathbf{p}, t)$, given its initial position.
This corresponds to a system of first order ODE.
Position of the particle at time t is obtained by integrating the differential equation:

$$\mathbf{p}(t) = \mathbf{p}(t_0) + \int_{t_0}^{t} \mathbf{g}(\mathbf{p}, t)dt$$

Numerical solution

There are various methods to solve numerically a first order ODE.
The easiest is Euler's method:

$$\mathbf{p}(t + dt) = \mathbf{p}(t) + dt\mathbf{g}(\mathbf{p}, t)$$

Where dt is a given integration time-step.
The problems of Euler's method are:

- its inaccuracy (the error is proportional to $(dt)^2$)
- its unstability (it requires very small integration time-step)

Alternative methods to solve ODE are:

- Runge-Kutta
- Adams
- Adaptive stepsize

Example: Euler's method

Example of pseudo-code to animate a particle in a flow field using Euler's method:

```
AnimateParticleInFlowField(p0, nt, dt) {
 {x,y,z} = p0;
 t = 0.0;
 for iter=0 to nt {
  DrawParticle({x,y,z});
  // Update the particle position:
  {x,y,z} = {x,y,z} + dt * g({x,y,z},t);
  t = t + dt;
 }
}
```

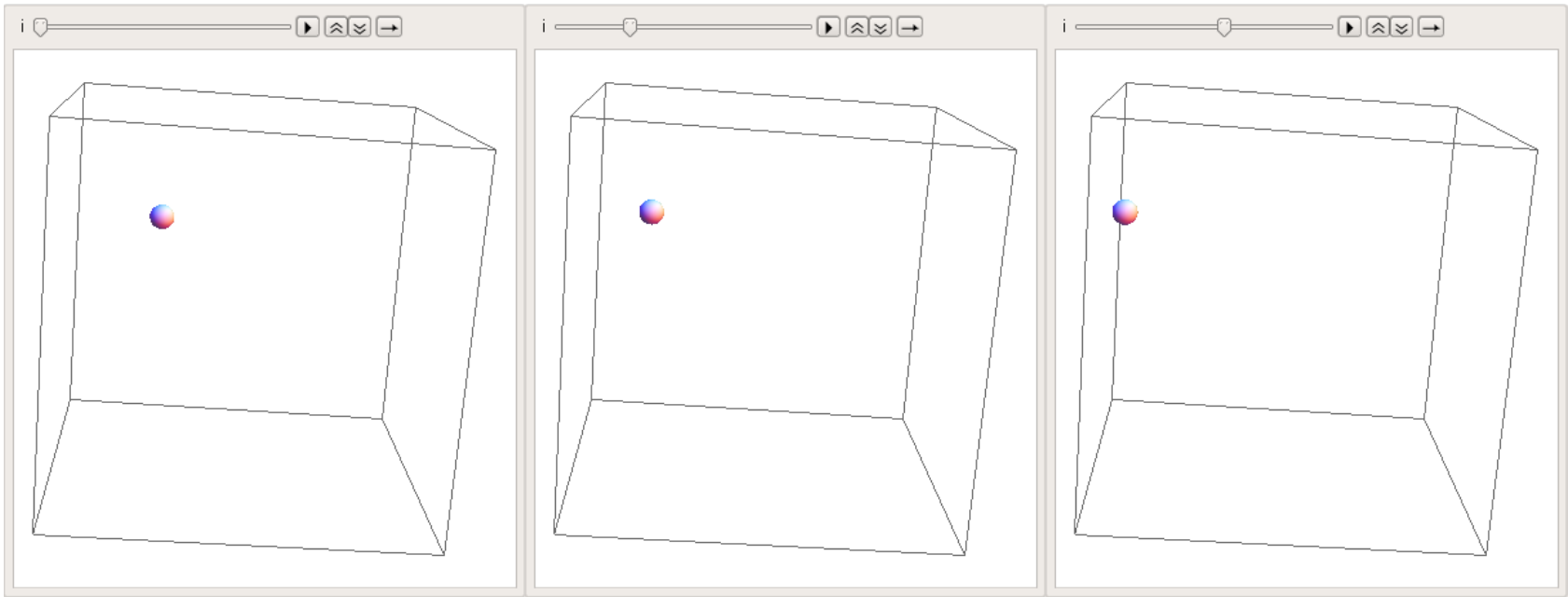
Numerical solution

ODE are easily solved numerically by systems like Maple or Mathematica. For example in Mathematica, we can solve for the position $\mathbf{p}(t)$ of a particle in the flow field $\mathbf{g}(\mathbf{p}, t)$ with:

```
T=1.0; initialPosition={1.0,1.0};
g[p_,t_]:={p[[1]],Sin[p[[2]]]};
eqns = {x'[t]==g[{x,y},t][[1]], y'[t]==g[{x,y},t][[2]]};
sols = NDSolve[{eqns,x[0]==initialPosition[[1]],
y[0]==initialPosition[[2]]},{x,y},{t,0,T}];
```

Example: particle in a flow field

The pictures below illustrate the position at various time of a particle in the flow field $g(\mathbf{p}, t) = (x, y, \sin(x))$:



Particle in a force field

A particle in a force field is accelerated according to Newton's second law:

$$m\frac{d^2\mathbf{p}}{dt^2} = \mathbf{f}((p),(v),t)$$

The mass m of the particle describes the inertial properties of the particle: lighter particles are easier to move than heavier particles. The force field $\mathbf{f}$ may depend on several variables such as the time t, and the position $\mathbf{p}$ or speed $\mathbf{v}$ of the particle.

Particle in a force field

The ODE controlling the particle motion is of second order but can be converted to a system of first order ODE by considering the change of variable $\mathbf{y} = (\mathbf{p}, \mathbf{v})$. The system of ODE to solve becomes:

$$\frac{d\mathbf{y}}{dt} = (\frac{d\mathbf{p}}{dt}, \frac{d\mathbf{v}}{dt}) = (\mathbf{v}, \frac{\mathbf{f}}{m})$$

This can be solved with the same techniques as discussed above.

Example: Animation of a particle in a force field with Euler's method

Example of pseudo-code to animate a particle in a force field, given its initial position and speed and using Euler's method:

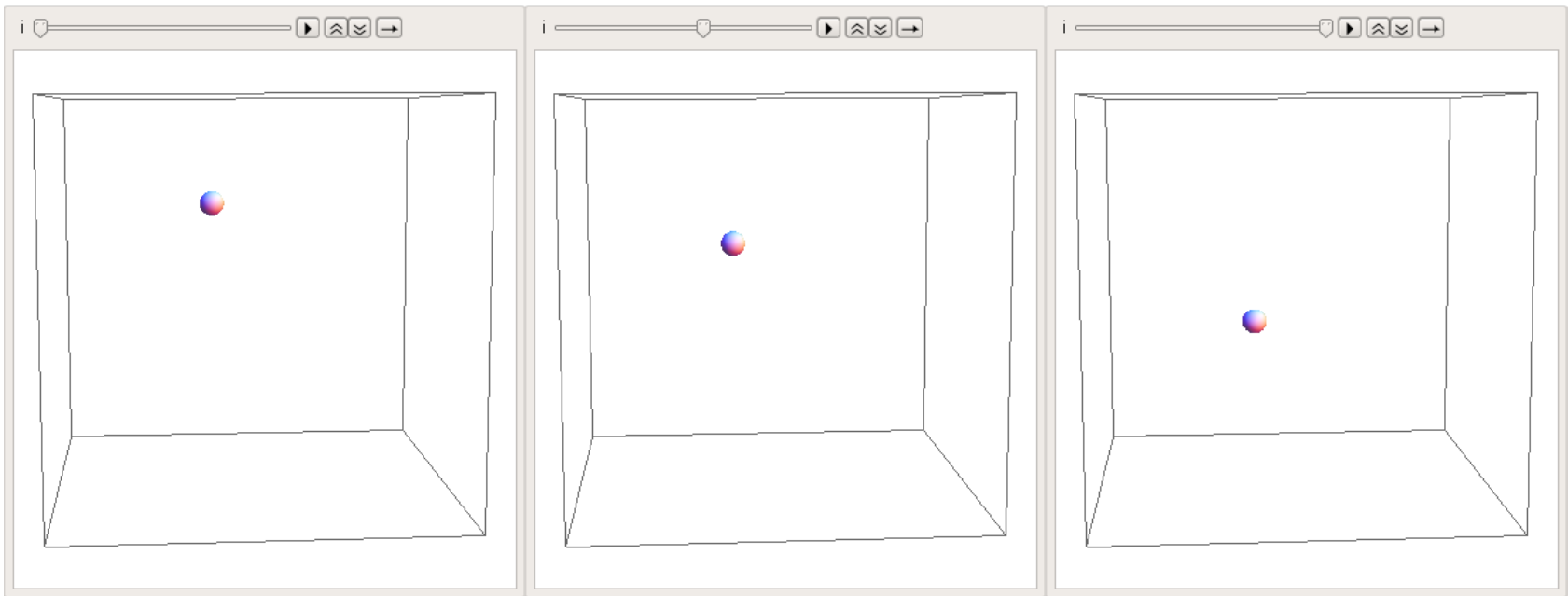
```
AnimateParticleInForceField(p0, v0, nt, dt) {
 {x,y,z} = p0;
 {vx,vy,vz} = v0;
 t = 0.0;
 for iter=0 to nt {
  DrawParticle({x,y,z});
  // Update the particle position:
  ...
 }
}
```

Example: Animation of a particle in a force field with Euler's method

```
// Update the particle position:
{x1,y1,z1} = {x,y,z};
{vx1,vy1,vz1} = {vx,vy,vz};
{x,y,z} = {x1,y1,z1} + dt * {vx1,vy1,vz1};
// Update the particle speed:
{vx,vy,vz} = {vx1,vy1,vz1} +
     dt / m * f({x1,y1,z1},{vx1,vy1,vz1});
t = t + dt;
```

Example: particle in a force field

The pictures below illustrate the position at various time of a particle accelerated by the gravity field. The initial position of the particle is random and it has no initial velocity:



Particle systems

Instead of considering a single particle in a force field, we can consider a system of particles. Interaction between particles can be considered in the force field (e.g. as repulsive forces) or ignored for simplicity.
System of particles can be used to animate amorphous objects like cloud, fire, water.
System of particles are also used as basis to mass-spring mesh to animate deformable objects (like cloth or hairs).

Collision

Simulating motions with collisions requires:

- Collision detection
- Collision response

We will start by the simple case of collision detection and response between a particle and a plane. It can be used as a basis for modeling collision between more complicated shapes.

Collision between a particle and a plane

Collision between a particle and a plane can be detected by evaluating the signed distance function to the plane at the particle's position.

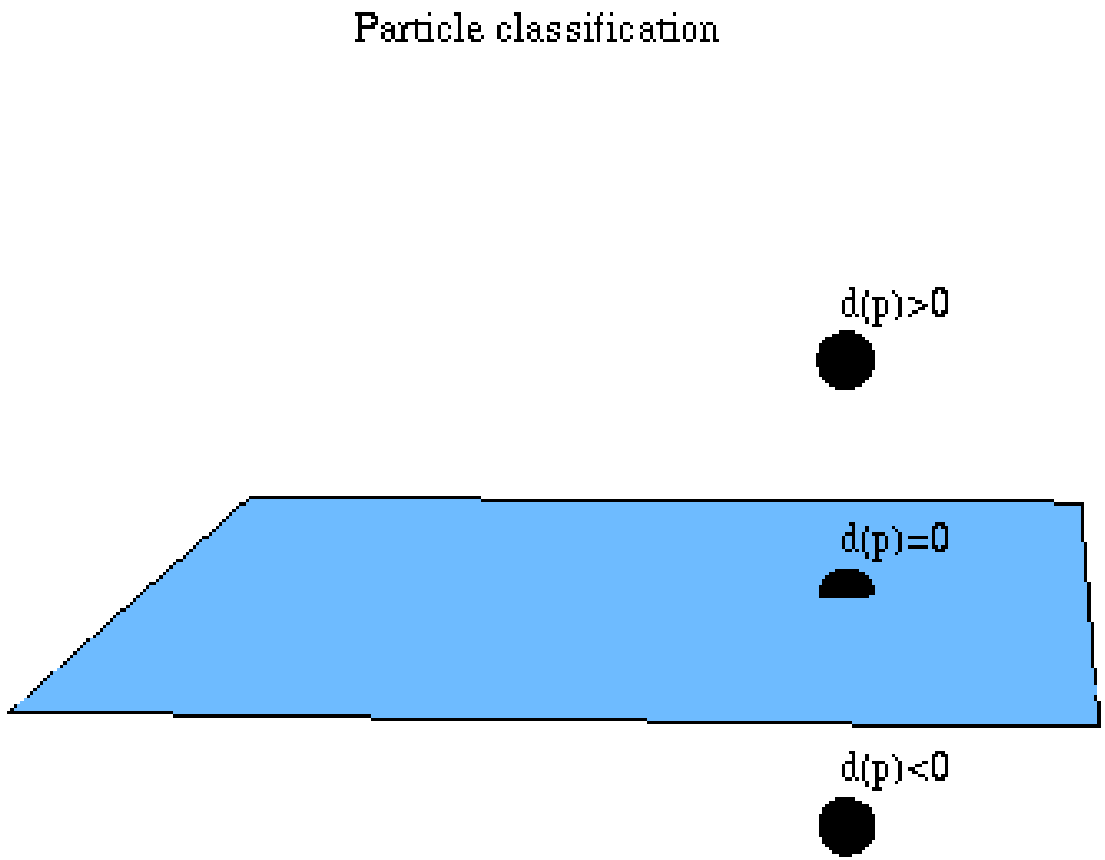
Assume the plane to be defined by its normal $\mathbf{n}$ and a point on the plane $\mathbf{p}_0$. The signed distance function to the point is given by $d(\mathbf{p}) = (\mathbf{p} - \mathbf{p}_0).\mathbf{n}$

Three cases are possible:

- The particle is on the surface: $d(\mathbf{p}) = 0$. An intersection is occurring
- The particle is on the positive side of the plane (in the direction pointed to the normal): $d(\mathbf{p}) > 0$. No intersection
- The particle is on the negative side of the plane: $d(\mathbf{p}) < 0$. An intersection occurred

The first case is rare. Usually: $d(\mathbf{p}(t)) > 0$ and $d(\mathbf{p}(t + dt)) < 0$, for some t. The intersection time and point can then be calculated by finding the intersection between the plane and the segment $\mathbf{p}(t + dt) - \mathbf{p}(t)$.

Collision between a particle and a plane



Collision response

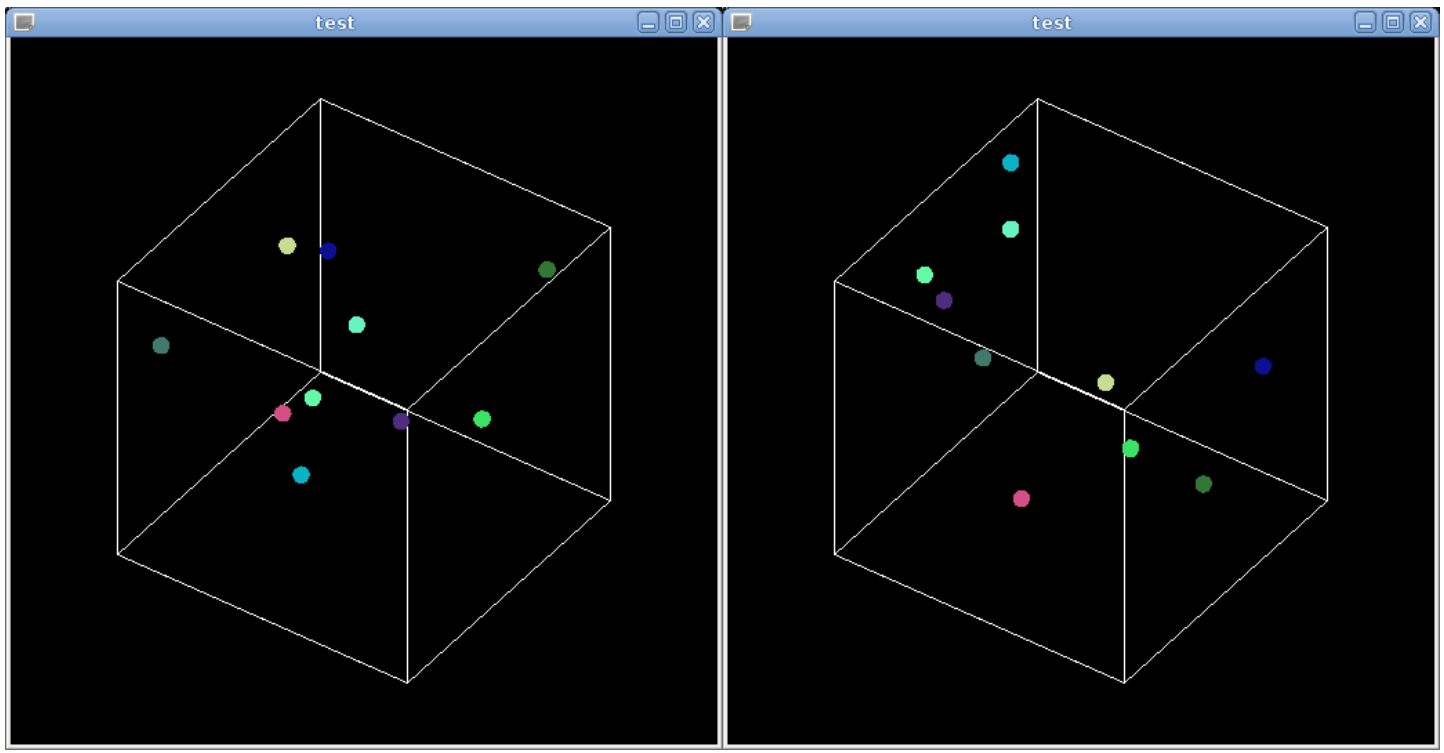
A simple model is an instantaneous exchange of impulse between the particle and the plane. Denote $\mathbf{v}_1$ the velocity just before impact, and $\mathbf{v}_2$ the velocity just after impact. In this simple model these two velocities are related by:

$$\mathbf{n}.\mathbf{v}_2 = -\epsilon \mathbf{n}.\mathbf{v}_1$$

Where $\epsilon \in [0, 1]$ control the impulse exchange.

Example: particle system contained in a box

The pictures below are taken from the animation of a particle system. Collision of the particles with the faces of the box are considered.



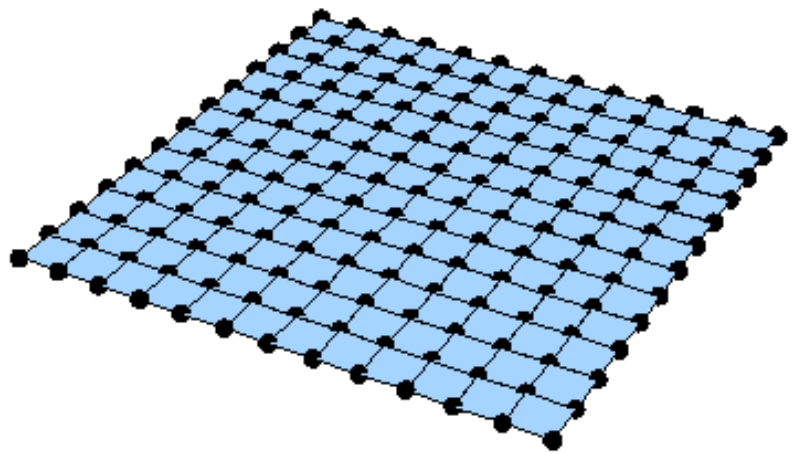
Collision: the general case

In the general case, objects are represented by triangle meshes made up of vertices (points) connected by triangles. Intersection between an object and its environment can be done by checking triangle - triangle intersections.
In order to minimize the number of intersection tests, (simpler) bounding volumes are computed for each object. Typical bounding volumes are boxes or spheres. Intersection tests for such objects are easier to perform.
More precise triangle - triangle intersections are computed only when the objects' bounding volumes intersect.

Mass-spring system

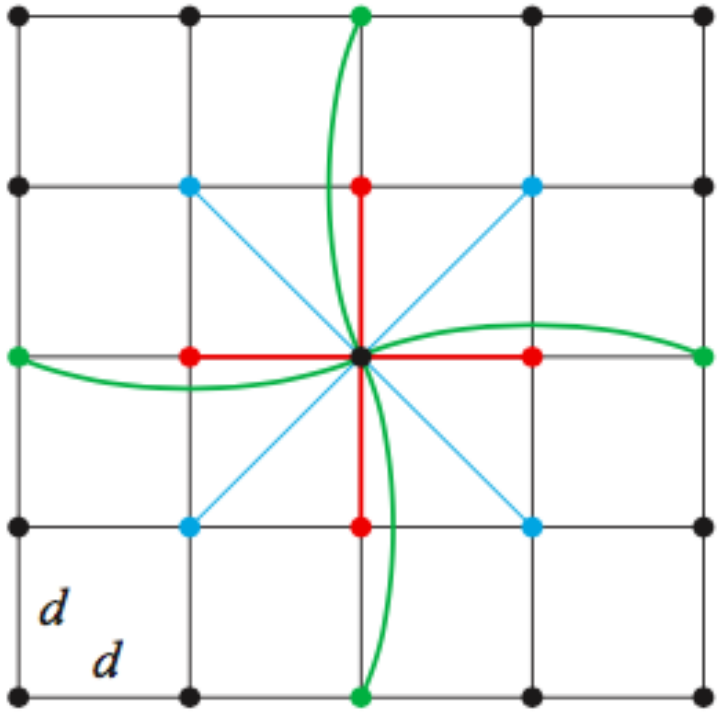
Mass-spring systems are used to model deformable objects, such as cloth, hairs.

In this model, particles are organized on a mesh where edges are "made" of springs to simulate deformation. An example of such a mesh is illustrated below:



Mass-spring system

To prevent weird behaviors in the simulation, it is possible to consider additional neighbors:



Mass-spring system

Forces are applied to each mass (particle) in the mesh, and the mesh particles will enter in motion under this force field. Applied forces can be divided in external and internal forces.
External forces typically include:

- Gravity
- Collision with the environment
- Force of wind (for cloth)

Internal forces are caused by the springs. The force applied to a particle connected by a string to a fixed position is given by Hooke's law:

$$\mathbf{F} = k(||\mathbf{y} - \mathbf{x}|| - d)\frac{\mathbf{y} - \mathbf{x}}{||\mathbf{y} - \mathbf{x}||}$$

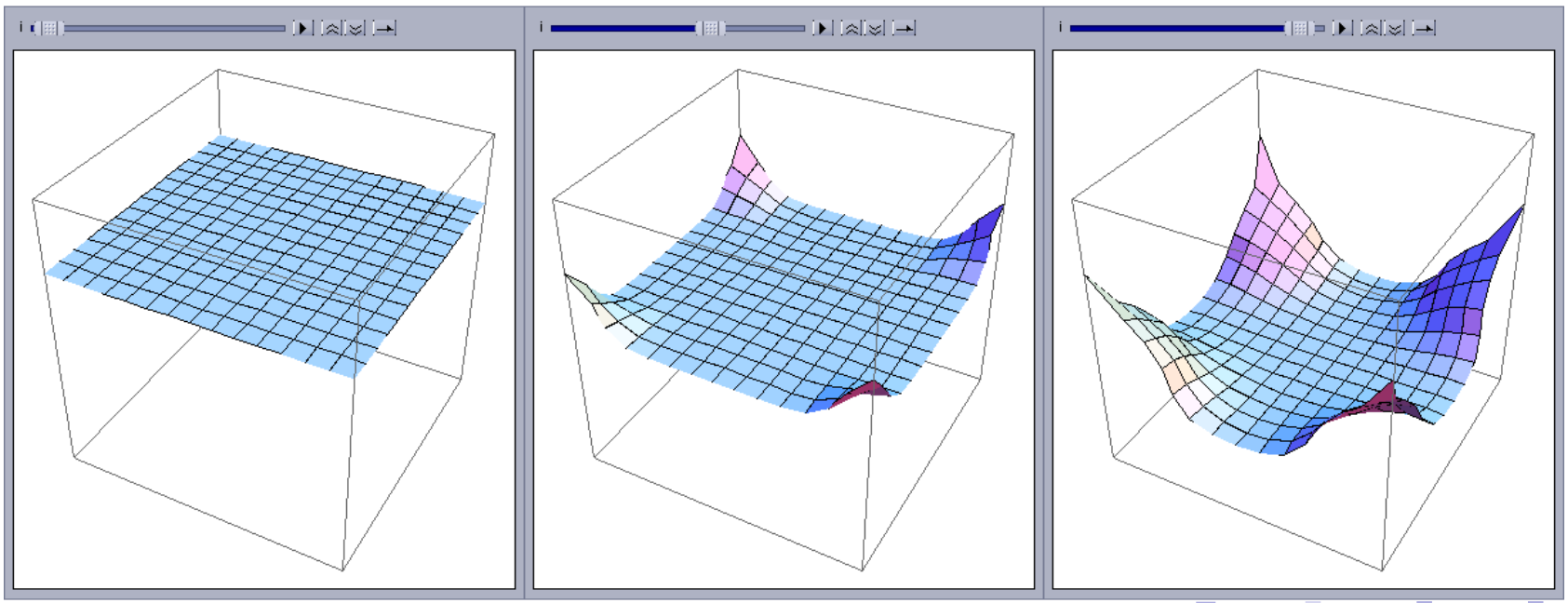
where $\mathbf{x}$ is a vertex on the grid, $\mathbf{y}$ one of its neighbors, d the distance between the vertices at rest, and k corresponds to the spring constant.

Mass-spring system

Simulation of deformable object, like clothes or a flag in the wind, can then be done with a procedure similar to the one used for animating particles in a force field.

Internal forces due to springs need to be added by considering for each mass in the mesh its neighbors, and compute the displacement of the mass from its rest position.

Some frames of the animation of a flag attached at its four corners in a gravity field are illustrated below:



Fluid simulation

Fluid simulation is increasingly popular for generating realistic animation of smoke, water, fire, . . .
Given input configuration for the fluid and the scene geometry, the simulation will evolve the fluid motion at each time-step.
Models used for fluid simulation range from solving complicated Partial Differential Equation (PDE) like the Navier-Stokes equations for high-quality animation (visual effects, movies) to simplified PDE or particle system models for real-time animation (games).

Height field models

Height field models simulate fluid by tracking the altitude of the fluid over its rest position on a two dimensional space.
One such model is given by the shallow water equations.
These equations can be obtained from the Navier Stokes equations by making the assumption that the horizontal length scale is greater than the vertical length scale. The shallow water equations are typically used to model tsunamis or wave propagation.

Water waves

The shallow water equations are complicated PDE and not suitable for real-time animation. They can be further simplified by ignoring the non-linear part of the equations, giving the wave equation:

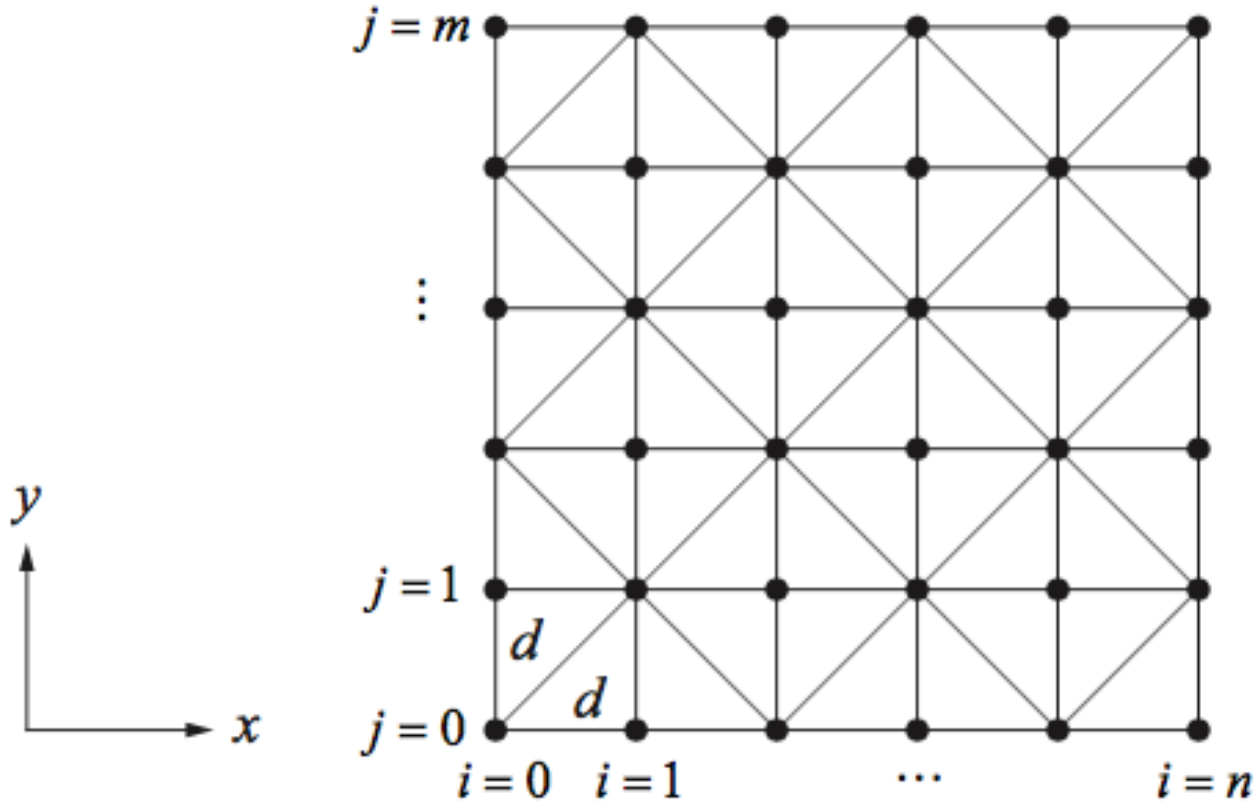
$$\frac{\partial^2 z(x,y,t)}{\partial t^2} = c^2 \Delta z(x,y,t)$$

where c acts as the velocity of the wave in the fluid and Δ is the Laplacian in Euclidean coordinates, i.e. $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$.
To make the result more realistic, dissipation of energy can be accounted by adding a viscuous term:

$$\frac{\partial^2 z(x,y,t)}{\partial t^2} = c^2 \Delta z(x,y,t) - \mu \frac{\partial z}{\partial t}$$

One way to solve this equation is by approximating the derivatives using finite differences on a two-dimensional grid.

Discretization: computational grid



Regular grid on which the wave equation is solved. The triangles are used for visualizing the resulting surface (using for example OpenGL).

Discretization of the derivatives

The derivatives of a function $z(x)$ can be approximated by finite difference:

$$z'(x) \simeq \frac{z(x+\epsilon) - z(x-\epsilon)}{2\epsilon}$$

$$z''(x) \simeq \frac{z(x+\epsilon) - 2z(x) + z(x-\epsilon)}{\epsilon^2}$$

where ϵ is a small quantity.

Discretization of the wave equation

Using the previous approximations of the derivatives in the wave equation, the height displacement $z(i, j, k+1)$ at the next time step becomes:

$$\begin{aligned} z(i,j,k+1) = & \frac{4 - 8c^2\Delta t^2/d^2}{\mu\Delta t + 2} z(i,j,k) + \frac{\mu\Delta t - 2}{\mu\Delta t + 2} z(i,j,k-1) \\ & + \frac{2c^2\Delta t^2/d^2}{\mu\Delta t + 2} \{z(i+1,j,k) + z(i-1,j,k) \\ & + z(i,j+1,k) + z(i,j-1,k)\} \end{aligned}$$

where $d = \Delta x = \Delta y$ is the distance between neighbors in the regular grid (along the x and y axes) and Δt is the simulation time-step.

Additional conditions

Some additional conditions are needed to run the simulation:

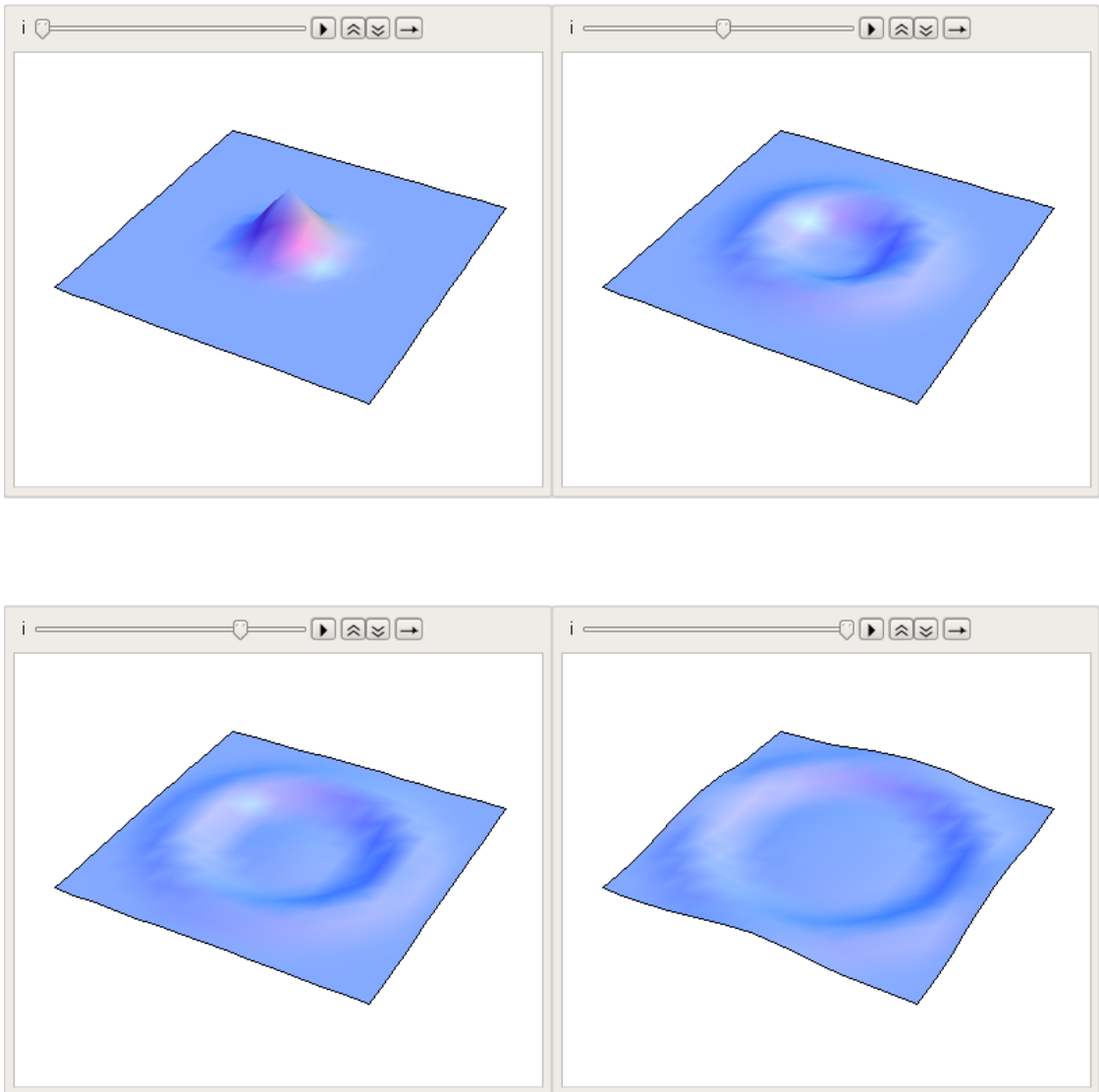
- Boundary condition: assume $z = 0$ at the boundary of the grid,
- Initial condition: specify some initial displacement $z(i, j, 0)$ and $z(i, j, 1)$ at the beginning of the simulation,
- Time-step condition: given the wave speed c, the time-step increment should satisfy: $\Delta t < \frac{\mu + \sqrt{\mu^2 + 32c^2/d^2}}{8c^2/d^2}$ to prevent the solution to diverge.

Animation of the wave equation

```
WaveAnimate(z0, z1, nt, dt) {
 for k = 1 to nt {
   for each (i,j) {
     Compute z(i,j,k+1)
   }
   Draw all triangles using z(i,j,k+1)
     as the height of the node (i,j)
   z0 <- z1
   z1 <- z(k+1)
 }
}
```

Example: wave propagation in fluid

A water drop initiating a wave:



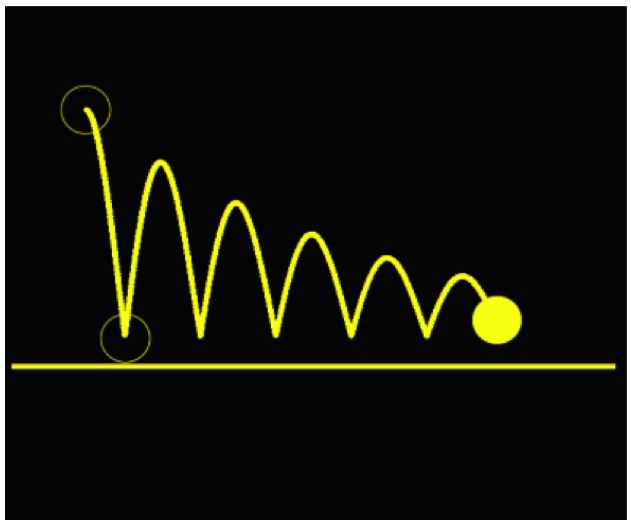
OpenGL example: animation of a bouncing ball

The model for the animation of a bouncing ball is given by:

$$x = x_{min} + (x_{max} - x_{min})p$$

$$y = A\, abs(\cos(\omega p))e^{-kp}$$

Where p is a process parameter changing from 0 to 1, k controls the dissipation of energy and A the amplitude of the motion. The figure below illustrates the trajectory of the ball over a sequence of time-steps:



OpenGL example: control parameters

```
// Global variables
// Parameters controlling the animation
GLfloat g_xp=-4.0, g_yp=3.0;
GLfloat g_param=0.0, g_delta_param=0.002;
GLfloat g_fps = 24.0; // frames per second
GLfloat g_amplitude = 3.0;
```

OpenGL example: the main function

```
int main(int argc, char** argv) {
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
 glutInitWindowSize(600,600);
 glutCreateWindow("Bouncing ball");

 InitGL();

 glutDisplayFunc(Display);
 glutKeyboardFunc(HandleKeyboardEvents);
 glutReshapeFunc(Reshape);
 glutIdleFunc(HandleIdleEvent);

 glutMainLoop();

 return 0;
}
```

OpenGL example: HandleIdleEvent function

```
static void HandleIdleEvent(void) {
 while ((clock() - g_previous_time)/CLOCKS_PER_SEC
   < 1.0/g_fps) ;

 g_xp = -4.0 + 8.0 * g_param;
 g_yp = g_amplitude * fabs(cos(6.0*g_param*M_PI))
     * exp(-1.5*g_param);
 g_param += g_delta_param;
 if (g_param > 1.0) g_param=0.0;
 g_previous_time = clock();
 glutPostRedisplay();
}
```

OpenGL example: Display function

```
static void Display(void) {
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 gluLookAt(0,1,6,0,1,0,0,1,0);

 glColor3f(1.0,0.0,0.0);
 // draw the ground
 glLineWidth(5.0);
 glBegin(GL_LINES);
  glVertex3f(-4.0,-0.4,0.0);
  glVertex3f(4.0,-0.4,0.0);
 glEnd();

 // .....
}
```

OpenGL example: Display function

```
static void Display(void) {
 // .....

 // move the ball
 glTranslatef(xp,yp,0.0);
 glutSolidSphere(0.4,25,25);

 glutSwapBuffers();
}
```