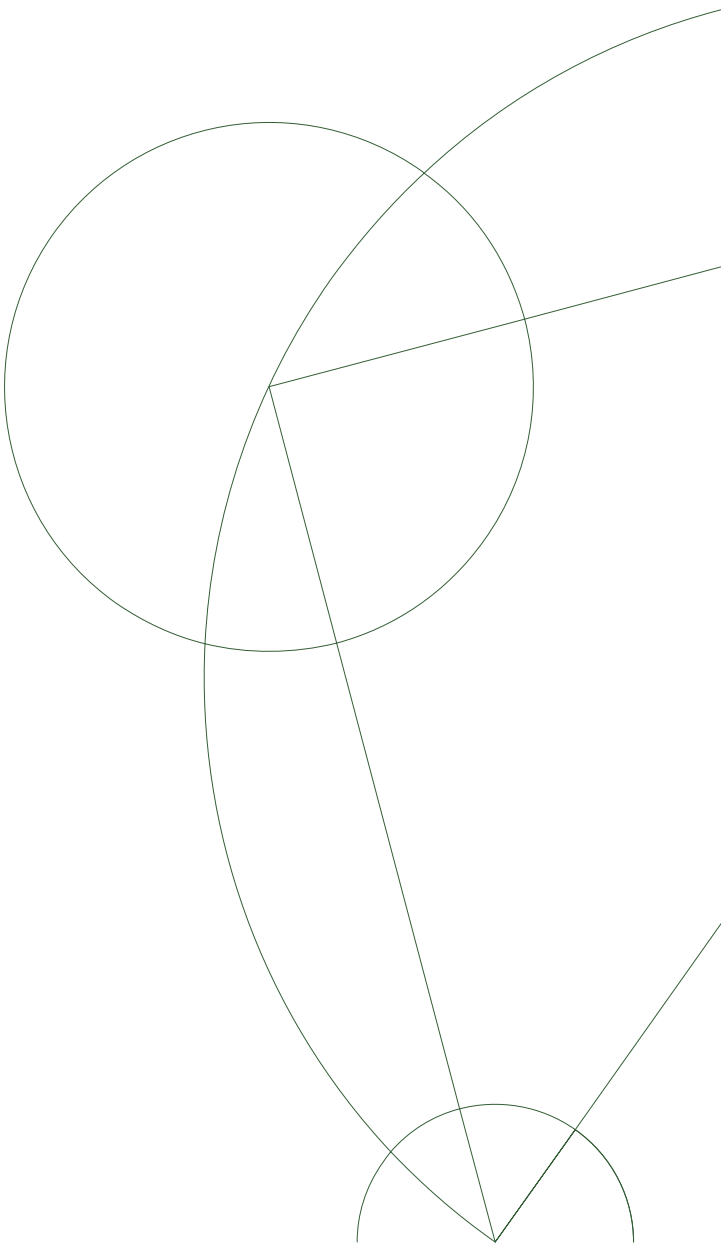
FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Master Thesis

Jian Wu – xcb479@alumni.ku.dk

Deep Contact

Accelerating Rigid Simulation with Convolutional Networks



Supervisor: Kenny Erleben

August 6th 2018

Abstract

This is a master theis from

Contents

List of Figures

List of Tables

Chapter 1

Introdustion

1.1 Motivation

Movies studies have been pushing facial and character animation to a level where machine learning can automize a large part of this work in a production pipeline. Research community is exploring machine learning for gait control too and quite successfully or for upscaling of liquid simulation[1].

However, for rigid body problems it is not quite clear how to approach the technicalities in applying deep learning. Some work have been done in terms of inverse simulations or pilings to control rigid bodies to perform a given artistic 'target'. These thechniques are more in spirit of inverse problems that maps initial conditions to a well defined outcome(number of bounces or which face up on a cude) or level of detail idea replacing interiors of piles with stracks of cylinders pf decreasing radius to make an overall apparent pile have a given angle of of repose.

1.2 Thesis Overview

This thesis explore the application of convolutional neural networks in Computer Simulation.

Chapter 2

Rigid Body Dynamics Simulation

This chapter mainly introduces rigid body simulation to help you understand how computer simulate rigid dynamics based on traditional newton-euler equations. For more details, some contact forces solvers are decribed in this chapter. Afterwards, we will use one of solver to run some simulation and get the image data for the next step, grids-transfer. All the discussion about rigid simulation and contacts solver are based on 2-D view.

2.1 Rigid dynamics Simulation

2.1.1 Simulation Basics

Simulating the motion of a rigid body is almost the same as simulating the motion of a particle, so I will start with partcle simulation. For particle simulation, we let function $\boldsymbol{x}(t)$ describe the particle's location in world space at time t. Then we use $\boldsymbol{v}(t) = \frac{\mathrm{d}}{\mathrm{d}(t)}\boldsymbol{x}(t)$ to denote the velocity of the particle at time t. So, the state of a particle at a time t is the particle's position and velocity. We generalize this concept by defining a state vector $\mathbf{Y}(t)$ for a system: for a single particle,

$$\mathbf{Y}(t) = \begin{pmatrix} \boldsymbol{x}_1(t) \\ \boldsymbol{v}_1(t) \end{pmatrix} \tag{2.1}$$

For a system with n particles, we enlarge $\mathbf{Y}(t)$ to be

$$\mathbf{Y}(t) = \begin{pmatrix} \boldsymbol{x}_1(t) \\ \boldsymbol{v}_1(t) \\ ... \\ \boldsymbol{x}_n(t) \\ \boldsymbol{v}_n(t) \end{pmatrix} \tag{2.2}$$

However, to simulate the motion of particles actually, we need to know one more thing – the forces. $\boldsymbol{F}(t)$ is defined as the force acting on the particle. If the mass of the particle is m, then the changes of $\mathbf{Y}(t)$ will be given by

$$\frac{\mathrm{d}}{\mathrm{d}(t)}\mathbf{Y}(t) = \frac{\mathrm{d}}{\mathrm{d}(t)} \begin{pmatrix} \boldsymbol{x}(t) \\ \boldsymbol{v}(t) \end{pmatrix} = \begin{pmatrix} \boldsymbol{v}(t) \\ \boldsymbol{F}(t)/m \end{pmatrix} \tag{2.3}$$

2.1.2 Rigid Body Concepts

Unlike a particle, a rigid body occupies a volume of space and has a particular shape. Rigid bodies are more complicated, beside translating them, we can rotate them as well. To locate a rigid body, we use $\boldsymbol{x}(t)$ to denote their translation and a rotation matrix $\boldsymbol{R}(t)$ to describe their rotation.

2.1.3 Rigid Body Equations of Motions

Whereas linear momentum $\boldsymbol{P}(t)$ is related to linear velocity with a scalar (the mass), angular momentum is related to angular velocity with a matrix $\boldsymbol{I}$, called the angular inertia matrix. The reason for this is that objects generally have different angular inertias around different axes of rotation. Angular momentum is defined as $\boldsymbol{L}$. The linear momentum is defined as 2.4, and angular momentum is defined as 2.5

$$\boldsymbol{P}(t) = m\boldsymbol{v}(t) \tag{2.4}$$

$$\boldsymbol{L}(t) = \boldsymbol{I}(t)\boldsymbol{\omega}(t) \tag{2.5}$$

The total torque $\boldsymbol{\tau}$ applied to the body is equal to the rate of change of the angular momentum, as defined in 2.6:

$$\boldsymbol{\tau} = \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{L} = \frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{I}\boldsymbol{\omega}) \tag{2.6}$$

Then we can covert all concepts we need to define stare $\mathbf{Y}$ for a rigid body.

$$\mathbf{Y}(t) = \begin{pmatrix} \boldsymbol{x}(t) \\ \boldsymbol{R}(t) \\ \boldsymbol{P}(t) \\ \boldsymbol{L}(t) \end{pmatrix} \tag{2.7}$$

Like what is epressed in $\mathbf{Y}(t)$, the state of a rigid body is mainly consist by its position and orientation (describing spatial information), and its linear and angualr momentum(describe velocity information). Since mass m and bodyspace inertia tensor $\boldsymbol{I}_{body}$ are constants, we can the auxiliary quantities $\boldsymbol{I}(t)$, $\boldsymbol{\omega}(t)$ at any given time.

$$\boldsymbol{v}(t) = \frac{\boldsymbol{P}(t)}{m} \quad \boldsymbol{I}(t) = \boldsymbol{R}(t)\boldsymbol{I}_{body}\boldsymbol{R}(t)^T \quad \boldsymbol{\omega}(t) = \boldsymbol{I}(t)^{-1}\boldsymbol{L}(t) \tag{2.8}$$

The derivative $\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{Y}(t)$ is

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{Y}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} \boldsymbol{x}(t) \\ \boldsymbol{R}(t) \\ m\boldsymbol{v}(t) \\ \boldsymbol{L}(t) \end{pmatrix} = \frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} \boldsymbol{v}(t) \\ \boldsymbol{\omega}(t) \times \boldsymbol{R}(t) \\ \boldsymbol{F}(t) \\ \boldsymbol{\tau}(t) \end{pmatrix} \tag{2.9}$$

Then, we can evaluate Equation 2.10 as follows:

$$\begin{aligned} \boldsymbol{\tau} &= \frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{I}\boldsymbol{\omega}) \\ &= \boldsymbol{I}\dot{\boldsymbol{\omega}} + \dot{\boldsymbol{I}}\boldsymbol{\omega} \\ &= \boldsymbol{I}\dot{\boldsymbol{\omega}} + \frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{R}\boldsymbol{I}_{body}\boldsymbol{R}^T)\boldsymbol{\omega} \\ &= \boldsymbol{I}\dot{\boldsymbol{\omega}} + (\dot{\boldsymbol{R}}\boldsymbol{I}_{body}\boldsymbol{R}^T + \boldsymbol{R}\boldsymbol{I}_{body}\dot{\boldsymbol{R}}^T)\boldsymbol{\omega} \\ &= \boldsymbol{I}\dot{\boldsymbol{\omega}} + ([\boldsymbol{\omega}]\boldsymbol{R}\boldsymbol{I}_{body}\boldsymbol{R}^T + \boldsymbol{R}\boldsymbol{I}_{body}\boldsymbol{R}^T\hat{\omega})\boldsymbol{\omega} \\ &= \boldsymbol{I}\dot{\boldsymbol{\omega}} + [\boldsymbol{\omega}]\boldsymbol{I}\boldsymbol{\omega} - \boldsymbol{I}[\boldsymbol{\omega}]\boldsymbol{\omega} \end{aligned} \tag{2.10}$$

Since $\boldsymbol{\omega} \times \boldsymbol{\omega}$ is zero, the final term can be cancels out. This relationship left is knowned as :

$$\boldsymbol{\tau} = \boldsymbol{I}\dot{\boldsymbol{\omega}} + [\boldsymbol{\omega}]\boldsymbol{I}\boldsymbol{\omega} \tag{2.11}$$

2.1.4 Twist/Wrench

We will now introduce vectors called twists, which describe velocities, and wrenches, which describe forces, and explain how these objects transform from one coordinate frame to another one.

Twist

A twist is a vector that expresses rigid motion or velocity. In Section 2.2, we saw how to parameterize the velocity of a rigid body as a linear velocity vector and an angular velocity vector. The coordinates of a twist are given as a 4-vector in 2-D simulation, which we can check in 2.12

$$\mathbf{v} = \begin{pmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{pmatrix} \tag{2.12}$$

. The defination can be found in 2.12, containing a linear velocity vector $\boldsymbol{v}$ and an angular velocity $\boldsymbol{\omega}$. According to

Wrench

A wrench is a vector that expresses force and torque acting on a body. A wrench can be defined by

$$\mathbf{f} = \begin{pmatrix} \boldsymbol{\tau} \\ \boldsymbol{f} \end{pmatrix} \tag{2.13}$$

A wrench contains an angular component $\boldsymbol{\tau}$ and a linear component f, which are applied at the origin of the coordinate frame they are specified in.

2.1.5 Newton-Euler Equation

The Newton-Euler equations for a rigid body can now be written in terms of the body's acceleration twist $\mathbf{v}$ methioned in 2.12 and the wrench $\mathbf{f}$ metioned in **??** acting on the body. We can simply write the Newton and Eular equations,

$$\begin{pmatrix} \boldsymbol{\tau} - \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega} \\ \boldsymbol{f} \end{pmatrix} = \begin{pmatrix} \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & m\mathbf{1}_{d\times d} \end{pmatrix} \dot{\mathbf{v}} \tag{2.14}$$

d stands for the number pf dimensions, like $d = 2$ in 2-D simulation.

If we define $\mathcal{M}$ and $\mathbf{h}$ as Equation 2.15 and **??**

$$\mathcal{M} = \begin{pmatrix} \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & m\mathbf{1}_{d\times d} \end{pmatrix} \tag{2.15}$$

$$\mathbf{h} = \begin{pmatrix} \boldsymbol{\tau} - \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega} \\ \boldsymbol{f} \end{pmatrix} \tag{2.16}$$

So, we can rewrite Newton-Euler equation as,

$$\mathcal{M}\dot{\mathbf{v}} = \mathbf{h} \tag{2.17}$$

2.2 Constrained Dynamics

most interesting simulations of rigid bodies involve some kind of constraints. Usually we want to model systems of bodies that are interacting in some way. Some bodies may be in contact with each other, or attached together by some types of joint. Since this report mainly research contact forces, we

2.2.1 Expressing Constraints as Equations

We express constraints mathematically as algebraic matrix equations with position, velocity, or acceleration vectors as the unknowns. In general, the configuration space of a set of n rigid bodies has dimension 6n. Adding constraint equations restricts the position (or velocity, or acceleration) to a subspace of smaller dimension.

The constraints discussed in this thesis will all be holonomic constraints, which means the constraint on the velocity can be found by taking the derivative of a position constraint. Constraints that do not fit this description are called nonholonomic. An example of a nonholonomic constraint is a ball rolling on a table without slipping. The position of the ball has five degrees of freedom, so there is only one degree of constraint (only the height is constrained). Taking the derivative of the position constraint would only give a velocity constraint of degree one. Additional constraints on the velocity are needed to prevent the ball from slipping.

We will describe position constraints with a constraint function $g(\boldsymbol{p})$, which is a function from the space of possible positions of the rigid bodies, to $\mathbf{R}^d$, where d is the number of degrees of freedom that constraint removes from the dynamic system. If the constraint

function returns a zero vector, then the position $\boldsymbol{p}$ satifies the constraint.

Position constraint can be divided into *equality constraint*(e.g., joint constraints), where the constraint is $g(\boldsymbol{p}) = 0$, and *inequality constraints*(e.g., contact constraints), where $g(\boldsymbol{p}) \geq 0$.
Constraining the position of an object also constrains its velocity. The velocity constraint function can be found by taking the time derivative of the constraint function. We want the velocity constraint function to be a linear function of the twists representing the velocities of all of the rigid bodies in the system. We let $\mathbf{v}$ to be:

$$\mathbf{v} = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \cdots \\ \mathbf{v}_n \end{pmatrix} \tag{2.18}$$

Velocity constraints are of the form

$$\frac{\mathrm{d}g_i}{\mathrm{d}t} = J_i\mathbf{v} = (\mathbf{J}_{i1}, ..., \mathbf{J}_{in}) \begin{pmatrix} \begin{pmatrix} \boldsymbol{\omega}_1 \\ \boldsymbol{v}_1 \end{pmatrix} \\ \begin{pmatrix} \boldsymbol{\omega}_2 \\ \boldsymbol{v}_2 \end{pmatrix} \\ \cdots \\ \begin{pmatrix} \boldsymbol{\omega}_n \\ \boldsymbol{v}_n \end{pmatrix} \end{pmatrix} \tag{2.19}$$

where i is unique for each constraint. The matrix J is called the constraint's *Jacobian matrix*, which we will refer to simply as the Jacobian.

When there are many constraints, like a system with n bodies, m constraints, the velocity constraint equation looks like

$$\begin{aligned}\frac{\mathrm{d}g}{\mathrm{d}t} &= \begin{pmatrix} g_1 \\ g_2 \\ ... \\ g_m \end{pmatrix} = \mathcal{J}\mathbf{v} = \begin{pmatrix} J_1 \\ ... \\ J_n \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ ... \\ \mathbf{v}_n \end{pmatrix} \\ &= \begin{pmatrix} \boldsymbol{J}_{11} & \boldsymbol{J}_{12} & ... & \boldsymbol{J}_{1n} \\ \boldsymbol{J}_{21} & \boldsymbol{J}_{22} & ... & \boldsymbol{J}_{2n} \\ ... & ... & ... & ... \\ \boldsymbol{J}_{m1} & \boldsymbol{J}_{m2} & ... & \boldsymbol{J}_{nn} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} \boldsymbol{\omega}_1 \\ \boldsymbol{v}_1 \end{pmatrix} \\ \begin{pmatrix} \boldsymbol{\omega}_2 \\ \boldsymbol{v}_2 \end{pmatrix} \\ ... \\ \begin{pmatrix} \boldsymbol{\omega}_n \\ \boldsymbol{v}_n \end{pmatrix} \end{pmatrix}\end{aligned} \tag{2.20}$$

Notes:

- $\boldsymbol{J}$ is used for Jacobians that relate the velocity of a single body to a single constraint.
- J is used for Jacobians that relate the velocity of mant bodies with many bodies to a single constraint.
- $\mathcal{J}$ is used for Jacobians that relate velocity of many bodies to many constraints.

2.2.2 Solving Constrained Dynamics Equations using Largange Multipliers

A constraint Jacobian matrix $\mathcal{J}$ has a dual use. In addition to relating the vlocities to the rate of change of the constraint function $\boldsymbol{g}$, the rows of $\mathcal{J}$ act as basis vectors for constraint forces. Thus, actually we do just need to solve for the coefficient vector $\boldsymbol{\lambda}$, which is called the *Lagrange multipliers* that contains the magnitudes of the forces that correspond to each of these basis vectors.

Solving Dynamics at the Force-Acceleration Level

Combining the Newton-Euler equations,

$$\mathcal{M}\dot{\mathbf{v}} = \mathcal{J}^T\boldsymbol{\lambda} + \mathbf{h}_{ext} \tag{2.21}$$

where $\mathcal{M}$ was defined in Equation 2.15, $\mathbf{h}_{ext}$ holds external and gyroscopic force terms.

2.2.3 Contact Constraints

From other papers [2], we can conclude some import features of contact constraint:

- Contact forces can push bodies apart, but cannot pull bodies towards each other. This leads to inequalities in the constraint equations.
- Contact are transient - they come and go. Contacts can appear suddenly when bodies are moving towards each other, resulting in an impact.

Impacts between rigid bodies in reality result in large forces applied in over a very short time period, leading to a sudden changes in velocity. There are serveral methods for handling contact in rigid body simulations. Mirtich[3] give a summary of serval different methods, and explains the advantages and disadvantages of each. We mainly introduced one method called .... from Kenny[4]

- A contract consist of a pair of contact points, one point attached, one point attached to one rigid body, and the other point attached to another rigid body. The contact points are sufficiently close together for our collision detection algorithm to report a collision. Normally, we caculate the distance between two rigid bodies(mainly on circle shape).

$$g_i = \|\boldsymbol{p}_{i1} - \boldsymbol{p}_{i2}\| - |r_{i1} + r_{i2}| \tag{2.22}$$

- A contact normal is a unit vector that is normal to one or both of the surfaces at the contact points.
- The contact constraint Jacobian for constraint i, denoted by $\boldsymbol{J}_{c_i}$, is a $1 \times 6n$ matrix, where n is the number of bodies (the subscript 'c' here stands for 'contact').
- A contact force $\boldsymbol{J}_{c_i}^T \lambda_{c_i}$ is a force acting in the direction of the contact normal which prevents the two rigid bodies from interpenetrating.
- The separation distance of a contact is the normal component of the distance between the two contact points. It is negative when the bodies are interpenetrating at the contact. The constraint function $g_i(\boldsymbol{p})$ for a contact constraint returns the separation distance. In other words, $g_i \geq 0$.

- The relative normal acceleration a_i of a contact is the second derivative of the separation distance with respect to time ($a_i = \ddot{g}_i$). The acceleration constrait is satisfied when $a_i = \boldsymbol{J}_{c_i}\dot{\mathbf{v}}_i + k_i \geq 0$

When g or $\dot{g}$ are positive, it indicates that the bodies are not touching, or moving apart at that contact, respectively, in other words, no acceleration is needed. The assumption that at each resting contact $g = 0$ and $\dot{g} = 0$ are critical to the constraint that we enumerate below.

Let $\boldsymbol{a}$ be a vector containing the relative normal accelerations $\{a_1, ..., a_n\}$ for all contacts and $\boldsymbol{\lambda}_c$ be a vector of contact force multipliers $\{\lambda_{c_1}, ..., \lambda_{c_n}\}$. The vectors $\boldsymbol{a}$ and $\boldsymbol{\lambda}_c$ are linearly related at a given $\mathbf{P}$. Additionally, we have three constraints:

1. The relative normal accelerations must be nongative:

$$\boldsymbol{a} = \mathcal{J}_c\dot{\mathbf{v}} + \boldsymbol{k} \geq 0 \tag{2.23}$$

   Since g and $\dot{g}$ are zero, a negative acceleration would cause interpenetration.

2. The contact force magnitudes must be nonnegative (so as to push the bodies apart):

$$\boldsymbol{\lambda}_c \geq \mathbf{0} \tag{2.24}$$

3. For each contact i, at least one of a_i, λ_{c_i} must be zero, we can write that

$$a_i^T \lambda_{c_i} = 0 \tag{2.25}$$

   Since there can only be a contact force if the bodies are actually touching($g_i\lambda_{c_i} = 0$). Differentiating this twice using chain rule, we get $\ddot{g}_i\lambda_{c_i} + 2\dot{g}\dot{\lambda}_{c_i} + g_i\lambda_i = 0$. Since we assume $g_i = 0$ and $\dot{g}_i = 0$, the second and third term cancel out leaving simply:

$$\ddot{g}_i\lambda_{c_i} = 0 \tag{2.26}$$

   In other words:

$$\boldsymbol{a}^T\boldsymbol{\lambda}_c = \mathbf{0} \tag{2.27}$$

Then we can write the whole system equation as Equation **??**

Chapter 3

Partcle-grid-particle

The basic method for generating training data which is more accessible to learning is that we will map a discrete element method(DEM) into a continuum setting use techniques from smooth particle hydrodynamics. Given a set of bodies δ and a set of contacts between these bodies C.

3.1 Grid-Based method

Traditional rigid motion simulation mainly use particle-based method. However, if we want to replace traditional contact solver with deep learning model, it is hard for cnn model to recognize the original image and do learning. Grid-based methos is a good to transfer original image to a grid-cells and then use

3.2 Smoothed Particle Hydrodynamics

Smoothed particle hydrodynamics (SPH) was invented to simulate nonaxisymmetric phenoma in astrophysis initially. The principal

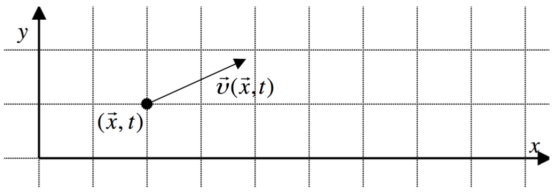


Figure 3.1: Grid description, $retrievedfromMIT$(2011)

idea of SPH is to treat hydrodynamics in a completely mesh-free fashion, in terms of a set of sampling particles. It turns out that the particle presentation of SPH has excellent conservation properties. Energy, linear momentum, angular momentum, mass and velocity.

3.2.1 Fundamentals

At the heart of SPH is a kernel interpolation method which allows any function to be expressed in terms of its values at a set of disordered points - the particles[5]. For ant field $A(\mathbf{r})$, a smoothed interpolated version $A_I(\mathbf{r})$ can be defined by a kernel $W(\mathbf{r}, h)$,

$$A_I(\mathbf{r}) = \int A(\mathbf{r}')W(\|\mathbf{r} - \mathbf{r}'\|, h)\,\mathrm{d}\mathbf{r}' \tag{3.1}$$

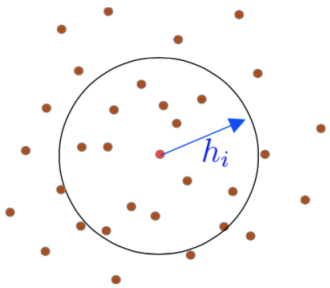


Figure 3.2: Visilaztion of SPH

where the integration is over the entire space, and W is an interpolating kernel with

$$\int W(\|\mathbf{r} - \mathbf{r}'\|, h)\,\mathrm{d}\mathbf{r}' = 1 \tag{3.2}$$

and

$$\lim_{h\to 0} W(\|\mathbf{r} - \mathbf{r}'\|, h)\,\mathrm{d}\mathbf{r}' = \delta(\|\mathbf{r} - \mathbf{r}'\|) \tag{3.3}$$

Normally, we want the kenel to be Non-negative and rotational invariant.

$$W(\|\mathbf{x}_i - \mathrm{x}_j\|, h) = W(\|\mathrm{x}_j - \mathrm{x}_i\|, h) \tag{3.4}$$

$$W(\|\mathbf{r} - \mathbf{r}'\|, h) \geq 0 \tag{3.5}$$

For numerical work, we can use midpoint rule,

$$A_I(\mathbf{x}) \approx A_S(\mathbf{x}) = \sum_i A(\mathbf{x}_i) W(\|\mathbf{x}_i - \mathbf{x}\|, h) \Delta V_i \tag{3.6}$$

Since $V_i = m_i/\rho_i$

$$A_S(\mathbf{x}) = \sum_i \frac{m_i}{\rho_i} A(\mathbf{x}_i) W(\|\mathbf{x}_i - \mathbf{x}\|, h) \tag{3.7}$$

The default, gradient and Laplacian of A are:

$$\begin{aligned} \nabla A_S(\mathbf{x}) &= \sum_i \frac{m_i}{\rho_i} A(\mathbf{x}_i) \nabla W(\|\mathbf{x}_i - \mathbf{x}\|, h) \\ \nabla^2 A_S(\mathbf{x}) &= \sum_i \frac{m_i}{\rho_i} A(\mathbf{x}_i) \nabla^2 W(\|\mathbf{x}_i - \mathbf{x}\|, h) \end{aligned} \tag{3.8}$$

3.2.2 Kernels

Smoothing kernels functions are one of the most important points in SPH. Stability, accurancy and speed of the whole method depends on these fuctions. Different kernels are being used for different purposes. One possibilyty for W is a Gaussian. However, most current SPH implementations are based on kernels with finite support. We mainly introduce gaussian, poly6 and spicky kernel here. And compare the different kernels and their property.

Poly6

The kernel is also known as the 6th degree polynomial kernel.

$$W_{poly6}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - \|\mathbf{r}\|^2)^3 & 0 \le \|\mathbf{r}\| \le h \\ 0 & \text{Otherwise} \end{cases} \tag{3.9}$$

Then, the gradient of this kernel function can be

$$\nabla W_{poly6}(\mathbf{r}, h) = -\frac{945}{32\pi h^9} \begin{cases} \mathbf{r}(h^2 - \|\mathbf{r}\|^2)^2 & 0 \le \|\mathbf{r}\| \le h \\ 0 & \text{Otherwise} \end{cases} \tag{3.10}$$

The laplacian of this kenel can be expressed by,

$$\nabla^2 W_{poly6}(\mathbf{r}, h) = -\frac{945}{16\pi h^9} \begin{cases} (h^2 - \|\mathbf{r}\|^2)(3h^2 - 7\|\mathbf{r}\|^2) & 0 \le \|\mathbf{r}\| \le h \\ 0 & \text{Otherwise} \end{cases} \tag{3.11}$$

As Müller stated[6], if the kernel is used for the computation of pressure forces, particles tend to build cluster under high pressure because 'as particles get very close to each other, the repulsive force vanishes because the gradient of the kerbek approaches zero to the center.', which we can see in Figure 3.4. Another kernel, spiky kernel, is proposed by Desbrum and Gascuel[7] to solve this problem.

Spicky

The kernel proposed by Desbrum and Gascuel[7]

$$W_{spiky}(\mathbf{r}, h) = \frac{15}{\pi h^6} \begin{cases} (h - \|\mathbf{r}\|)^3 & 0 \leq \|\mathbf{r}\| \leq h \\ 0 & \text{Otherwise} \end{cases} \tag{3.12}$$

Then, the gradient of spiky kernel can be described by,

$$\nabla W_{spiky}(\mathbf{r}, h) = -\frac{45\mathbf{r}}{\pi h^6 \|\mathbf{r}\|} \begin{cases} (h - \|\mathbf{r}\|)^2 & 0 \leq \|\mathbf{r}\| \leq h \\ 0 & \text{Otherwise} \end{cases} \tag{3.13}$$

The laplacian of spiky can be expressed by,

$$\nabla^2 W_{spiky}(\mathbf{r}, h) = \frac{90}{\pi h^6} \begin{cases} h - \|\mathbf{r}\| & 0 \leq \|\mathbf{r}\| \leq h \\ 0 & \text{Otherwise} \end{cases} \tag{3.14}$$

3.2.3 Grid size and smoothing length

The grid should be also fine enough to capture the variation in our simulation. In our case, it is reasonable to have a grid fine enough such that no two contact points are mapped into the same cell.

Smoothing length, h, is one of the most important parameters that affects the whole SPH method by changing the kernel value results abd neighbor searching results. Too small or too big values might cause lose essencail information in the simulation.

3.2.4 Neignbor Search

Neighbor search is one of the most crucial procedures in SPH method considersing all interpolation equations, $A(\mathbf{r})$, needs the neighbor list for every particle (refer to equation 3.8). A naive neighbor searching approach would end up with a complexity of $o(n^2)$. The complexity

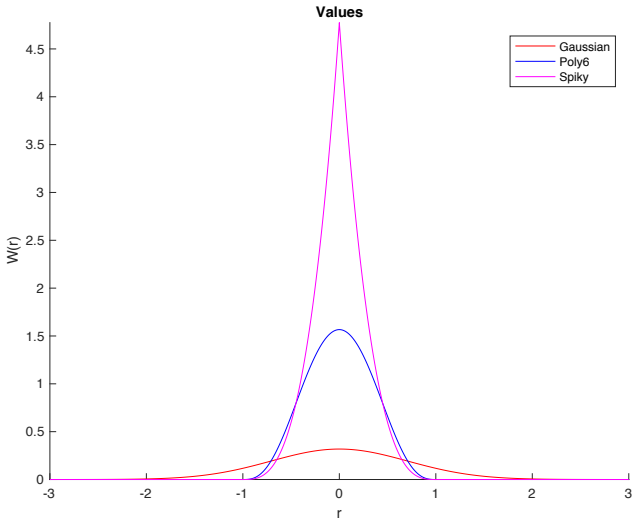


Figure 3.3: Comparation of different kernels, we set smoothing length $h = 1$ here.

is not good enough since it is impossoble to reach any interactive speed when the particle count increses. With an efficient nearest neighbor searching(NNS) algorithm, it is possible to have a significant performance increase since it is the most time consuming procedure in SPH computation. In order to decrease the complexity, we choose to use $k - d$ tree data structure to store the particla spatial information and then do the nearest neighbor searching.

Hierarchical Tree

Using an adaptive hierarchy tree search is proposed by Paiva[8] to find particle neighbors. Since the simulation takes place in two dimensions, $k - d$ tree data structure was used in this approach.

An octree structure has been adapted from Macey (b) Octree Demo. The hierarchy tree is formed with a pre-defined height. The simulation box is divided recursively into eight pieces, nodes. The nodes at height $= 1$ are called leaves. Each node has a surrounding box for particle query which is used to check if the particle is inside the node. Each parent node, contains the elements that are divided

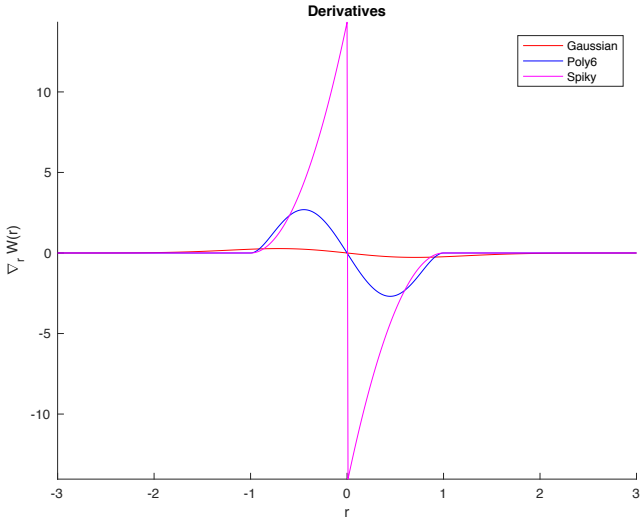


Figure 3.4: Comparation of gradient of different kernels, we set $h = 1$ here.

through its children. The particle is being checked at each level of the tree if it's intersecting the node. If it does, descend one level down in that node. After reaching to leaves, bottom level, particle has been checked if its within the search distance, h. If the particle is in the range, add it to the neighbor list. Neighbor searching is done when the whole tree has been traversed. The search distance set to be smoothing length in this implementation.

The complexity of this tree search method is $O(nlog(n))$, n being the number of particles. The performance of this algorithm is worse than Spatial Hashing method. In addition, the results obtained using this NNS algorithm wasn't accurate and stable for this implementation. Therefore as mentioned above, Spatial Hashing method was preferred.

3.3 Grid to particle

SPH will be used for us to transform current state of dynamic system to grid images. After getting the grid image for simulation state in

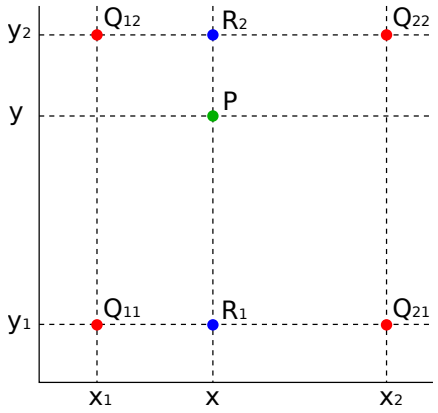


Figure 3.5: The figure shows visiualization of bilinear interpolation. The four red dots show the data points and the green dot is the point at which we want to interpolate.

time t, we will use the grid cells as input and renew the contact grid image based on trained model. Once the contact force image is obtained, we will use contact position to interpolate image values. The interpolated values will be stored in the contact points and used as starting iterates for contact force solver. Then we can update states of all rigid bodies in time $t + \Delta t$.

3.3.1 Bilinear interpolation

We applied bilinear interpolation in our case, since we did mainly research on a rectilinear $2 - D$ grid. The key idea is to perform linear interpolation first in one direction, and then again in the other direction. Although each step is linear in the sampled values and in the position, the interpolation as a whole is not linear but rather quadratic in the sample location.

As shown in Figure 3.5, We have known $Q_{a,b} = (x_a, y_b)$ and $a \in \{1, 2\} \quad b \in \{1, 2\}$. Then, we can firstly do linear interpolation in the x-direction. This yields

$$
\begin{aligned}
f(x, y_1) &\approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}), \\
f(x, y_2) &\approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}).
\end{aligned} \tag{3.15}
$$

After getting the two values in $x-$direction $f(x, y_1)$ and $f(x, y_2)$, we can use these values to do interpolation in $y-$ direction.

$$
f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2) \tag{3.16}
$$

Combine $f(x, y_1)$ and $f(x, y_2)$ defined in equation 3.15, we can get,

$$
\begin{aligned}
f(x, y) \approx & \frac{y_2 - y}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \right) \\
& + \frac{y - y_1}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \right) \\
= & \frac{1}{(x_2 - x_1)(y_2 - y_1)} (\\
& f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{21})(x - x_1)(y_2 - y) \\
& + f(Q_{12})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1)) \\
= & \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \\
& \cdot \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}
\end{aligned} \tag{3.17}
$$

3.4 Experiment and Conclution

Our hope is that strating iterates will be close to "solution" of the contact problem, which can indicate the contact force solvers will coverage very rapidly or maybe not even need to iterate. In order to test whether the sph method can be applied in our case, the contact force solution is mapped to image and interpolated values are generated and used to re-start the contact force solver. Our hyposis is that iterative solver quickly recovers an iteration close to the original solution before mapping to force grid image.

Chapter 4

Deep Learning For Simulation

Deep Learning, as a branch of Machine Learning, employs algorithms to process data and imitate the thinking process[9], or to develop abstractions. Deep Learning (DL) uses layers of algorithms to process data, understand human speech, and visually recognize objects. Information is passed through each layer, with the output of the previous layer providing input for the next layer. The first layer in a network is called the input layer, while the last is called an output layer. All the layers between the two are referred to as hidden layers. Each layer is typically a simple, uniform algorithm containing one kind of activation function.

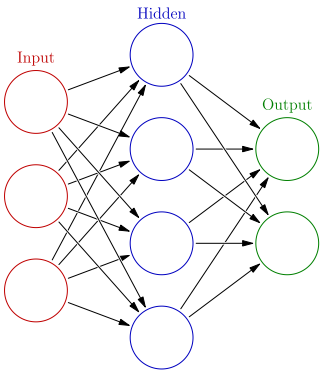


Figure 4.1: visulization of one simple 3-layers neural networks, including input layer, hidden layer and output layer, *retrieved from Wikipedia*

With deep learning becoming more and more popular in many fields of researching, some classical methods can be replaced by deep learning. Many successful application of deep learning can be found in computer vision part, like image segmentation[10],objection recognition[11].

4.1 Convolutional Neural Networks

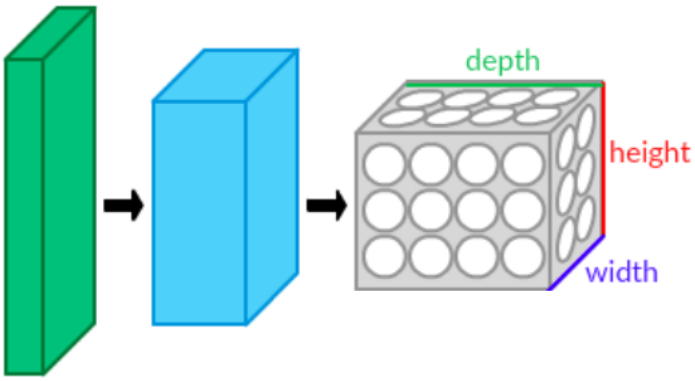


Figure 4.2: Visualization of convolutions network, *retrieved from Wikipedia*

4.2 CNN Constructure

4.2.1 Convolutional layers

4.2.2 Full-connected layers

4.3 Traing configuration

4.3.1 Loss Function

4.3.2 Optimizer(SGD)

4.4 Training Results

4.5 Simulation based on Trained model

Chapter 5

Implementation

Chapter 6

Conclusion

6.1 Future work

References

[1] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, "Accelerating Eulerian Fluid Simulation With Convolutional Networks", *ArXiv e-prints*, Jul. 2016. arXiv: `1607.03597 [cs.CV]`.

[2] J. Bender, K. Erleben, and J. Trinkle, "Interactive simulation of rigid body dynamics in computer graphics", in *Computer Graphics Forum*, Wiley Online Library, vol. 33, 2014, pp. 246–270.

[3] B. Mirtich, "Rigid body contact: Collision detection to force computation", in *Workshop on Contact Analysis and Simulation, IEEE Intl. Conference on Robotics and Automation*, 1998.

[4] K. Erleben, "Rigid body contact problems using proximal operators", in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, 2017, p. 13.

[5] J. J. Monaghan, "Smoothed particle hydrodynamics", *Annual review of astronomy and astrophysics*, vol. 30, no. 1, pp. 543–574, 1992.

[6] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications", in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, 2003, pp. 154–159.

[7] M. Desbrun and M.-P. Gascuel, "Smoothed particles: A new paradigm for animating highly deformable bodies", in *Computer Animation and Simulation'96*, Springer, 1996, pp. 61–76.

[8] A. Paiva, F. Petronetto, T. Lewiner, and G. Tavares, "Particle-based non-newtonian fluid animation for melting objects", in *Computer Graphics and Image Processing, 2006. SIBGRAPI'06. 19th Brazilian Symposium on*, IEEE, 2006, pp. 78–85.

[9] J. Schmidhuber, "Deep learning in neural networks: An overview", *Neural networks*, vol. 61, pp. 85–117, 2015.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *nature*, vol. 521, no. 7553, p. 436, 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.