

อัลกอริทึมตรวจสอบการชนของวัตถุที่มีการเปลี่ยนแปลงรูปร่างโดยใช้วิธีทางอนุภาค



นางสาว นิดา แสงแห่งธรรม

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์


คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2548

ISBN 974-53-2966-5

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

COLLISION DETECTION ALGORITHM FOR DEFORMABLE OBJECTS  
USING PARTICLE-BASED METHOD



Miss Nida Saenghaengtham

สถาบันวิทยบริการ

จุฬาลงกรณ์มหาวิทยาลัย

A Thesis Submitted in Partial Fulfillment of the Requirements  
for the Degree of Master of Engineering Program in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2005

ISBN 974-53-2966-5

Thesis Title                      Collision Detection Algorithm for Deformable Objects using  
Particle-based Method

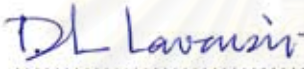
By                                      Miss Nida Saenghaengtham

Field of Study                      Computer Engineering

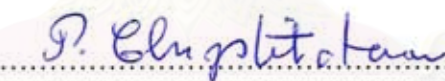
Thesis Advisor                      Pizzanu Kanongchaiyos, Ph.D.


---


Accepted by the Faculty of Engineering, Chulalongkorn University in Partial  
Fulfillment of the Requirements for the Master's Degree


  
..... Dean of the Faculty of Engineering  
(Professor Direk Lavansiri, Ph.D.)

THESIS COMMITTEE

  
..... Chairman  
(Associate Professor Prabhas Chongstitvatana, Ph.D.)

  
..... Thesis Advisor  
(Pizzanu Kanongchaiyos, Ph.D.)

  
..... Member  
(Athasit Surarerks, Ph.D.)

  
..... Member  
(Supatana Auethavekiat, Ph.D.)

นิตา แสงแห่งธรรม : อัลกอริทึมตรวจสอบการชนของวัตถุที่มีการเปลี่ยนแปลงรูปร่างโดยใช้วิธีทางอนุภาค (COLLISION DETECTION ALGORITHM FOR DEFORMABLE OBJECTS USING PARTICLE-BASED METHOD)

อาจารย์ที่ปรึกษา: ดร.พิชณู คนองชัยยศ, 69 หน้า. ISBN 974-53-2966-5

อัลกอริทึมตรวจสอบการชนส่วนใหญ่มักมีพื้นฐานอยู่บนการประมาณวัตถุด้วยรูปทรงลำดับชั้น ซึ่งเทคนิคเหล่านี้ไม่เหมาะสมที่จะนำมาใช้กับวัตถุที่เปลี่ยนแปลงรูปร่างได้ เนื่องจากต้องทำการคำนวณรูปทรงลำดับชั้นที่ยุ่งยากใหม่ทุกครั้งที่วัตถุเปลี่ยนแปลงรูปร่างไป ดังนั้นงานวิจัยนี้จึงได้นำเสนออัลกอริทึมใหม่เพื่อใช้ในการตรวจสอบการชนของวัตถุที่เปลี่ยนแปลงรูปร่างได้โดยใช้วิธีทางอนุภาค หลักการของอัลกอริทึมนี้ก็คือ การกำหนดให้แต่ละอนุภาคเป็นเหมือนตัวตรวจรู้ในการตรวจสอบแต่ละพื้นที่ของตนว่ามีโอกาสสูงที่จะเกิดการชนขึ้นหรือไม่ ซึ่งการแบ่งพื้นที่นั้นทำได้โดยการจัดจุดของวัตถุเป็นกลุ่มๆ โดยจำนวนกลุ่มนั้นประมาณได้จากการวิเคราะห์หาจำนวนวัตถุทั้งหมดที่สามารถชนกับพื้นผิวที่พิจารณาอยู่ได้ จากนั้นจะกำหนดให้อนุภาคแต่ละตัวมีหน้าที่ควบคุมในแต่ละพื้นที่ดังกล่าว โดยสามารถเคลื่อนที่ไปมาภายในพื้นที่ตามแรงดูดที่เกิดจากอนุภาคบนวัตถุอื่นที่อยู่ข้างเคียง ดังนั้นเมื่อวัตถุเคลื่อนเข้าใกล้กันในระยะที่ก่อให้เกิดการชนอนุภาคที่อยู่บนวัตถุนั้นๆ ก็จะถูกดึงดูดเข้าหากันจนถึงระยะที่สามารถสรุปว่าเกิดการชนกันขึ้นได้ ด้วย ทั้งนี้จะทำการกำหนดค่าที่ยอมรับได้ค่าหนึ่งมาเปรียบเทียบกับระยะระหว่างอนุภาค เพื่อพิจารณาแนวโน้มที่วัตถุจะเกิดการชน

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

ภาควิชา.... วิศวกรรมคอมพิวเตอร์.....ลายมือชื่อนิสิต..... *N. Somya*.....  
สาขาวิชา....วิศวกรรมคอมพิวเตอร์.....ลายมือชื่ออาจารย์ที่ปรึกษา..... *[Signature]*.....  
ปีการศึกษา .....2548.....ลายมือชื่ออาจารย์ที่ปรึกษาร่วม.....



## 4770582121 : MAJOR COMPUTER ENGINEERING

KEY WORD: COLLISION DETECTION / PARTICLE-BASED / SURFACE PARTITIONING

NIDA SAENGAENGTHAM: COLLISION DETECTION ALGORITHM FOR DEFORMABLE OBJECTS USING PARTICLE-BASED METHOD THESIS

ADVISOR: PIZZANU KANONGCHAIYOS, Ph.D., 69 pp. ISBN 974-53-2966-5

Most collision detection algorithms have been proposed based on hierarchical bounding representation. These techniques are notable to be used with deformable surfaces because their bounding representations have to be updated when surface deformation occurs which costs quite expensive. Therefore, this research proposed an alternative algorithm for collision detection among non-rigid deformable polygonal models using particle-based method. The basis of this algorithm is to set each particle as a sensor in a separated area to determine whether there exists a high possibility for collision. Surface partitioning is firstly applied by equally dividing vertices into several groups which can be approximated as the number of objects that can touch the surface. Each area is then assigned with a particle moving inside by the attractive forces from other particles on neighboring objects. If the collision occurs, their corresponding particles will also be collided. A tolerable parameter is properly set in order to determine the distance between two particles on the verge of collision.

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

Department.... Computer Engineering.... Student's signature.....*N. Saengtham*.....  
Field of study.... Computer Engineering...Advisor's signature.....*P. Kanongchaiyos*.....  
Academic year ...2005.....Co-advisor's signature.....

## ACKNOWLEDGMENTS

It is a great pleasure to acknowledge my thesis advisor, Pizzanu Kanongchaiyos, Ph.D., for his intellectual advices and invariable assistances throughout this research. I would also like to express my grateful thanks to my thesis committee, Associate Professor Prabhas Chongstitwatana, Ph.D, Athasit Surarerks, Ph.D., Supatana Auethavekiat, Ph.D. for their beneficial guidance and suggestions.

I also want to extend my thanks to all 20<sup>th</sup> floor members especially my associates in computer graphic lab (CG Lab) for their generous helps, encouragements and truly relationships which make my life through the course filled with amusements and happiness.

Finally, I deeply wish to thank my parents for their love, understanding and invaluable supports throughout my graduate study.



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## TABLE OF CONTENTS

	Page
ABSTRACT (THAI) .....	iv
ABSTRACT (ENGLISH) .....	v
ACKNOWLEDGMENTS .....	vi
LIST OF FIGURES .....	ix
LIST OF TABLES .....	xii
CHAPTER 1 INTRODUCTION .....	1
1.1 Background and Problem Statement .....	1
1.2 Objectives .....	3
1.3 Scopes of Study .....	3
1.4 Research Procedure .....	3
1.5 Expected Benefits .....	3
1.6 Thesis Structure .....	4
1.7 Publications .....	4
CHAPTER 2 THEORETICAL BACKGROUND AND RELATED WORKS .....	5
2.1 Theory .....	5
2.1.1 Collision Detection .....	5
2.1.2 Particle-based Method .....	7
2.1.3 Vector Quantization .....	10
2.2 Related works .....	14
2.2.1 Sphere .....	14
2.2.2 Axis-Aligned Bounding Boxes .....	15
2.2.3 Oriented Bounding Boxes .....	17
2.2.4 Particle-based Collision Detection .....	19
CHAPTER 3 PROPOSED ALGORITHM .....	22
3.1 The Number of Particles .....	23
3.2 Surface Partitioning .....	25
3.3 Interaction Forces .....	31

3.4 Movement of Particles.....	33
3.5 Collision Distance.....	36
3.6 Repartition Checking.....	36
CHAPTER 4 ALGORITHM ANALYSIS.....	42
4.1 Correctness .....	42
4.1.1 Case 1: two convex objects which have the same size .....	43
4.1.2 Case 2: several convex objects which have the same size.....	44
4.1.3 Case 3: several convex objects which have non-equivalent size.....	48
4.1.4 Case 4: two concave objects .....	52
4.1.5 Case 5: several concave objects.....	53
4.1.6 Case 6: both convex and concave objects.....	53
4.2 Complexity analysis .....	54
4.2.1 Surface partitioning.....	54
4.2.2 Collision detection .....	54
4.2.3 Repartition checking .....	55
CHAPTER 5 DISCUSSION CONCLUSION AND FUTURE WORK.....	57
5.1 Discussion.....	57
5.1.1 Problem of the complex topology.....	57
5.1.2 Problem of partitioning symmetry object centered at the origin .....	58
5.1.3 Problem of the improper effective radius of attraction ( $R_{eff}$ ).....	59
5.1.4 Problem of the improper collision distance value ( $\mu$ ).....	61
5.1.5 Problem of the improper acceptable area ( $A^*$ ) .....	63
5.1.6 Problem of object deformation .....	63
5.1.7 Complexity discussion.....	65
5.2 Conclusion .....	66
5.3 Future Work.....	66
REFERENCES .....	67
BIOGRAPHY .....	69



## LIST OF FIGURES

Figure		Page
1-1	A car crashes a wall with and without collision detection.....	1
1-2	Cloth simulation.....	2
2-1	Intersection of triangles.....	5
2-2	Intersection of planes.....	6
2-3	Sphere - plane collision.....	6
2-4	Smoke simulation using particle-based method.....	8
2-5	Acceleration and velocity of particles on 2-dimentional plain.....	9
2-6	1-Dimension Vector quantization .....	10
2-7	2-Dimension Vector Quantization .....	10
2-8	The LBG procedure .....	13
2-9	Error from using sphere collision detection.....	14
2-10	Sphere dividing method for determination of the objects.....	15
2-11	The AABB box arrangement.....	15
2-12	New box creations when the object rotate.....	16
2-13	A linear stick and its AABB box in various directions.....	16
2-14	The OBB box arrangement.....	17
2-15	Hierarchical methods for OBBs.....	18
2-16	Collision detection using Particle-based method.....	19
2-17	Particle distribution.....	20
2-18	Collision detection between two models using particle-based method.....	20
2-19	Time to perform a collision query.....	20
2-20	Improper particle dispersion at the initialization.....	21
3-1	A new algorithm developed for the collision detection.....	22

Figure	Page
3-2 Kissing number.....	23
3-3 Circular shadow on the imaginary sphere.....	24
3-4 Approximated spheres for objects.....	24
3-5 Vertex classification.....	25
3-6 Vector quantization.....	26
3-7 Improper partitioning of LBG with a concave object.....	27
3-8 Two close adjacent vertices having different normal vectors.....	27
3-9 Flow chart shows the partitioning.....	31
3-10 Attractive forces acts on particle.....	32
3-11 Particle position and its neighboring coordinate.....	34
3-12 Undetected collision when particles can move only one vertex each time.....	34
3-13 Flow chart shows the movement of particles.....	35
3-14 Four directions used to find size of the area.....	37
3-15 Tracking along +X axis.....	38
3-16 The distance estimation along +X axis.....	39
3-17 Flow chart shows the repartition checking procedures.....	40
4-1 Two equivalent spheres touching at point C.....	43
4-2 Three touching spheres.....	44
4-3 Six sets of particles.....	45
4-4 Three touching spheres after all movement of particles.....	47
4-5 Three touching spheres.....	48
4-6 Six sets of particles.....	48
4-7 Updated figure of three touching non-equivalent spheres.....	49
4-8 Three touching spheres after all movement of particles.....	51
4-9 Two touching concave objects.....	52

Figure	Page
4-10	Each concave is considered as a combination of three convex parts.....52
5-1	Particles are unnecessarily generated due to a special case of topology.....57
5-2	An object which has complex topology.....58
5-3	(a) a cube centered at the origin, (b) a cube centered away from the origin...59
5-4	Undetected collision due to a very small value of $R_{eff}$ .....59
5-5	Two touching sphere objects.....60
5-6	A collision on object edge.....62
5-7	An incorrectly report of collision due to a high value of collision distance...63
5-8	Deformation case that cause the particle deficiency to detect the collision...64
5-9	Undetected collision due to a very long edge.....64

**LIST OF TABLES**

Table		Page
2-1	Advantages and disadvantages of each collision detection algorithm.....	18
3-1	Comparison between original particle-based and the proposed algorithm...41	
4-1	Six possible cases.....	43



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

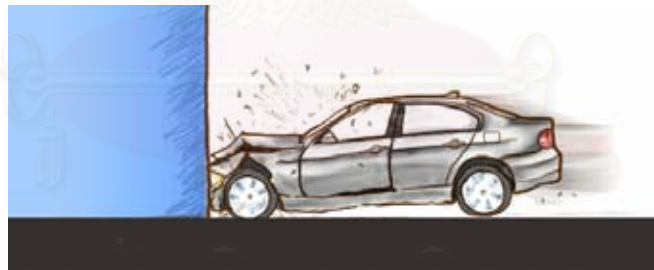
# CHAPTER 1

## INTRODUCTION

### 1.1 Background and Problem Statement

The collision detection is one of the most important tasks in the animation systems which have more than one moving object. Also, it is one of the most interesting problems in computer graphic field [1] since it is applicable to several areas, for example, the robotics, computer games, computational biology, computer simulation, and virtual reality, etc.

The main purpose of the collision detection is to report whether there exist a high possibility for collision and then shows the collision response which is very crucial for preparation procedure in the simulation. The importance of the collision detection is shown in Figure 1-1. It is obvious that, without collision detection process, the system will never know that there is a collision occurs and let the car pass through the wall. Therefore, the collision detection is required in order to check whenever the car hits the wall, and not allows it to pass through the wall.



(a)



(b)

Figure 1-1. A car crashes a wall with and without collision detection.



There are several kinds of models in computer graphic such as Constructive Solid Geometry (CSG), Implicit or Parametric surface, etc. Objects can be either rigid or deformable object. The object represented by polygonal model usually deform by updating the positions of the polygon vertices.

Most collision detection algorithms are used with solid and rigid objects, while collision detection for deformable objects is also important. Therefore, a number of researches are increase significantly since it can be applied to several fields such as computer animation, cloth simulation, and virtual reality, etc. For instance, in order to generate cloth simulation [2] looks like real cloth, the collision detection has to be employed between the cloth and neighboring objects, and even between the cloths. As the cloth shape is not constant, general techniques which based on hierarchical bounding representation cannot detect the collision efficiently. Their bounding representations have to be updated every time surface deformation occurs which can also be expensive in the sense of required memory to store the hierarchical structure. This leads to a new method that involves the particle determination [3] using interaction forces between particles as the main principal. This method, unlike the conventional approaches, offers a significant benefit since it can estimate the interactions forces between particles at all time and not require any complex recalculation every time the objects deform.



Figure 1-2. Cloth simulation [2]

## 1.2 Objectives

The objective of this study is to present the collision detection algorithm that

1. Can effectively detect the collision of the deformable objects
2. Able to detect several objects at the same time.
3. Reduce the errors which might happen from the inappropriate selection of the particles' initial positions.
4. Decrease the errors that might happen from the large deformation of the objects.

## 1.3 Scopes of Study

1. Can be used for both rigid and deformable objects.
2. Be able to investigate the collision of many objects at the same time.
3. Can only used with the objects that represented by polygonal model.
4. Cannot use with a large deformation that tears the object apart.

## 1.4 Research Procedure

1. Study theories
  - The collision detection theories
  - The particle-based theories
  - The surface partitioning theories
2. Research and study the previous work along with the advantages and disadvantages analysis.
3. Design the algorithm.
4. Do the conclusion and suggestions

## 1.5 Expected Benefits

1. The proposed algorithm can be efficiently for detecting the collision of deformable objects.
2. The algorithm is able to investigate the collision of several objects at the same time.
3. The algorithm is suitable with the applications involved with the deformable materials such as the textile, the biological structures or the work that need high precision, for example, Virtual Reality (VR), Surgery Simulation ,etc.

## 1.6 Thesis Structure

This thesis is divided into 5 chapters which are Introduction, Theoretical background and related works, Proposed algorithm, Algorithm analysis, Conclusion discussion and future work.

First chapter, Introduction, provides problem statement, objectives, scope, research procedure, benefits, research structure and publications. Chapter 2 gives a brief description about related theories. Moreover, some previous collision detection algorithms are also discussed in this chapter. In Chapter 3, a proposed collision detection algorithm is presented. After that, the analysis of the proposed algorithm is shown in Chapter 4, follow by the conclusion discussion and future works in the final chapter.

## 1.7 Publications

1. Nida Saenghaengtham and Pizzanu Kanongchaiyos. 2004. Collision Detection Algorithm for Deformable Objects using Particle. The 1<sup>st</sup> Thailand Computer Science Conference (ThCSC 2004), December, Kasetsart University, Bangkok, Thailand.
2. Nida Saenghaengtham and Pizzanu Kanongchaiyos. 2006. A Collision Detection Algorithm Using Particle Sensor. 2006 IEEE International Conference on Robotics, Automation & Mechatronics (RAM2006), June, Bangkok, Thailand.
3. Nida Saenghaengtham and Pizzanu Kanongchaiyos. 2006. Using LBG Quantization for Particle-based Collision Detection Algorithm. Journal of Zhejiang University SCIENCE, June, Zhejiang, China.

## CHAPTER 2

### THEORETICAL BACKGROUND AND RELATED WORKS

In order to have common understanding on basic, theoretical used in this research are briefly explained in section 2.1. Furthermore, some of the most famous collision detection algorithms are described in section 2.2.

#### 2.1 Theory

This section is divided into 3 parts which are a brief theory about collision detection, particle-based method, and vector quantization. These topics are described below.

##### 2.1.1 Collision Detection

The basic idea of collision detection is to check the intersection between objects. This can be logically performs by checking the collision of every object elements such as point, line, and triangle.

##### *Intersection of Triangles*

In order to test the collision of meshes, each triangle has to be checked for the collision. Figure2-1 shows the intersection test of each triangle on object A with each triangle on object B.

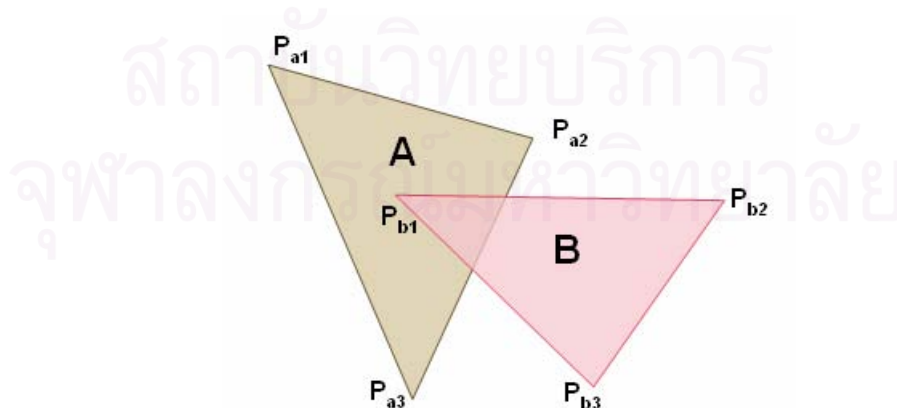


Figure 2-1. Intersection of triangles

The plane that each triangle is lying on can be created as shown in Figure 2-2. An intersection of these two planes is checked. If both triangles lie on the same part of line then the triangles intersect.

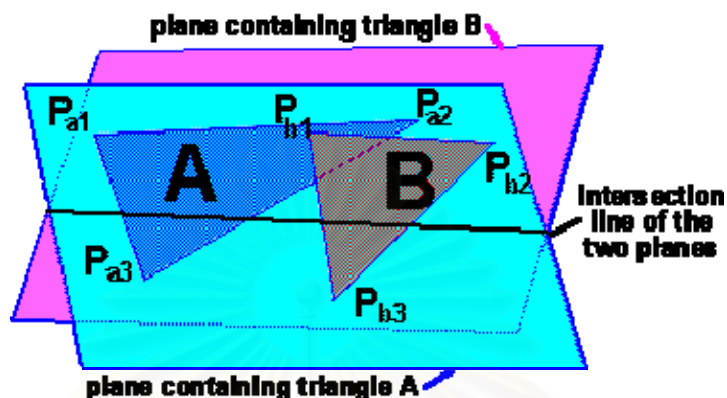


Figure 2-2. Intersection of planes

### *Sphere - Plane Collision*

A way to detect a collision in a 3D world is the sphere-plane detection method. The sphere-plane method is relatively easy to compute since every polygons of a complex model are not required to be checked for collision. Detecting collisions with a sphere tends to be easier to calculate because of the symmetry of the sphere object. The entire surface on a sphere has the same distance from the center, so it is easy to determine whether an object has intersected with a sphere. If the distance from the center of the sphere to an object is less than or equal to the sphere's radius, then a collision has occurred.

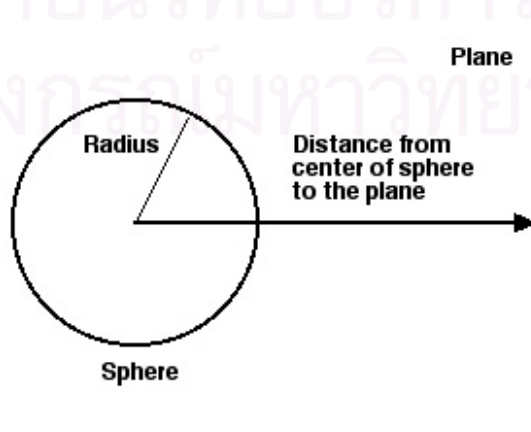


Figure 2-3. Sphere - plane collision.



The main idea is not to let the sphere get too close to the plane. First, every plane needs to have its own normal vector and D value, which are taken from the planar equation.

$$Ax + By + Cz + D = 0 \quad (2-1)$$

The distance between the plane and a vertex, which is the sphere's center in this case, is calculated by taking the dot product of the plane's normal and the sphere's position.

$$\text{Distance} = \text{plane.normal} \cdot \text{sphere.position} \quad (2-2)$$

Depending on which side of the plane the sphere is on, the distance value can be either positive or negative. If the distance becomes zero, then the sphere is intersecting the plane, which is generally not a desirable effect when detecting a collision. This can be corrected by subtracting the sphere's radius from the distance and waiting for an object's distance to reach zero. If the sphere's velocity is high, it might pass entirely through the plane on its next move. The way to check for this situation is to see if the distance to the plane has either turned negative or positive. If the sphere passed through the plane, the distance's numeric sign will change. If the sign changes, it can then be concluded that a collision occurs.

When checking for a collision, there are two importance factors to be considered: the distance between sphere and a plane should not become zero, and the numeric sign of the distance also should not change. If it does change, then the sphere has moved through the wall. The game program will first check if a collision will result when the object moves in the desired direction. If there is a collision, then the program will respond appropriately, such as refusing to move in the desired direction.

### **2.1.2 Particle-based method**

A particle-based method [8, 9] is a modeling and rendering techniques which involve with the control of particles cluster. Each particle has a simple characteristic and behavior. However, they can represent the characteristic of complex-structured objects without any complicated calculation. Therefore, this particle-based method is used for modeling of fluids and complex natural phenomena, for example, cloud, fog, smoke, fire and explosion, etc.



Figure 2-4. Smoke simulation using particle based method

The behavior of each particle is often defined by basic physics principle. Nonetheless, there are a large number of particles which make the calculation more complicated. Hence, the assumption is necessary in order to decrease the calculation complexity; e.g. neglect the collision between particles, ignore the shadows of particles that lay on other particles, etc.

Each particle has attributes which will be random at the beginning, and changed with the calculation, for instance, position, velocity, physical appearance and shape, color, transparency and ages, etc. The changed value can be computed using several methods.

The age of each particle is defined which refer to the number of frames that the particle can stay alive. When a new frame is calculated, the age of particles is then minus by one and decreases continuously until zero, which will be deleted out of the scene.

The particle position can be computed using the rigid dynamic equations. This can avoid the complicated calculation and unfamiliarity of fluid dynamic equation by calculating all forces from environments that applied to particles, for example, gravity force, friction force, wind, etc. Also, it can be calculated from the interaction forces between particles - e.g. spring force. When the mass and forces are known, the velocity and acceleration of each particle can be calculated, at any time  $t$ , from the Newton's law below.

The acceleration is

$$\vec{a} = \frac{d^2 \vec{x}}{dt^2} = \frac{\vec{F}}{m} \quad (2-3)$$

where  $\vec{F}$  = forces applied to particles

$m$  = mass of particles

$\vec{x}$  = particles' position

$\vec{a}$  = particles' acceleration

$\vec{F}$ ,  $\vec{x}$  and  $\vec{a}$  are vectors on the 3 dimensional area ( $x, y, z$ ). The second order equation above can rearranged to the first order equation by

$$\vec{v} = \frac{d\vec{x}}{dt} \quad (2-4)$$

$$\vec{a} = \frac{d\vec{v}}{dt} \quad (2-5)$$

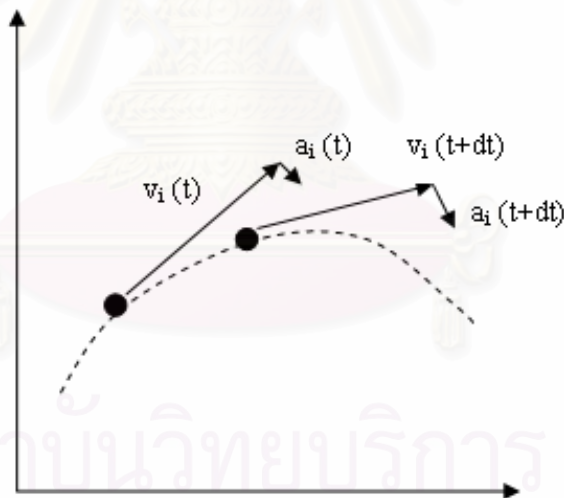


Figure 2-5. Acceleration and velocity of particles on 2-dimentional plain. [8]

The characteristics of particles can be found from various sources such as the color of particles might be controlled by time, the lifetime, height, etc.

### 2.1.3 Vector Quantization (VQ)

Vector quantization [4,5,6] is a method to classify all data into several groups. The value of each group can be represented by its average value.

The diagram bellows is an example of 1-dimantional data division. The numbers between -2 to 0 are determined to be in the same group and its data is represented by -1. The representative data is showing by 2-bits number, so called “1-dimensional, 2 bits VQ”, and having the ratio of 2 bits/dimension.



Figure 2-6. 1-Dimension Vector quantization

An example of the 2-dimensional Vector Quantization (VQ) is shown in Figure 2-7. All data in the same region is put in the same group and is represented by the stars in the picture. In this method, the data is showing by 4-bits number, so called “2-dimensional, 4 bits VQ”. However, the ratio remains constant with 2 bits/dimension.

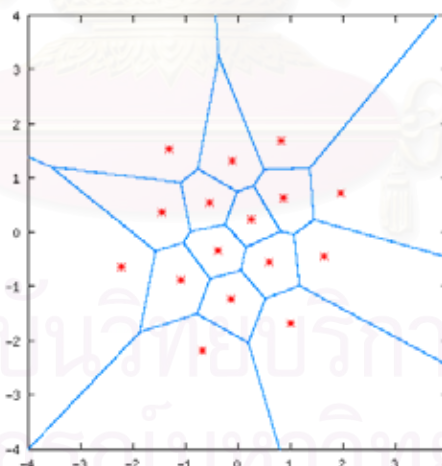


Figure 2-7. 2-Dimension Vector Quantization

From previous two examples, they show that the data is classified into several groups and each of which is represented by the star as shown in the figure. These areas are called “*encoding region*” which are data sets of the data in the same group. The representative value, the group of encoding region, and the set of representative values is called “*code vector*” “*partition of the space*” and “*codebook*”, respectively.

Moreover, each input data is called “source vector” while the set all data is called “*training sequence*”.

Classifying procedures begin with receiving training sequence and set the required number of group. The outcome of this process is code book and partition of the space.

Training sequence

$$\tau = \{X_1, X_2, \dots, X_M\} \quad (2-6)$$

Code book

$$CB = \{C_1, C_2, \dots, C_N\} \quad (2-7)$$

Partition

$$P = \{S_1, S_2, \dots, S_N\} \quad (2-8)$$

When code vector  $C_n$  is a representation of source vector  $X_m$ , it can be written as followed.

$$Q(X_m) = C_n \quad \text{if } X_m \in S_n \quad (2-9)$$

And then, find the error by calculating the squared-error distortion measure as followed.

$$D_{ave} = \frac{1}{Mk} \sum_{m=1}^M \|X_m - Q(X_m)\|^2 \quad (2-10)$$

In the case that the data has k dimension,

$$\|e\|^2 = e_1^2 + e_2^2 + \dots + e_k^2 \quad (2-11)$$

However, the received code vector and partition must satisfy the following 2 criteria, *Nearest Neighbor Condition* and *Centroid Condition*.

- **Nearest Neighbor Condition:**

$$S_n = \{X : \|X - C_n\|^2 \leq \|X - C_{n'}\|^2 \quad \forall n' = 1, 2, \dots, N\} \quad (2-12)$$

This condition means that the data in the same  $S_n$  group must be closer to the representative data  $C_n$  than others.



- **Centroid Condition:**

$$C_n = \frac{\sum_{x_m \in S_n} X_m}{\sum_{x_m \in S_n} 1} \quad n = 1, 2, \dots, N \quad (2-13)$$

While this condition shows that each received representative data  $C_n$  has to be the average value of all data in the same  $S_n$  group and also has to be confirmed that there is at least one data in each group in order to ensure that the denominator of the above equation will not equal to zero.

***LBG Design Algorithm (Linde-Buzo-Gray) [7]***

LBG algorithm is an effective algorithm for finding codebook in data classification process which follows two previous conditions. The algorithm begins with receiving set of data, source vector, and then find out the initial codebook ( $C^{(0)}$ ) using a random method or a splitting technique. After that, each code vector is gradually updated and the squared error distortion ( $D_{ave}$ ) is checked until all code vectors become stable. This iteration process are continued until achieve required number of code vectors.

Code vectors are computed using the splitting technique as follow.

1. Find the first code vector from the average value among all source vectors.

2. Define the first code vector as a reference code vector ( $C_1^*$ ) to create the other two code vectors from these following equations.

$$C_1^{(0)} = (1 + \varepsilon)C_i^* \quad (2-14)$$

$$C_2^{(0)} = (1 - \varepsilon)C_i^* \quad (2-15)$$

The number in parenthesis shows the number of times that code vector is adjusted.

3. Classify each source vector into group of closest code vector using the nearest neighbor condition. Then, calculate the average value to find updated code vectors,  $C_1^{(1)}$  and  $C_2^{(1)}$ .

4. Repeat the procedures until

$$C_1^{(i)} \cong C_1^{(i-1)} \text{ and } C_2^{(i)} \cong C_2^{(i-1)}$$

which means that all code vectors become stable

5. Define these code vectors as reference code vectors ( $C_1^*, C_2^*$ ) in order to create other 2 code vectors for each reference code vector, as in the 2<sup>nd</sup> procedure, follow by grouping and finding updated code vectors, as in 3<sup>rd</sup> procedure until all code vectors become stable. Repeat all procedures until the required number of code vectors is achieved.

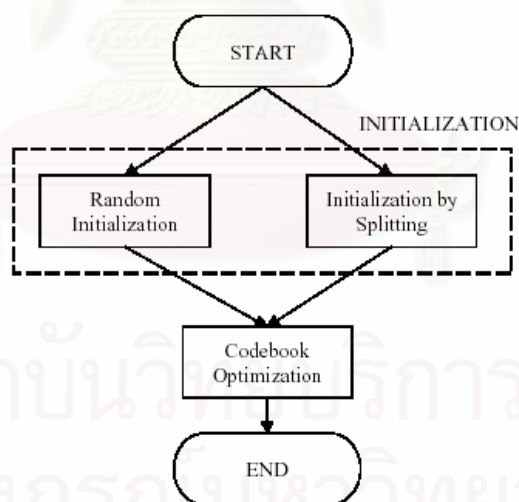


Figure 2-8. The LBG procedure

## 2.2 Related works

A collision between a pair of objects occurred when parts of objects intersect with each other. Basically, a collision can be detected by finding the intersection between lines, plane, or triangle. However, when object has a very complex structure, the determination of every intersection takes very long computational time and thus cannot be processed in real time. Therefore, there are a lot of researches have been proposed on the collision detection. The advantages and disadvantages of some interesting methods are described as follow.

### 2.2.1 Sphere

Sphere [10, 11, 12, 13] is a collision detection algorithm which approximates each object as a sphere and checks whether the spheres intersect each other. The determination can be achieved by measuring the distance between the centers of spheres. If the distance is shorter than the summation of two spheres radiuses, it can be concluded that two spheres are intersected and hence the objects might be collided. On the contrary, if the distance is longer than the summation of radiuses, it can be concluded that there is no collision occurs.

This method uses a very easy, and simple, calculation, but also gives the unreliable outcome since the sphere cannot approximate the object efficiently. There are some gaps inside the sphere that do not contain any part of the object and thus might cause an error in the collision detection. As shown in Figure 2-9, in this method, two spheres can intersect each other without any collision occurs.

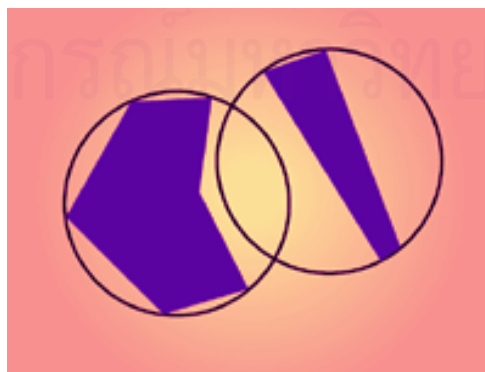


Figure 2-9. Error from using sphere collision detection. [13]

However, the efficiency can be improved by subdividing a sphere into many small spheres, the basic idea of hierarchy and subdivision, so that it can better approximate the object. The detection begins by checking the intersection between spheres. If there is no intersection, it can be concluded that there is also no collision between the objects. On the other hand, if the intersection occurs, there will be a sub-investigation on each sphere and this will continue until the required efficiency is met as shown in Figure 2-10.

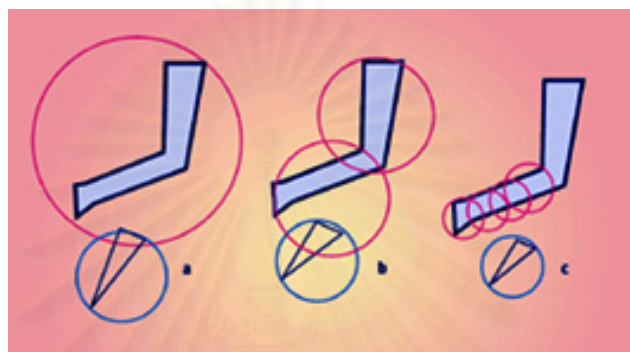


Figure 2-10. Sphere dividing method for determination of the objects. [13]

### 2.2.2 Axis-Aligned Bounding Boxes (AABBs)

Since most objects used in computer graphic have a characteristic like a box, there is an idea to use box to approximate the object. This method is called “*Axis-Aligned Bounding Boxes*” (AABBs) [13,14,15,16] The word “*Axis-Aligned*” shows the alignment of box which used to approximate the object lying on the world axes. Each side of the box must perpendicular to one coordinate axis. This method is easy and fast and, as a result, is very popular.

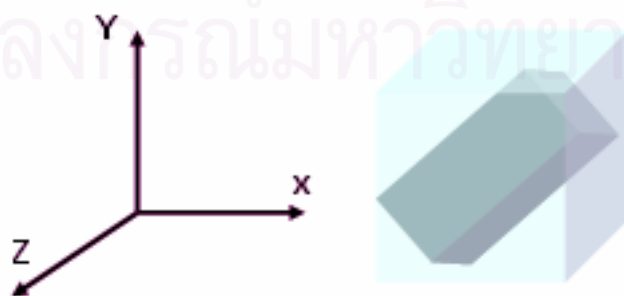


Figure 2-11. The AABB box arrangement.

Nonetheless, the principle of this method is that box has to align only with the world axes and thus cannot move along with the object when the object rotates. Therefore, a box has to be recalculated to cover the object every time the rotation occurs as shown in Figure 2-12 and 2-13. These figures show that size of box changes with the object alignment. Moreover, if the object deforms, AABB box also has to be recalculated.

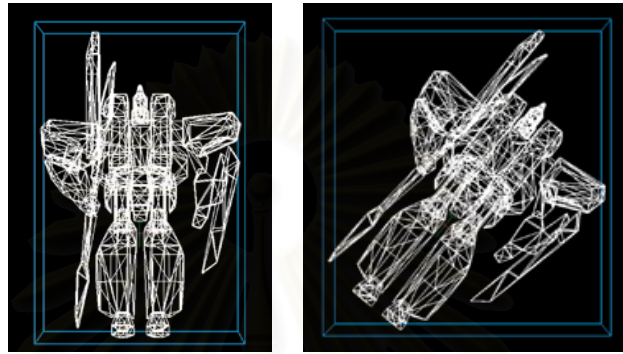


Figure 2-12. New box creations when the objects rotate [13]

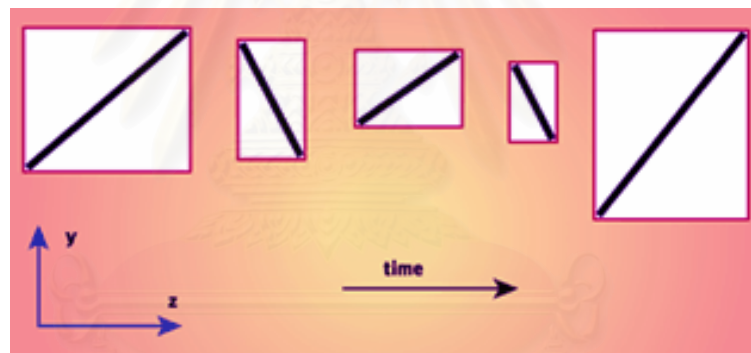


Figure 2-13. A linear stick and its AABB box in various directions. [13]

Although the recalculations to find a new box is not difficult and does not slow the procedure down much, there is another problem of using this method. The problem is the imprecision of results which is the same problem met in the sphere method. AABB box also cannot exactly fit the object shape and there are also some gaps which can false the collision detection.



### 2.2.3 Oriented Bounding Boxes (OBBs)

As object cannot exactly fit by the sphere or AABB box, there is an idea of trying to find the new arithmetic shape that can improve the approximation of the object by reducing empty gaps to the least. A new method which called “*Oriented Bounding Boxes*” is created by using the box that can rotate along with the object. With this method, the empty gaps can be reduced and also can increase precision of the detection.



Figure 2-14. The OBB box arrangement [13]

This method can detect the collision more precise and also more efficient than the spheres and the AABB method. However, the calculation is much more complex which slows down the detection procedure especially when using with the object that has unstable surface. Hence, this method is not suitable with the deformable object.

Like other methods, this technique can increase the precision and efficiency of collision detection by creating subdividing sequence of the detection, so called *Hierarchical methods*. The calculation is also more complicated in order to find the box that fits the object best. The volume is divided into two parts along with the longest axis (if cannot, then divided along the second longest axis). After that, subdivided boxes are checked whether it intersect with others. If there is an intersection, the dividing and checking procedures are repeated until the required precision is achieved as shown in Figure 2-15.

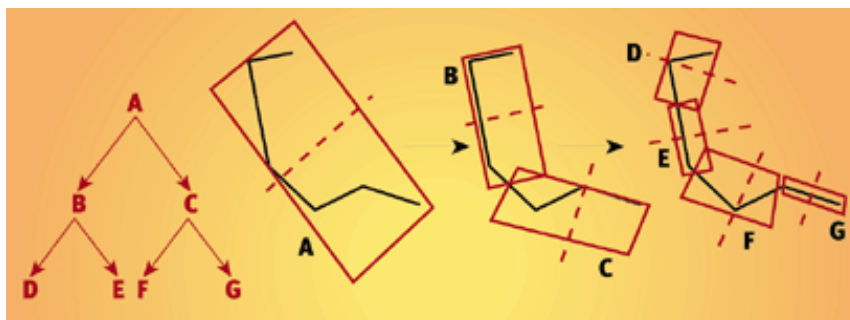


Figure 2-15. Hierarchical methods for OBBs

The advantage and disadvantage of each collision detection algorithm can be summarized as shown in Table 2-1.

Table 2-1. Advantages and disadvantages of each collision detection algorithm

Algorithm	Advantage	Disadvantage
Sphere	<ul style="list-style-type: none"> <li>- Simple calculation</li> <li>- Easy to implement</li> <li>- Few space needed</li> <li>- No need to calculate when object rotates</li> </ul>	<ul style="list-style-type: none"> <li>- Very coarse detection since sphere cannot fit the object efficiently</li> </ul>
AABB	<ul style="list-style-type: none"> <li>- Simple calculation</li> <li>- Easy to implement</li> <li>- Few space needed</li> </ul>	<ul style="list-style-type: none"> <li>- Fairly coarse detection since AABB cannot fit the object efficiently</li> <li>- Have to recalculate when rotation and deformation occurs</li> </ul>
OBB	<ul style="list-style-type: none"> <li>- More accurate than Sphere and AABB</li> <li>- Few space needed</li> </ul>	<ul style="list-style-type: none"> <li>- Difficult to implement</li> <li>- Have to recalculate when object deforms</li> <li>- Inappropriate for deformable object</li> </ul>
Bounding hierarchical	<ul style="list-style-type: none"> <li>- Very high accuracy</li> </ul>	<ul style="list-style-type: none"> <li>- Difficult to implement</li> <li>- Take very long computational time</li> <li>- Requires much memory to store hierarchical structure</li> <li>- Inappropriate for deformable object</li> </ul>

### 2.2.4 Particle-based Collision Detection

Most algorithms can be used only with the solid objects that have a constant surface, for instance, Hierarchical methods which have to recalculate every time the surface deformation occurs. This leads to an alternative algorithm that involves the particle determination used the interaction forces between particles as the main principal. This method, unlike the conventional approaches, offers a significant benefits since it can calculate the interactions forces between particles every time deformation occurs without any recalculation.

The collision detection, with this particle-based method [3], used the attractive interactions between particles that spread over objects surface as the main principle. Particles will be randomly spread over the surface and then charged with same-charged ions, for the particles on the same objects, and different-charged ions, for the particles in distinct objects. The force, occurs on each particle, is then calculated from

1. The attractive force between different charge particles, in order to pull particles on different objects into each other when the distinct objects move closer.
2. The repulsive force between same charged particles, in order to equally spread particles all over the surface and prevent particles aggregation in either side of the object.

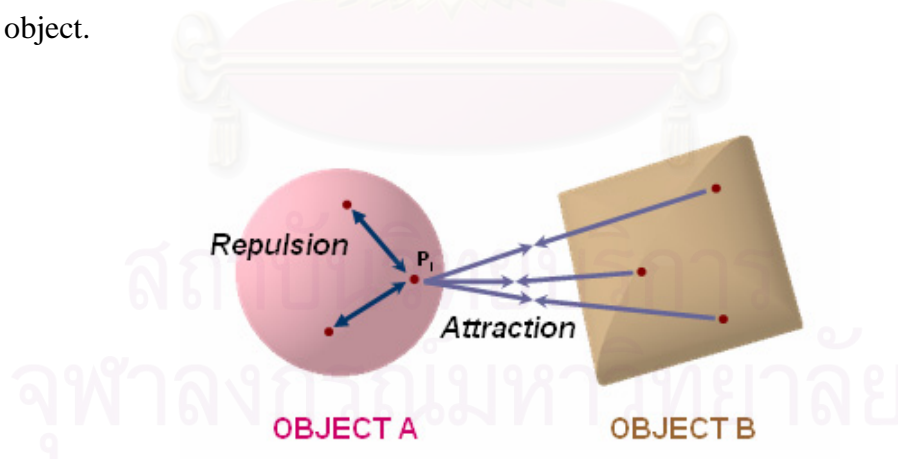


Figure 2-16. Collision detection using Particle-based method

Particle position, at any time  $t$ , can be calculated by the summation of attractive and repulsive forces. The collision can be detected by measuring distance between particles. When particles come closer than a tolerable value, it can be

concluded that there is high possibility that the collision occurs. Therefore, a collision is then investigated precisely between those parts of objects.



Figure 2-17. Particle distribution

Figure 2-17 shows two polygonal models of the letter "T" defined by 1380 polygons and the distribution of 3 particles on each model which can define a sufficiently correct collision point.



Figure 2-18. Collision detection between two models using particle-based method.

Figure 2-18 shows collision detection between two polygonal models of letters "W", defined by 11920 polygons, with 5 particles on each model. The average time needed to perform a collision detection query was 12.67 ms., and the maximum time was 47 ms.; average time during the "warming" steps was 12.06 ms., maximum time was 47 ms. also.

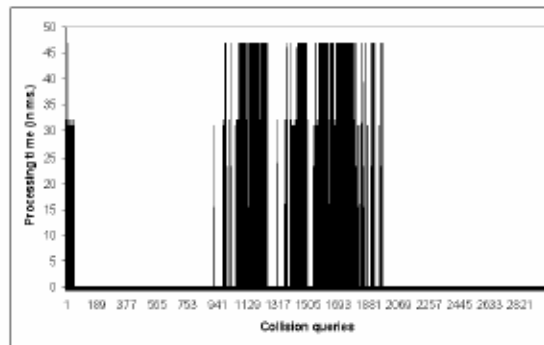


Figure 2-19. Time to perform a collision detection query

However, when there are more than 2 objects in the system, this technique might not sufficient enough since a situation where particles receive high attractive forces and then disperse together in either side of the object. As a result, the other sides will lack of particles to detect collision and if there is another object crashes on this side, there will not have enough attractive forces to pull the particles back and hence cannot detect collision.

Moreover, as the number of particles is initially fixed, this method might have less efficiency when the object has large deformation such that surface areas increase dramatically and cause the insufficiency of particles to detect the collision all over the objects.

Besides, random placement of particles onto object surface might leads to the improper of the particle dispersion and thus decreases the collision detection efficiency. For example, as shown in Figure 2-20, three particles are randomly disperse at the right hand side of the V-shape object and cannot move to the other side to detect a collision when there is another object crashing on the left hand side.

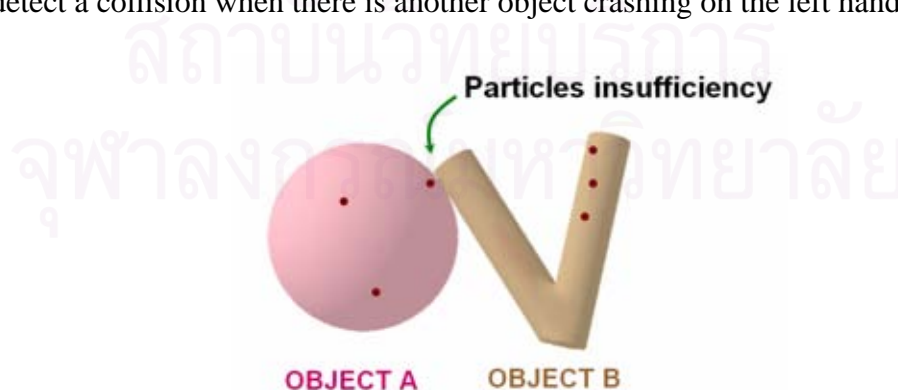


Figure 2-20. Improper particle dispersion at the initialization

## CHAPTER 3

### PROPOSED ALGORITHM

In this research, an algorithm to improve the efficiency of particle-based collision detection is presented. In order to eliminate the inappropriate dispersion problem of the particles, object surface is divided into several small areas, each of which will be filled with a particle. These particles can move only inside their area regions and cannot move across to any other areas. With this technique, it can ensure that particles always cover all over the object. Therefore, this can eliminate the problem of insufficient particles when there is more than one object crashing on the other sides at the same time.

After that, the attractive forces are equipped between particles on different objects. Consequently, when objects come closer to each other, particles on each object will also move along the surface into each other. The distance between two particles can be observed. If the distance is shorter than a tolerable value, it can then be concluded that there is a high possibility for a collision. The collision is then determined precisely between the specific region.

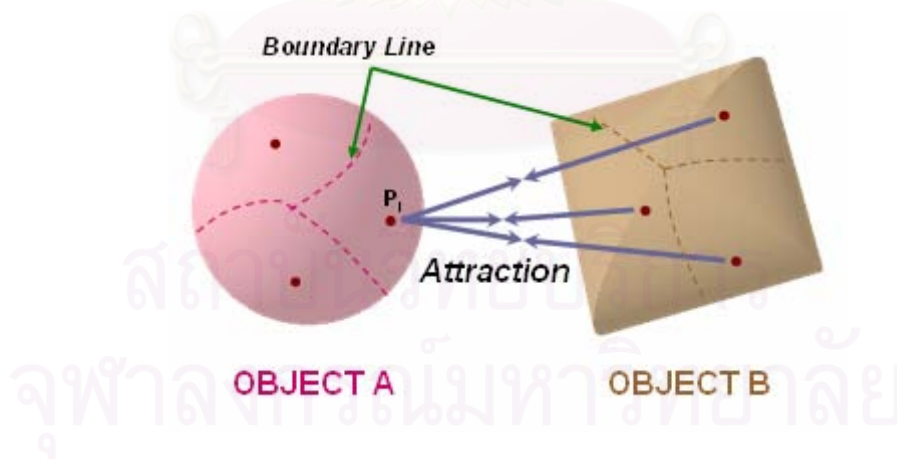


Figure 3-1. A new algorithm developed for the collision detection

This method provides each particle to detect collision in each separated region, prevent the agglomeration of particles in either side of the surface which leads to the insufficient of particles to detect the collision when other objects come from other directions. Moreover, it can also cut off the attractive forces between particles on the same object which prevent the same problem in the conventional method.



### 3.1 The number of particles

In order to ensure that there are enough particles to detect several collisions at the same time, a suitable number of particles has to be set equal to the approximated number of objects that can touch the surface. The basis of *Kissing number* [18] is then applied to find this approximated value.

Kissing number or Newton number is the number of equivalent spheres which can touch an equivalent sphere without any intersections. First proof was produced in the 1993 by Conway and Sloane that the kissing number in three dimensions was 12, as shown in Figure 3-2.

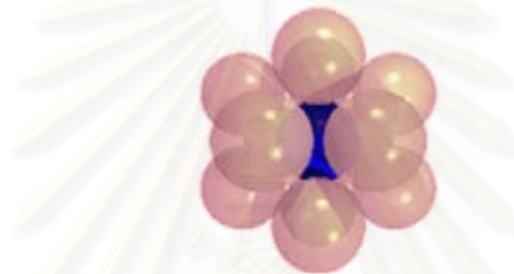


Figure 3-2. Kissing number.

It can be concluded that each sphere object can be collided by the same size spheres with no more than 12 objects at the same time. Therefore, when there are only equivalent sphere objects in the scene, 12 can be used as the number of particle for each object.

However, when there are several sizes of spheres, this assumption might not be longer sufficient. This principle is then applied by using the maximum number of smallest sphere in the scene that can touch the object as the required number of particles. This value can be estimated as follow:

Let  $X$  is the considered sphere object and  $Y$  is the smallest in the scene. The whole arrangement is deposited into an imaginary sphere, as shown in Figure 3-3. A lamp is then imagine at the centre of object  $X$  which cast shadows of surrounding spheres onto inside of the imaginary sphere. Each circular shadow has an area of  $B$  and cannot overlap. The number of sphere objects that have the same size as  $Y$  which can touch the sphere object  $X$  can then be estimated as  $A/B$  when  $A$  is the surface area of the imaginary sphere. Therefore, the number of particles needed for  $X$  is not more than  $A/B$ .

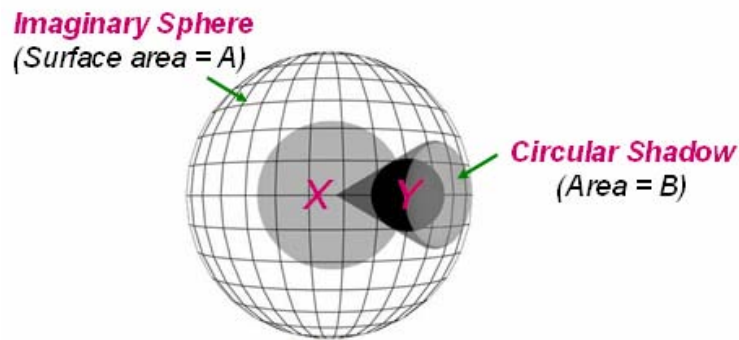


Figure 3-3. Circular shadow on the imaginary sphere.

This technique can also be used with the object that has other shapes apart of sphere. However, a modification is required before this technique is applied. Each object is approximated as a collection of spheres by analyzing model geometry which can be classified into 2 cases which are

*Convex object* - In geometry, convex is an object which has no interior angle greater than  $180^\circ$ . In this algorithm, each convex object is approximated as a sphere as shown in Figure 3-4 a.

*Concave object* - In geometry, concave is an object which has interior angle greater than  $180^\circ$ . In this algorithm, each convex object is approximated as a collection of spheres as shown in Figure 3-4 b.

After that, the suitable number of particles can be achieved for each part of the object.

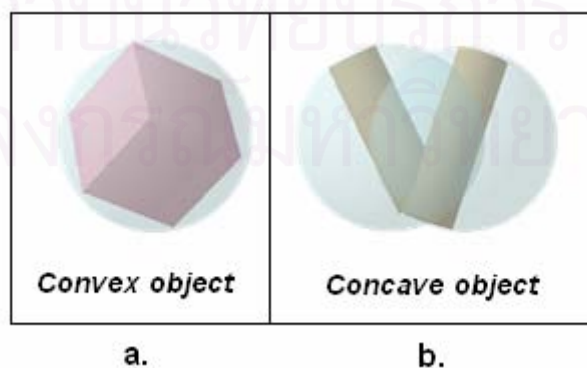


Figure 3-4. Approximated spheres for a. convex object, and b. concave object

However, several numbers of particles are not necessary when there are only two convex objects in the system. As there cannot be more than one collision between a pair of convex objects at the same time, each object therefore requires only a particle to detect the collision. As a result, it can be concluded that when the process consist of two convex, only a particle is needed for each object.

### 3.2 Surface Partitioning

The procedure of this process is to partitioning surface area into several regions. A particle is then put in each region to detect a collision in each separated area. A partitioning method is firstly chosen in order to achieve great partition.

Most surface partitioning algorithms are based on the idea of classifying surface element into groups as shown in Figure 3-5. Several classification techniques have been proposed, for example, a technique proposed by J. Shen and D. Yoon [19]. The concept of this technique is to performing a breadth-first search to propagate over the surface. First, vertex's adjacency information is created which provides a list of neighbor vertices for each vertex. An arbitrary surface element is randomly selected as an initial vertex for the first group. Each neighboring element is then gradually put into the group using the breadth-first search until reaching a specified condition. The following vertex that cannot put in the group is subsequently set as an initial vertex for the next partition. The breadth-first search is repeated over unprocessed regions until all surface elements are covered by a partition.

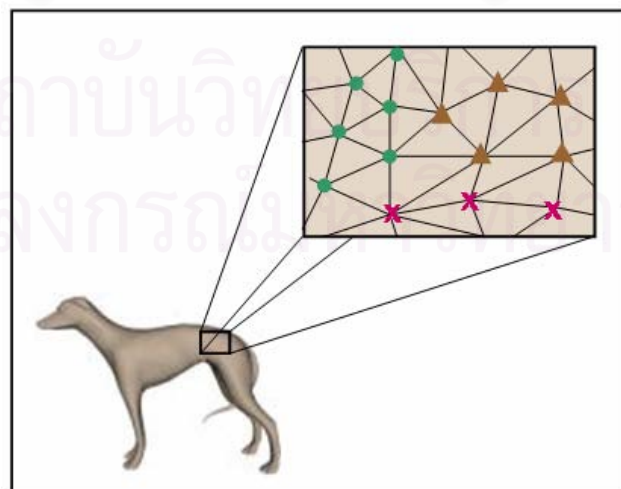


Figure 3-5. Vertex classification

However, this technique has to consider each factor separately which can be complicated and inefficient. For this reason, the proposed algorithm employs an alternative method in order to achieve effective partitioning process.

LBG quantization is a grouping technique which classifies vector data into several groups. The value of each group can be represented by its average value called code vector as shown in Figure 3-6. As the input data can be unlimited dimension vector, this technique can therefore efficiently partition object surface by concurrently considering several factors such as vertex position, normal vector, and color, etc. Each determinant is arranged into each vector dimension which, in this algorithm, there are 2 factors to consider as will be described later.

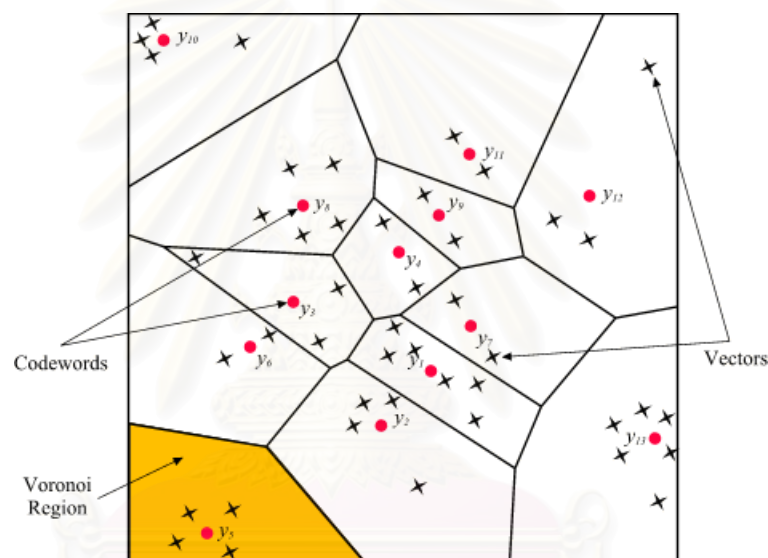


Figure 3-6. Vector quantization

According to the advantage of the LBG quantization, the proposed algorithm applies this technique to partitioning object surface by classifying vertices into groups. Each group is then assigned with a particle moving only between vertices in the same group. This seems like object surface is divided into several area.

However, this technique can cause a problem when using with a concave object. Some vertices might be classified into the same group even though there is no path for the particle to move between these vertices. An example of this case is the surface partitioning of a fork-shape object as shown in Figure 3-7. The vertices in the circle as shown are set in the same partition since their position and normal vector are close to each other. This can cause an error when there is another object crashing on the position as the arrow shown in the figure.

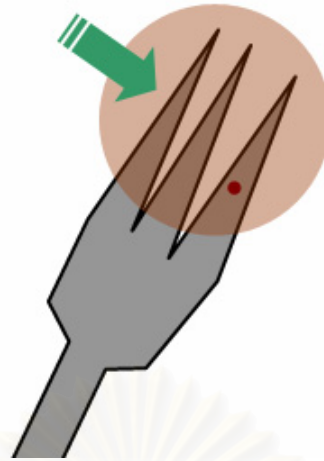


Figure 3-7. Improper partitioning of LBG with a concave object

Fortunately, each object is already divided into several convex parts as described in the previous section. Therefore, a classification process using LBG quantization is performed to each separated part of the object that was created as described in Section 3.1. The number of partitions for each part can be simply found. Since a particle is assigned to control in each region, the number of partitions for each part is therefore being the same number as the required number of particles in the corresponding part which is already calculated.

As mentioned above, this algorithm considers 2 factors for classifying a surface. Since some closed adjacent vertices should not be included to the same partition due to their different planes as shown in Figure 3-8, the partitioning should not be depended only on their positions, but should also on their normal vectors. Therefore, each input vector for the classifying process is a 6-dimensional vector composed of vertex coordinate and its normal vector.

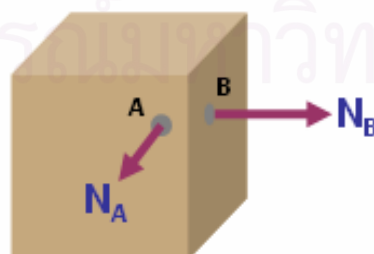


Figure 3-8. Two close adjacent vertices having different normal vectors



### Classification Procedure Using LBG Quantization

According to the LBG quantization theory as described in Chapter 2, the classification process can be performed by finding average value among all input vector. The average value can split data into 2 groups which each group can be split continuously until the required number of groups is achieved. This process can be described technically as follow.

1. Collect each vertex of the objects as a 6-dimensional vector form

$$\text{Source Vector: } V_m = (v_1, v_2, v_3, v_4, v_5, v_6)$$

Where  $v_1, v_2, v_3$  represent vertex position, in 3-dimensions, on the  $(x, y, z)$  axes, while  $v_4, v_5, v_6$  represent its normal vectors.

The set of all source vectors is;

$$\text{Training Sequence: } (\tau) = (V_1, V_2, \dots, V_M)$$

2. Fixed  $\varepsilon > 0$  to be a small value.
3. Find out the first initial code vector by fixing the number of the required code vectors  $N = 1$ . Then calculate the average value of all source vector and get;

$$C_1^* = \frac{1}{M} \sum_{m=1}^M V_m \quad (3-1)$$

After that, calculate the squared error distortion value at point  $C_1^*$ ;

$$D_{ave}^* = \frac{1}{6M} \sum_{m=1}^M \|V_m - C_1^*\|^2 \quad (3-2)$$

4. Classify source vector into several groups using the reference code vector,  $C_i^*$ , as a divider and find the initial code vector of each group.

Repeat the procedures for all code vectors used as dividers,

For  $i = 1, 2, \dots, N$

$$C_i^{(0)} = (1 + \varepsilon)C_i^* \quad (3-3)$$

$$C_{N+i}^{(0)} = (1 - \varepsilon)C_i^* \quad (3-4)$$

For example, according to point  $C_1^*$ , source vector can be classified into two parts – and able to compute a code vector for each part as follows

$$C_1^{(0)} = (1 + \varepsilon)C_1^* \quad (3-5)$$



$$C_2^{(0)} = (1 - \varepsilon)C_1^* \quad (3-6)$$

5. Double the  $N$  value by using

$$N = 2N$$

6. The iteration process to find the appropriate code vector

6.1 Classify the source vectors into the groups and compute a code vector for each group as shown in 4<sup>th</sup> step. Source vector will be grouped by the Nearest Neighbor Condition Principle. To find the nearest vector elements, *Euclidian distance* is used as a measurement function.

Let iteration index  $i = 0$

For  $m = 1, 2, \dots, M$  Find the lowest value of

$$d(C_n^{(i)}, V_m) = \sqrt{\sum_{j=1}^6 (C_{nj} - V_{mj})^2} \quad \text{For } n = 1, 2, \dots, N \quad (3-7)$$

If  $C_{n^*}^{(i)}$  is a code vector that create the lowest value, then put it into the group.

$$Q(V_m) = C_{n^*}^{(i)} \quad (3-8)$$

6.2 When grouping is finished, find a new code vector that get from the average in each group.

For all value  $n = 1, 2, \dots, N$

$$C_n^{(i+1)} = \frac{\sum_{Q(V_m)=C_n^{(i)}} V_m}{\sum_{Q(V_m)=C_n^{(i)}} 1} \quad (3-9)$$

6.3 Set  $i = i + 1$

6.4 Calculate

$$D_{ave}^{(i)} = \frac{1}{Mk} \sum_{m=1}^M \|V_m - Q(V_m)\|^2 \quad (3-10)$$

6.5 Check the code vector whether it already stable by considering

$$\frac{D_{ave}^{(i-1)} - D_{ave}^{(i)}}{D_{ave}^{(i-1)}} \leq \varepsilon \quad (3-11)$$

If process still not meets the condition, it means that different between  $C_n^{(i)}$  and  $C_n^{(i-1)}$  are too much and thus this cycle has to be repeated until all code vectors is stable.

7. When code vectors are already stable, take it as reference ( $C_i^*$ ) for splitting procedures in the next cycles.

$$\text{Let } D_{ave}^* = D_{ave}^i$$

For  $n = 1, 2, \dots, N$  Set  $C_n^* = C_n^{(i)}$

8. Repeat the 6<sup>th</sup> and 7<sup>th</sup> cycles until reaching the required number of code vectors.

9. Classify source vector ( $V_m$ ) into each group as described in step 6.1.

Finally, the  $N$  groups of 6-dimensional vectors are achieved.

Each 6-dimensional vector is then converted back to a vertex element by extracting the first 3 elements of the vector. Lastly,  $N$  groups of vertices,  $N$  areas, can be achieved.

10. Put a particle onto a vertex in each group.

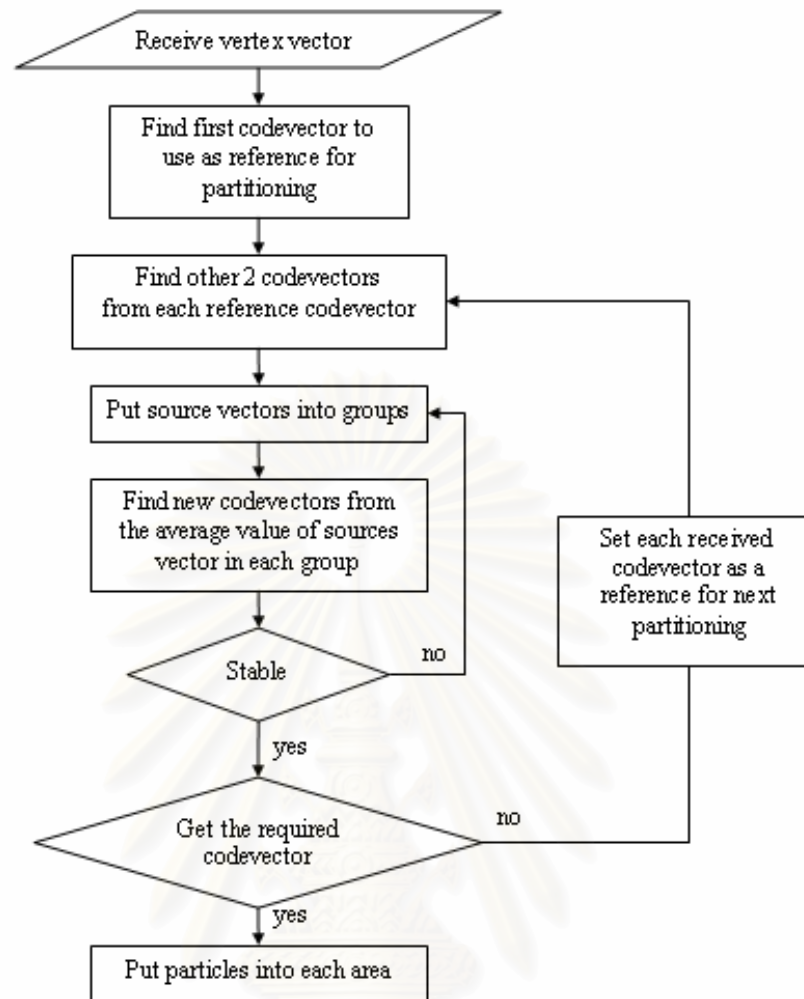


Figure 3-9. Flow chart shows the partitioning

### 3.3 Interaction forces

As mentioned, the main idea of this algorithm is to imply each particle as a sensor to determine the collision by checking the distance between particles. A collision is concluded to be occurs when the distance between a pair of particle is less than a tolerable value. It is obvious that when object come closer to one another, their corresponding particles also have to be pulled into each other in order to decrease the distance between this pair of particles. Therefore, attractive force is created between particles on different objects to achieve this purpose. The attractive force corresponds to the distance between particles as represented in the following equations. The force increases when particles move closer to each other, decreases when particles move farther and finally becomes zero when the distance between particles is longer than an

effective radius of attraction. In this research, the effective radius of attraction is preliminary calculated as the longest distance between initial positions of particles.

$$f(p_i, p_j) = 0 \quad \text{When} \quad r \geq R_{eff} \quad (3-12)$$

$$= \frac{1}{r^2} \quad \text{When} \quad r < R_{eff} \quad (3-13)$$

When  $f(p_i, p_j)$  = Attractive force between particle  $i$  and  $j$

$r$  = Distance between particles

$R_{eff}$  = Effective radius of attraction

When consider the forces acting on particle  $i$ , there are attractive forces from other particles on the different objects as shown in Figure 3-10. Hence, the total force which acts on particle  $i$ , is the vector summation of attractive forces on particle  $i$  emanating from all other particles which can be represented in Equation 3-14.

$$F_{p_i} = \sum_i^N f(p_i, p_j) \quad (3-14)$$

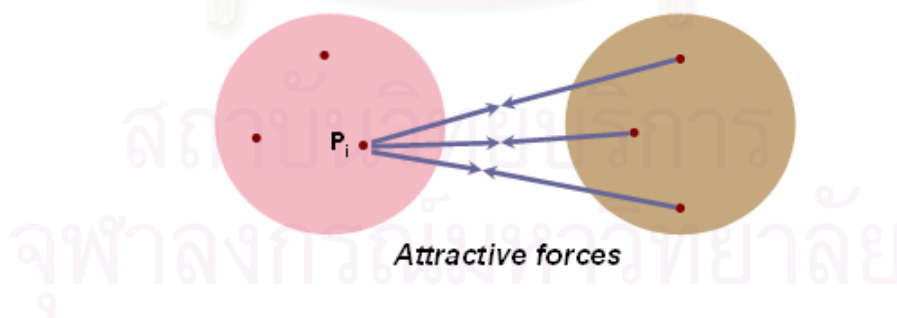


Figure 3-10. Attractive forces acts on particle

### 3.4 Movement of particles

In the simplest case, particles are limited to be located only at vertices within its specific area. Particles are assigned to move along the vertices to the neighboring vertex which receives maximum force ( $V_{\max}$ ). This can ensure that each particle always tend to move to a collision point.

As a particle's movement can be computed by considering position of other particles, moving a particle can cause effects to all others. Therefore, in order to update particle position, the movement has to be computed for all particles at the same time. In this algorithm, the movement of particles is computed frame-by-frame by finding a destination vertex for each particle. However, a particle can not be moved until all particle destinations are found. In the other word, the movement for each particle can be computed by assuming that other particles have no movement. After all destination vertices are found, all particles are moved to its corresponding destination.

In order to choose a particle destination, we have to know the particle position ( $P_i$ ) and its neighboring coordinates ( $Q_{i,j}$ ) as shown in Figure 3-11. The forces acting on these vertices are calculated as described in the previous section. These forces are then compared to each other to find the vertex which receives the maximum force ( $V_{\max}$ ). This vertex cannot suddenly set as a destination vertex ( $V_{d,i}$ ) since it might stays outside the boundary. Therefore, this vertex has to be checked whether it belong to the same area as its original position as shown in Equation 3-15. If the vertex is not outside the boundary, it is therefore set as the particle's destination as described in the Equation 3-16.

$$Q(V_{\max}) = Q(P) \quad (3-15)$$

$$V_{d,i} = V_{\max} \quad (3-16)$$

This can be concluded that a particle will have no movement when the  $V_{\max}$  received is outside the area or when it is the same vertex as its original position ( $P_i$ ) which can be expressed as follow.

$$\text{No movement} = [Q(V_{\max}) \neq Q(P_i)] \vee [V_{d,i} = V_{\max}] \quad (3-17)$$

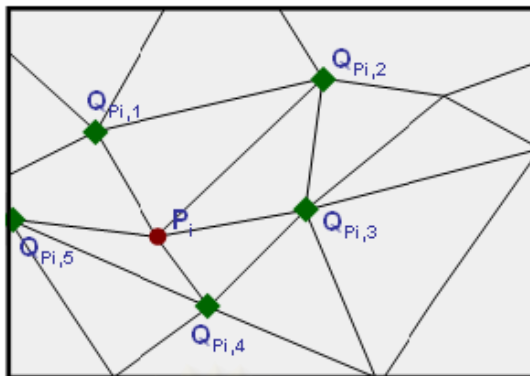


Figure 3-11. Particle position and its neighboring coordinates

However, when objects move very fast to each other, limiting particle to move only one vertex each time might not sufficient enough to catch up with the collision. A collision might not be detected since particles can not move to a collision point in time as shown in Figure 3-12. As a result, this algorithm allows particles to move continuously along the vertices until their positions become stable. The procedure of this process can be described in the chart shown in Figure 3-13.

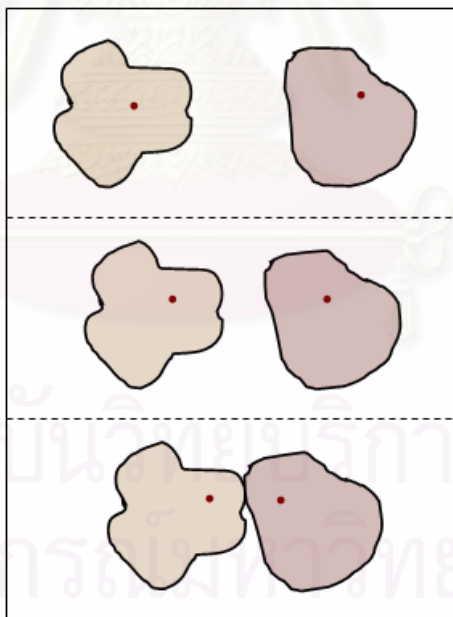


Figure 3-12. Undetected collision when particles can move only one vertex each time



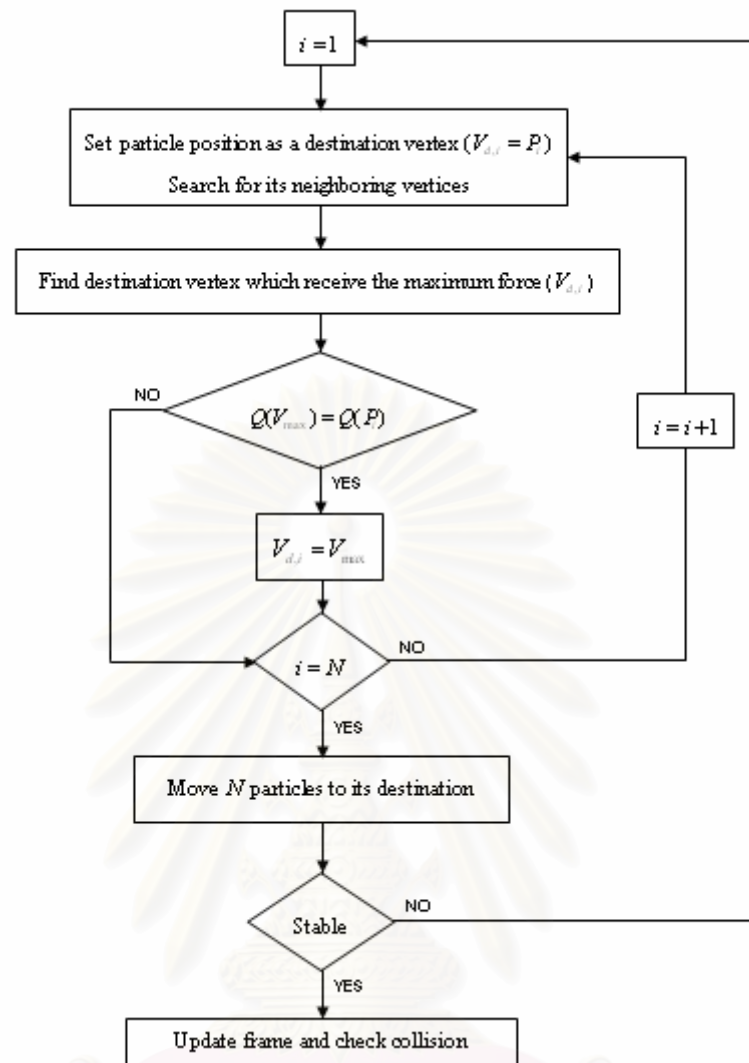


Figure 3-13. Flow chart shows the movement of particles

According to the flow chart shown in Figure 3-13, the movement of all  $N$  particles can be described as follow. Each particle has to be considered to find its destination vertex. First, a particle position ( $P_i$ ) is set as the first destination vertex ( $V_{d,i}$ ). From this position, its neighboring vertices ( $Q_{i,j}$ ) can be found. The force acting on these vertices, including at the particle position, are then compared to each other. The vertex which receive maximum force ( $V_{max}$ ) is checked whether it belong to the same area as the particle position. If it is still inside the boundary, this vertex is then set as a new destination vertex ( $V_{d,i}$ ). However, if it is outside the boundary, the destination vertex is not updated and therefore still is the same vertex as the particle position ( $P_i$ ). This procedure is repeated until the destination vertices for all  $N$

particles are found. After that, particle position can be updated by moving all  $N$  particles to its corresponding destination. For each movement loop, the process is checked for the stable state. When it becomes stable, collision is checked and the frame is finally updated.

### 3.5 Collision Distance

For each collision detection process, the attractive forces between particles  $F_{p_i}$  have to be calculated until stable state is reached. Then measure the distance between particles that have the strongest force which is a pair of the closest particles. Finally, the distance is compared to the collision distance value ( $\mu$ ) which is the average distance between object vertices as described in the following equations.

Given each vertex of the objects is a 6-dimensional vector

$$V_m = (v_1, v_2, v_3, v_4, v_5, v_6)$$

And the set of all vertices is

$$\tau = (V_1, V_2, \dots, V_M)$$

The collision distance value can be calculated from

$$\mu = \text{avg}(V_i - V_j) | V_i, V_j \in \tau \quad (3-18)$$

If the distance is shorter than  $\mu$ , which means there is high possibility that the collision occurs, collision is then precisely checked at the corresponding area.

### 3.6 Repartition Checking

As general objects used in computer graphics do not deform noticeably, repartitioning does not have to be processed every time the object deforms. However, when the object greatly deforms, some area might become too large so that only one particle is not enough for detecting collisions. Hence, object surface should be repartitioned when the deformed area is larger than an acceptable area ( $A^*$ ).

The investigation of each area can be achieved by measuring the distance along the vertices from a particle to the edge around that area. In this research, the measuring is performed in four directions,  $+X$ ,  $-X$ ,  $+Y$  and  $-Y$ , as shown in Figure 3-

14. A rectangular area,  $A$ , can be estimated as shown in the figure. This approximated area is then compared to the acceptable area ( $A^*$ ) which, in this algorithm, is set to the largest approximated rectangular area among of all initial areas. If the computed area is larger than the acceptable value, the repartitioning process for the entire object is then required.

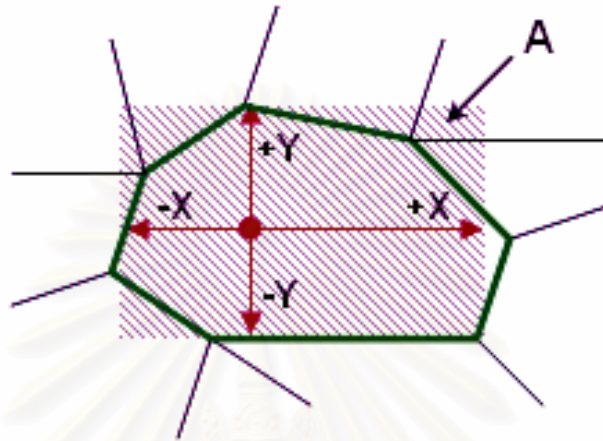


Figure 3-14. Four directions used to find size of the area.

The distance from particle to the edge of each area,  $i$ , can be calculated as follow.

1. Receive a particle position ( $P_i$ )
2. Set the particle position as the first reference point ( $P_i^*$ )

$$P_i^* = P_i$$

Set initial distance  $S = 0$

3. Find neighboring vertices around the reference point ( $P_{ij}^*$ ).

$$P_{ij}^* \text{ when } j = 1, 2, \dots, J$$

$J$  = number of the vertices connected to the reference vertex ( $P_i^*$ )

Find vertex  $q$ ,  $q = P_{ij}^*$ , that makes the lowest angle between  $\vec{V}_{ij}$  and unit vector along the  $+X$  axis of the particle ( $+\vec{X}$ ). (or  $-\vec{X}$ ,  $+\vec{Y}$ ,  $-\vec{Y}$ ) When  $\vec{V}_{ij}$  is the unit vector that has the direction away from the origin to any vertex,  $P_{ij}^*$ .

The angle  $\theta$  is calculated from

$$A \bullet B = |A| |B| \cos \theta \quad (3-19)$$

$$\theta = \cos^{-1} \frac{AB}{|A||B|} \quad (3-20)$$

$$\therefore \theta_{ij} = \cos^{-1}(\vec{V}_{ij}) \cdot (+\vec{X}) \quad (3-21)$$

Figure 3-15 shows a tracking procedure along +X axis by following vertices in the +X direction. First, neighboring vertices are found. At the vertex  $P_{11}^*$ , the angle between vector  $\vec{V}_{ij}$  and +X axis is minimum so that  $P_{11}^*$  is set as vertex  $q$ .

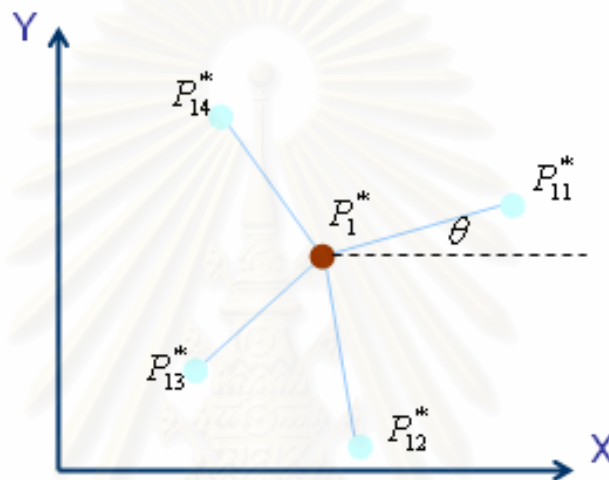


Figure 3-15. Tracking along +X axis

4. Increase the distance by the distance between the reference vertex and the vertex  $q$  received in the 3<sup>rd</sup> procedure.

$$S = S + |q - P_i^*| \quad (3-22)$$

5. Set vertex  $q$  as the next reference vertex.

$$P_i^* = q$$

6. Repeat the 3<sup>rd</sup> to the 5<sup>th</sup> procedure until the computed reference vertex out of the considered area.

$$Q(q) \neq C_i$$

7. Also check out, with the same method, for the rest directions +X, +Y and -Y. (Change +X in the 3<sup>rd</sup> procedure to -X, +Y and -Y)

8. Estimate a rectangular area compared to the acceptable area ( $A^*$ ). If the area is larger, then the repartition required.

$$A > A^*$$

Figure 3-16 shows a method to find the distance along +X axis by following the position along vertex that gives minimum angle to the +X axis until that vertex does not stay in the considering border, shown as point A in Figure 3-16. The procedures are stop and the summation of all distances, shows as bold lines, is then computed. A rectangular area can be estimated and compared to the acceptable area ( $A^*$ ) to see whether the repartition process is required.

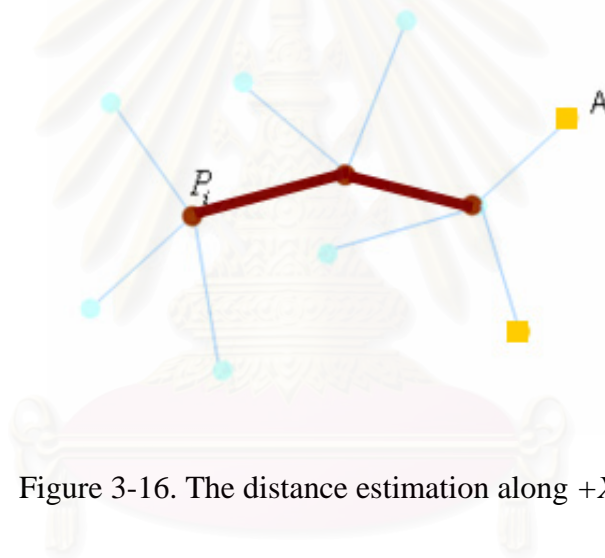


Figure 3-16. The distance estimation along +X axis

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

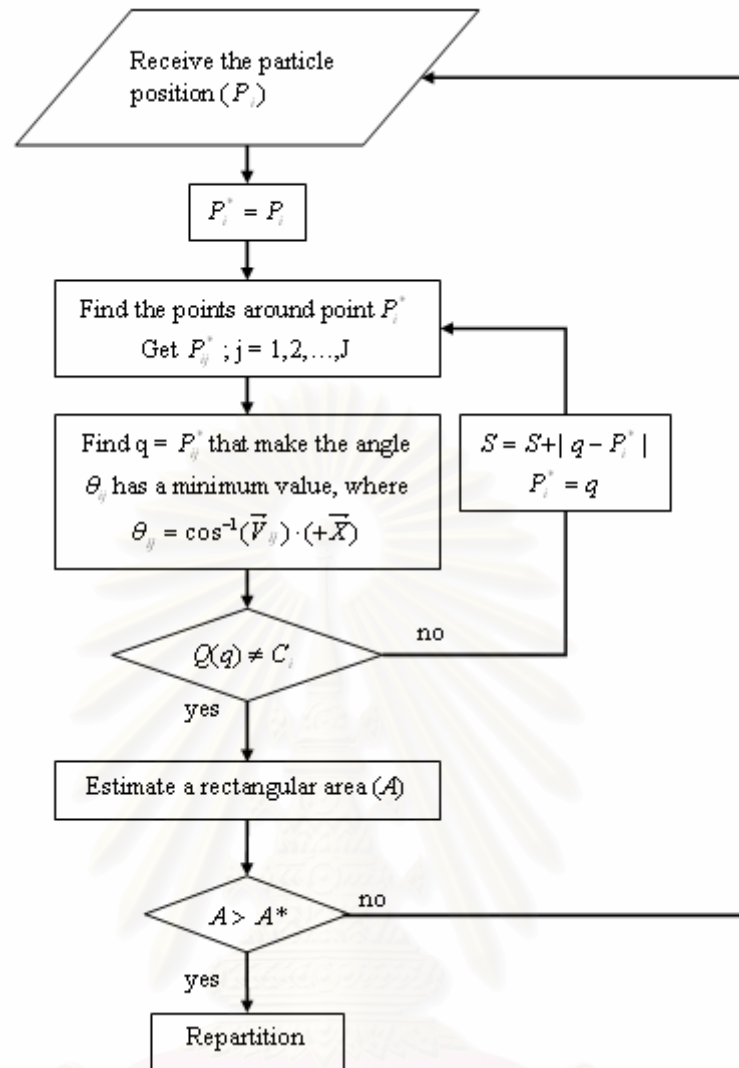


Figure 3-17. Flow chart shows the repartition checking procedures

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย



Comparison between the original particle-based collision detection algorithm [3] and the proposed algorithm are summarized as shown in Table 4-1.

Table 3-1. Comparison between original particle-based and the proposed algorithm

Original particle-based [3]	Proposed algorithm
Number of particle is arbitrary set.	Number of particles is reasonably chosen by applying the Kissing number.
Particles are randomly placed at the initialization.	Particles are suitably dispersed at the initialization.
Number of particles is fixed.	Number of particles can be adjusted to suit the deformed object.
Particles might not disperse well in some cases.	Particles always cover all over the object.

## CHAPTER 4

### ALGORITHM ANALYSIS

A collision detection algorithm has been clearly proposed as described in the previous chapter. However, some evidences are still required in order to be convinced that the proposed algorithm can efficiently detect the collision. This chapter therefore provides some substantiation as shown in section 4.1. Furthermore, it also gives an analysis of the time complexity which shows that the algorithm can be processed in polynomial time.

#### 4.1 Correctness

This section provides some proofs to verify the practicality of the proposed algorithm. The proofs show that particles are always moved to the collision points. Hence, the collision can certainly be detected as required.

Process can be classified into several cases depends on kind of objects and the number of objects in the system. Here, the process is considered as six cases.

- Two convex objects which have the same size.
- Several convex objects which have the same size.
- Several convex objects which have non-equivalent size.
- Two concave objects.
- Several concave objects.
- Both convex and concave objects.

Table 4-1. Six possible cases

Case	Number of objects	Kind of objects	Size
1	2	Convex	Equivalent
2	>2	Convex	Equivalent
3	>2	Convex	non-equivalent
4	2	Concave	-
5	>2	Concave	-
6	>2	convex, concave	-

Note: symbol “ - ” refers to not specify (Both equivalent and non-equivalent can be considered as the same case.)

The details of each case are summarized in Table 4-1. Each of these six cases is considered as follows.

#### 4.1.1 Case 1 – two convex objects which have the same size.

This is the simplest case since there are only two objects which also have the same size. In the proof, sphere objects are used to represent a convex object as shown below.

**Proof:** Given two equivalent spheres, object *A* and object *B*, touching each other at point *C* as shown in Figure 4-1. On each there is a particle which is supposed to be at point *a* for object *A*, and point *b* for object *B*. A collision can be detected by moving these particles into each other to the collision point *C*.

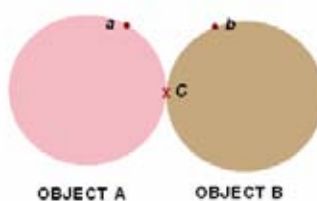


Figure 4-1. Two equivalent spheres touching at point *C*.

As described in section 3.4, Movement of particles, it can be concluded that each particle tends to move to the vertex which receives maximum force. This force can be computed by taking the force equation shown in section 3.3 which states that

$$F = \frac{1}{r^2}$$

When  $F$  is the force between particles

$r$  is the distance between particles

This equation implies that the force between particles ( $F$ ) becomes maximum when the distance between particles ( $r$ ) is a minimum value. As a result, particles are therefore moved to the vertices which the distance between these vertices is shortest. This can be concluded that particles are always moved to a collision point so that a collision between two convex objects which have the same size is absolutely detected as claimed. #

**4.1.2 Case 2 - several convex objects which have the same size.**

**Proof:** Given three touching spheres with the equivalent size, each diameter is equal to  $D$ . In this case, there are more than two objects in the system so that partitioning has to be applied. As these spheres have the same size, Kissing number, 12, is used as the number of particles.

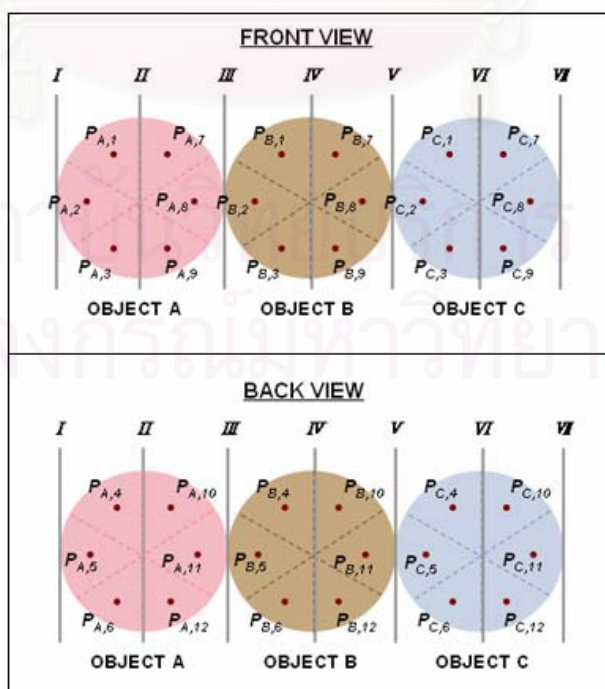


Figure 4-2. Three touching spheres.

Particles are classified into six sets as shown in Figure 4-3 which will be considered separately.

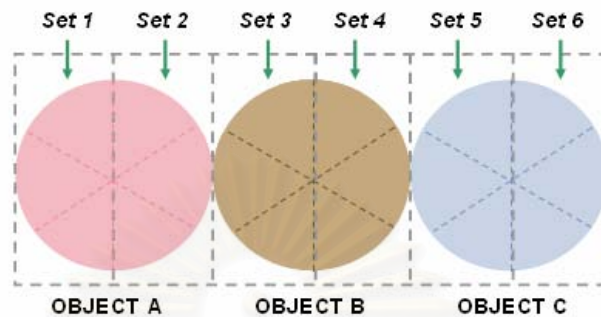


Figure 4-3. Six sets of particles

Set 1 - Particle between plane *I* and *II*:  $P_{A,1}, P_{A,2}, P_{A,3}, P_{A,4}, P_{A,5}, P_{A,6}$

Set 2 - Particle between plane *II* and *III*:  $P_{A,7}, P_{A,8}, P_{A,9}, P_{A,10}, P_{A,11}, P_{A,12}$

Set 3 - Particle between plane *III* and *IV*:  $P_{B,1}, P_{B,2}, P_{B,3}, P_{B,4}, P_{B,5}, P_{B,6}$

Set 4 - Particle between plane *IV* and *V*:  $P_{B,7}, P_{B,8}, P_{B,9}, P_{B,10}, P_{B,11}, P_{B,12}$

Set 5 - Particle between plane *V* and *VI*:  $P_{C,1}, P_{C,2}, P_{C,3}, P_{C,4}, P_{C,5}, P_{C,6}$

Set 6 - Particle between plane *VI* and *VII*:  $P_{C,7}, P_{C,8}, P_{C,9}, P_{C,10}, P_{C,11}, P_{C,12}$

Let the effective radius of attraction ( $R_{eff}$ ) is equal to  $D$ . The movement of each particle set can be considered as follow. Note that, as shown below, the movement of particles does not have to be considered in the right sequence since the movement process has to be repeated until all particles become stable.

#### Set 1

There are only particles between plane *II* and *IV*, particles in set 2 and 3, which can be in the area of  $R_{eff}$ . However, particles in set 2 are not considered since it is on the same object as particles in set 1. As a result, the movement of particles in set 1 can be computed by considering only particles in set 3. As shown in the figure, particles in set 3 can either be in the  $R_{eff}$  area or be out of the  $R_{eff}$  area.

Consequently, each particle in set 1 can either move to the +X direction or it can have no movement.

### Set 6

There are only particles between plane *IV* and *VI*, particles in set 4 and 5, which can be in the area of  $R_{eff}$ . However, particles in set 5 are not considered since it is on the same object as particles in set 6. As a result, the movement of particles in set 6 can be computed by considering only particles in set 4. As shown in the figure, particles in set 4 can either be in the  $R_{eff}$  area or be out of the  $R_{eff}$  area. Consequently, each particle in set 6 can either move to the -X direction or it can have no movement.

### Set 2

There are only particles between plane *I* and *V*, particles in set 1, 3 and 4, which can be in the area of  $R_{eff}$ . However, particles in set 1 are not considered since it is on the same object as particles in set 2. As a result, the movement of particles in set 2 can be computed by considering only particles in set 3 and 4. As shown in the figure, particles in set 3 and 4 are on the right hand side of particles in set 2. Therefore, particles in set 2 are moved to +X direction until the end of their area.

### Set 5

There are only particles between plane *III* and *VII*, particles in set 3, 4 and 6, which can be in the area of  $R_{eff}$ . However, particles in set 6 are not considered since it is on the same object as particles in set 5. As a result, the movement of particles in set 5 can be computed by considering only particles in set 3 and 4. As shown in the figure, particles in set 3 and 4 are on the left hand side of particles in set 5. Therefore, particles in set 5 are moved to -X direction until the end of their area.

### Set 3

There are only particles between plane *I* and *VI*, particles in set 1, 2, 4 and 5, which can be in the area of  $R_{eff}$ . However, particles in set 4 are not considered since it is on the same object as particles in set 3. As a result, the movement of particles in set 3 can be computed by considering only particles in set 1, 2 and 5. As shown in the



figure, there are 12 particles which pull particles in set 3 to the  $-X$  direction, while there are only 6 particles which pull this set of particles to the  $+X$  direction. In addition, it is closer to Set 2 than Set 5 so that Set 2 can take more effects on this set of particles. Therefore, particles in set 3 are moved to  $-X$  direction until the end of their area.

#### Set 4

There are only particles between plane *II* and *VII*, particles in set 2, 3, 5 and 6, which can be in the area of  $R_{eff}$ . However, particles in set 3 are not considered since it is on the same object as particles in set 4. As a result, the movement of particles in set 4 can be computed by considering only particles in set 2, 5 and 6. As shown in the figure, there are 12 particles which pull particles in set 3 to the  $+X$  direction, while there are only 6 particles which pull this set of particles to the  $-X$  direction. In addition, it is closer to Set 5 than Set 2 so that Set 5 can take more effects on this set of particles. Therefore, particles in set 4 are moved to  $+X$  direction until the end of their area.

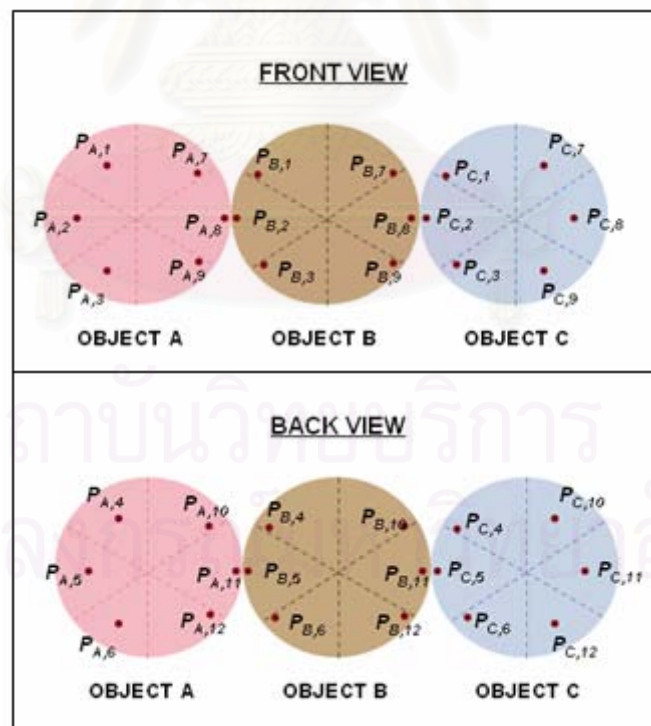


Figure 4-4. Three touching spheres after all movement of particles.

Finally, particle position can be updated as shown in Figure 4-4. Particles are moved to collision points so that the collision at point  $C_1$  and  $C_2$  can be detected. Therefore, it can be concluded that collision between several convex objects which have the same size can be detected as claimed. #

**4.1.3 Case 3 - several convex objects which have non-equivalent size.**

**Proof:** Given three touching spheres which its diameter equal to  $D$ ,  $\frac{D}{2}$  and  $D$  respectively.

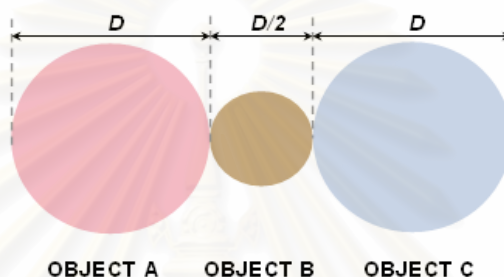


Figure 4-5. Three touching spheres.

Suppose that partitioning was applied and particles are suitably dispersed according to the algorithm shown in section 3.1 and 3.2. Particles are classified into six sets as shown in Figure 4-6 which will be considered separately.

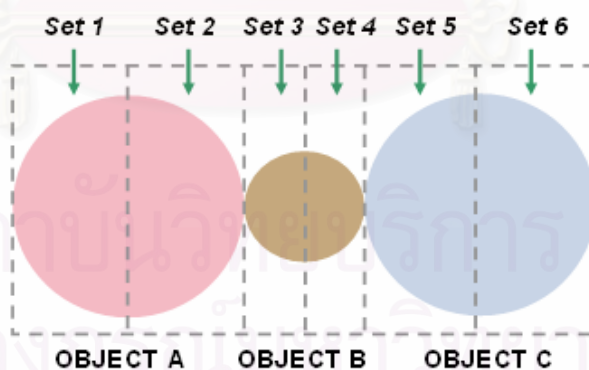


Figure 4-6. Six sets of particles

Let the effective radius of attraction ( $R_{eff}$ ) equal to  $\frac{D}{2}$ . The movement of each particle set can be considered as follow.

Set 1

There are only particles in set 2 which can be in the area of  $R_{eff}$ . However, these particles are on the same object as particles in set 1 so that this will not take any effect on particles in set 1. As a result, particles in this set therefore have no movement.

### Set 6

There are only particles in set 5 which can be in the area of  $R_{eff}$ . However, these particles are on the same object as particles in set 6 so that this will not take any effect on particles in set 6. As a result, particles in this set therefore have no movement.

### Set 2

There are only particles in set 1, 3 and 4 which can be in the area of  $R_{eff}$ . However, particles in set 1 are not considered since it is on the same object as particles in set 2. As a result, the movement of particles in set 2 can be computed by considering only particles in set 3 and 4. As shown in the figure, both particles in set 3 and 4 are on the right hand side of particles in set 2. Therefore, particles in set 2 are moved to +X direction until the end of their area.

### Set 5

There are only particles in set 3, 4 and 6 which can be in the area of  $R_{eff}$ . However, particles in set 6 are not considered since it is on the same object as particles in set 5. As a result, the movement of particles in set 5 can be computed by considering only particles in set 3 and 4. As shown in the figure, both particles in set 3 and 4 are on the left hand side of particles in set 5. Therefore, particles in set 5 are moved to -X direction until the end of their area.

At this state, particle position can be shown in Figure 4-7.

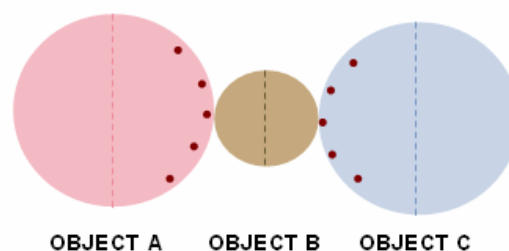


Figure 4-7. Updated figure of three touching non-equivalent spheres.

Set 3

There are only particles in set 2, 4 and 5 which can be in the area of  $R_{eff}$ . However, particles in set 4 are not considered since it is on the same object as particles in set 3. As a result, the movement of particles in set 3 can be computed by considering only particles in set 2 and 5.

According to the force equation shown in section 3.3, the force acting on each particle can be considered as two directions, force which pull particle to +X direction and -X direction.

$$F_{+x} = \sum \frac{1}{(\overline{xy})^2} \quad \text{and} \quad F_{-x} = \sum \frac{1}{(\overline{xz})^2} \quad (4-1)$$

when  $x$  is coordinate of particle in set 3,  $y$  is coordinate of particle in set 5, and  $z$  is coordinate of particle in set 2.

$$\begin{aligned} F_{+x} &= \sum_{j=1}^6 \frac{1}{(\overline{P_{B,i}P_{C,j}})^2} \\ &= \frac{1}{(\overline{P_{B,i}P_{C,1}})^2} + \frac{1}{(\overline{P_{B,i}P_{C,2}})^2} + \frac{1}{(\overline{P_{B,i}P_{C,3}})^2} + \frac{1}{(\overline{P_{B,i}P_{C,4}})^2} + \frac{1}{(\overline{P_{B,i}P_{C,5}})^2} + \frac{1}{(\overline{P_{B,i}P_{C,6}})^2} \end{aligned} \quad (4-2)$$

$$\begin{aligned} F_{-x} &= \sum_{k=7}^{12} \frac{1}{(\overline{P_{B,i}P_{A,k}})^2} \\ &= \frac{1}{(\overline{P_{B,i}P_{A,7}})^2} + \frac{1}{(\overline{P_{B,i}P_{A,8}})^2} + \frac{1}{(\overline{P_{B,i}P_{A,9}})^2} + \frac{1}{(\overline{P_{B,i}P_{A,10}})^2} + \frac{1}{(\overline{P_{B,i}P_{A,11}})^2} + \frac{1}{(\overline{P_{B,i}P_{A,12}})^2} \end{aligned} \quad (4-3)$$

As shown in the figure, particles in set 3 are closer to set 2 than set 5 so that particles in set 2 can take more effects as described below.

As  $\overline{P_{B,i}P_{A,7}} < \overline{P_{B,i}P_{C,1}}$ ,  $\overline{P_{B,i}P_{A,8}} < \overline{P_{B,i}P_{C,2}}$ ,  $\overline{P_{B,i}P_{A,9}} < \overline{P_{B,i}P_{C,3}}$ ,  $\overline{P_{B,i}P_{A,10}} < \overline{P_{B,i}P_{C,4}}$ ,  $\overline{P_{B,i}P_{A,11}} < \overline{P_{B,i}P_{C,5}}$  and  $\overline{P_{B,i}P_{A,12}} < \overline{P_{B,i}P_{C,6}}$ , all terms of Equation 4-3 is therefore greater than in Equation 4-2 so that

$$F_{-X} > F_{+X}$$

Therefore, particles in set 3 tend to move to  $-X$  direction. These particles are moved away from other particles on the right hand side. As a result, particles in set 3 are moved continuously to  $-X$  direction until the end of their area without passing any stable position.

#### Set 4

There are only particles in set 2, 3 and 5 which can be in the area of  $R_{eff}$ . However, particles in set 3 are not considered since it is on the same object as particles in set 4. As a result, the movement of particles in set 4 can be computed by considering only particles in set 2 and 5. As shown in the figure, particles in set 4 are closer to set 5 than set 2 so that particles in set 5 can take more effects according to the proof shown for set 3. Therefore, particles in set 4 tend to move to  $-X$  direction. These particles are moved away from other particles on the left hand side. As a result, particles in set 4 are moved continuously to  $+X$  direction until the end of their area without passing any stable position.

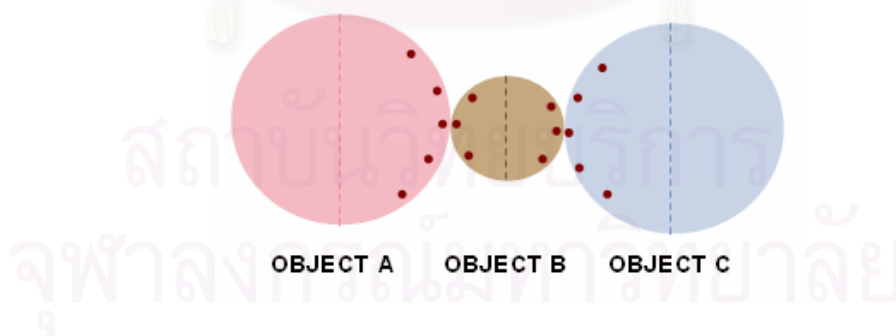


Figure 4-8. Three touching spheres after all movement of particles.

After all movement of particles is computed, particle position is finally updated as shown in Figure 4-8. Particles are moved to collision points so that the collisions at point  $C_1$  and  $C_2$  can be detected. Therefore, it can be concluded that

collisions between several convex objects which have non-equivalent sizes can be detected as claimed. #

#### 4.1.4 Case 4 - two concave objects.

This case can be easily discussed since each concave object can be approximated as a group of convex objects. In the proof, *U-shape* object is used to represent a concave object as shown below.

**Proof:** Given two touching concave objects, object *A* and object *B*, as shown in Figure 4-9.

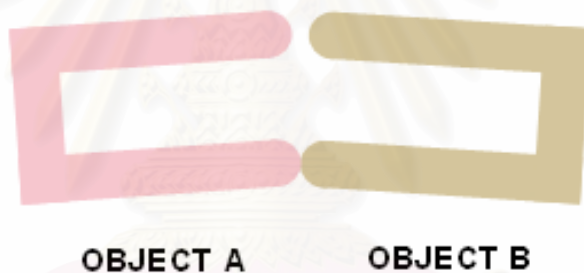


Figure 4-9. Two touching concave objects.

As described in section 3.1 and 3.2, each concave object has to be considered as a combination of convex objects. For the given case, each concave object is divided into 3 convex objects as shown in figure 4-10.

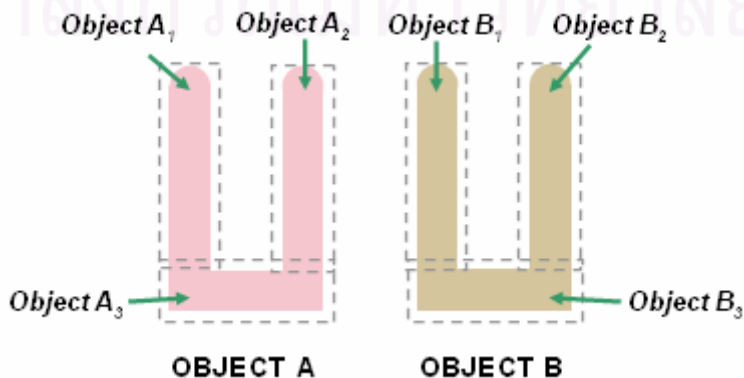




Figure 4-10. Each concave is considered as a combination of three convex parts.

This seems like there are 6 convex objects in the system. The algorithm is processed by checking a collision between each pair of convex object. This can be performed as discussed in case 3. Therefore, it can be concluded that collision between two concave objects can be detected as claimed. #

#### *4.1.5 Case 5 - several concave objects.*

**Proof:** As described in case 4, each concave object is considered as a combination of convex parts. A collision is then checked between each pair of convex objects which can be performed as discussed in Section 4.1.3. Therefore, it can be concluded that collision between several concave objects can be detected as claimed. #

#### *4.1.6 Case 6 - both convex and concave objects.*

**Proof:** As described in case 4, the concave object is considered as a combination of convex parts. A collision is then checked between each pair of convex objects which can be performed as discussed in Section 4.1.3. Therefore, it can be concluded that collision between convex and concave objects can be detected as claimed. #

All the proofs above show that, on each collided object, there certainly be at least a particle which is attracted to a collision point. Consequently, there always be a pair of particles at the vertices which closest to the collision point. This can ensure that the proposed algorithm will not miss any collision and therefore can efficiently detect the collision.

## 4.2 Complexity analysis

Complexity of the proposed algorithm corresponds to the computational time of 3 processes which are the partitioning process, the collision detection process and the repartition checking process. First, let  $M$  be the number of vertices, and  $N$  be the number of particles or the number of partitions. The algorithm can be analyzed as follow.

### 4.2.1 Surface partitioning

The number of required particles for each object can be estimated by checking every vertex for critical points, so that the computation takes time  $O(M)$  where  $M$  is number of vertices.

The surface partitioning can be performed by applying LBG vector quantization. Each iteration process is computed by considering  $M$  vertices which take time  $O(M)$ . Given the iteration steps equal to  $k$ , so that the whole iteration process has the complexity of  $O(kM)$ . Consequently, for  $N$  partitions, the complexity is  $O(kMN)$ . However, in the worst case, the number of particles ( $N$ ) can become equal to the number of vertices. Therefore, the complexity for the worst case is  $\Theta(kM^2)$ . Assume that the iteration step ( $k$ ) is a constant value, so that the computational time is therefore equal to  $O(M^2)$ .

However, the partitioning process does not have to be computed every time object deforms as it is the preprocessing procedure. Therefore, the computational time for this process can be ignored and the complexity of the proposed algorithm can be estimated from other two following processes.

### 4.2.2 Collision detection

This process consists of a movement process and a collision checking process. The computational time for each process can be described as follow.

In the movement process, each particle can continuously move along vertices until its destination vertex is outside the boundary. The number of steps which each particle can move is approximated by considering the area of each partition. Assume that vertices are equally distributed so that the number of vertices in each partition

is  $\frac{M}{N}$ . Hence, the movement of each particle can be computed with a complexity of  $O\left(\frac{M}{N}\right)$ . This process is performed for  $N$  particles so that the complexity of overall movement process is  $O(M)$ .

For the collision checking, the distance between each pair of particles is determined on the verge of collision. This can be performed with a complexity of  $O(N^2)$ .

Generally, the number of required particles ( $N$ ) is much less than the number of vertices ( $M$ ) as show in the equation.

$$N \ll M \quad (5-1)$$

However, it can becomes the worst case when the object has very complex geometry which its number of particles ( $N$ ) is nearly to the number of vertices ( $M$ ). Therefore, in the worst case, the collision is checked with  $\Theta(M^2)$ .

#### 4.2.3 Repartition checking

The area of each partition is investigated in order to determine whether a repartition should be performed. The area can be computed by searching along the vertices to the edge around. Approximately, the number of vertices in each partition is  $\frac{M}{N}$  so that the computational time for estimating each area is  $O\left(\frac{M}{N}\right)$ . Therefore,  $N$  area can be estimated with a complexity of  $O(M)$ .

As mention above, the complexity of the proposed algorithm can be computed by considering the partitioning and the detection process. The summation time for overall process can be analyzed as shown below.

$$\begin{aligned} T &= \text{collision detection} + \text{repartition checking} \\ &= (\text{movement of particle} + \text{collision checking}) + \text{repartition checking} \\ T_{\text{worst}} &= (O(M) + O(M^2)) + O(M) \\ &= O(M^2) \end{aligned}$$

In this algorithm, the process becomes worst case when using complex object which has the number of particle nearly to the number of vertices. However, most objects used in computer graphic is not as much complex as mention. The number of particles is typically much less than the number of vertices. Therefore, it should be concluded that the complexity of this algorithm is  $O(N^2)$  when  $N$  is the number of particles.



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## CHAPTER 5

### DISCUSSION CONCLUSION AND FUTURE WORK

The algorithm still needs improvement in some points. The discussion of this research, conclusion, and future work are summarized in the sections that follow.

#### 5.1 Discussion

This section provides a discussion of the proposed algorithm. Several related points will be thoroughly discussed which can be analyzed as follow.

##### *5.1.1 Problem of the complex topology*

As described in section 3.1, the number of particles can be computed depends on size of the smallest part in the scene. If the smallest part is very small compared to others as shown in Figure 5-1, the computed number of particles might be exceeded which can slow down the process. The figure shows some particles are unnecessarily to detect a collision.

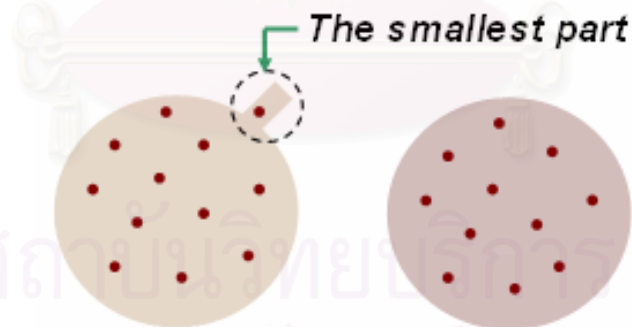


Figure 5-1. Particles are unnecessarily generated due to a special case of topology.

Moreover, when an object has very complex topology such as an object which has several branches shown in Figure 5-2, the object is therefore approximated into several parts. Hence, a great number of particles are added to this process which can cost too much computational time. It can be concluded that the proposed algorithm might have less efficiency when using with an object having complex topology.

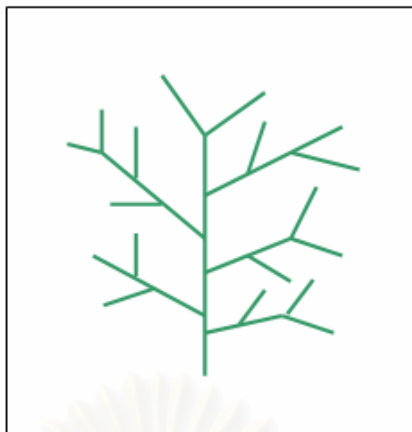


Figure 5-2. An object which has complex topology.

### ***5.1.2 Problem of partitioning symmetry object centered at the origin***

LBG quantization is a classification technique which has not been proposed to be applied with a surface partitioning. As the vertex classification due to this technique does not consider the connectivity between vertices, the computed partition might not suitably represent object surface. However, in the proposed algorithm, this problem can be eliminated by dividing each concave part into several convex parts so that proper partitions are consequently achieved.

Besides, LBG quantization can not partition a symmetry object centered at the origin as shown in Figure 5-3 a. According to the LBG theory described in Section 2.1.3, an average of all 8 vertices in the figure is  $(0,0,0)$ . This average value has to be adjusted in order to split vertex data into 2 groups. Two code vectors for two groups can be found;  $(1 + \varepsilon)(0,0,0)$  and  $(1 - \varepsilon)(0,0,0)$ , which both are  $(0,0,0)$  so that any data can not be split. Therefore, in order to efficiently perform a surface partitioning process, the input symmetry object's center should not be placed at the origin. As shown in Figure 5-3 b, the same cube which its center stays away from the origin. The average can be found which can split input vertices into two groups as required.



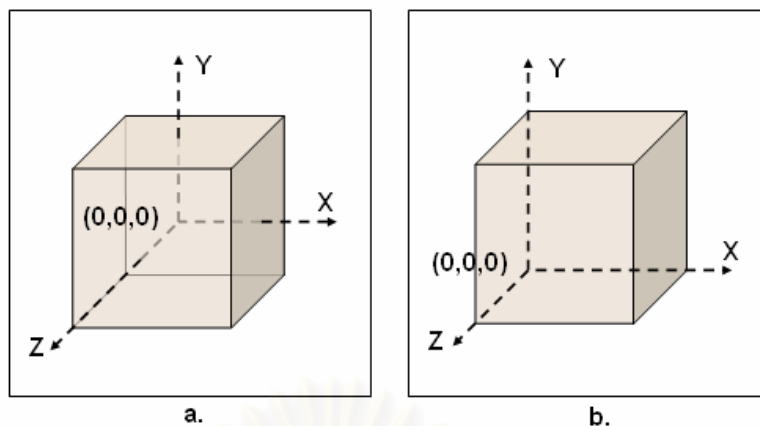


Figure 5-3. (a) a cube centered at the origin, (b) a cube centered away from the origin.

### 5.1.3 Problem of the improper effective radius of attraction ( $R_{eff}$ )

According to the interaction force equation shown in Section 3.3, the existence of force depends on the effective radius of attraction ( $R_{eff}$ ). If it is set to a small value, the force might take effect lately. However, this does not affect the process much since particles can move continuously. A collision still can be efficiently detected even when the  $R_{eff}$  is set to a small value. Nevertheless, an error can be occurred if  $R_{eff}$  value is too small. When an object comes too close and cause a collision, the force are still not calculated due to a very small value of  $R_{eff}$  so that a collision can not be detected. As shown in Figure 5-4, a distance between particle A and B is longer than a specified  $R_{eff}$  so that forces can not be calculated and particles can not move to detect the collision.

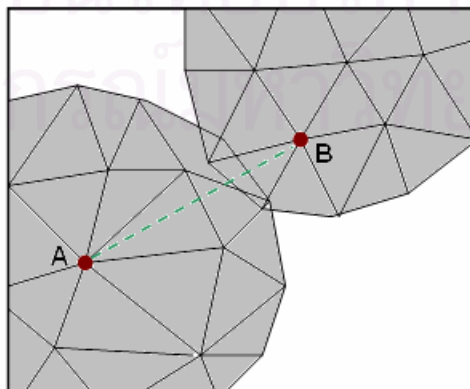


Figure 5-4. Undetected collision due to a very small value of  $R_{eff}$ .

In order to eliminate this problem, the effective radius of attraction ( $R_{eff}$ ) is set not smaller than the collision distance value ( $\mu$ ).

**Proof:** Given two touching sphere objects, object A and object B, as shown in Figure 5-5.

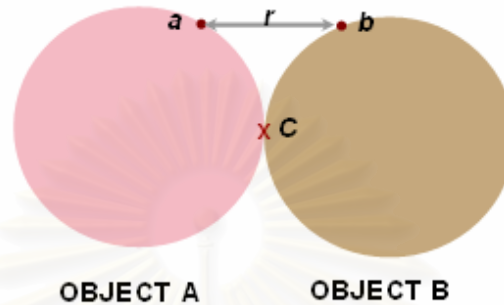


Figure 5-5. Two touching sphere objects.

Suppose the particle of object A is at point  $a$  and the particle of object B is at point  $b$ . Let the distance between these particles ( $r$ ) is shorter than a collision distance value ( $\mu$ ) which can be expressed as follow.

$$\text{Given: } r = \mu + \varepsilon \quad (5-1)$$

$$\text{When } \varepsilon \text{ is a small positive value: } \varepsilon > 0 \quad (5-2)$$

According to a regulation in the proposed algorithm, a collision can be detected only if the distance between a pair of particle ( $r$ ) is shorter than a collision distance value ( $\mu$ ) as shown in the following equation.

$$\text{Collision is detected when: } r < \mu \quad (5-3)$$

In the given case,  $r$  can be replaced by  $\mu + \varepsilon$  according to Equation 5-1

$$\mu + \varepsilon < \mu$$

$$\varepsilon < 0 \quad (5-4)$$

The contrast between Equation 5-2 and 5-4 shows that particles at point  $a$  and  $b$  can not detect the collision since  $r$  is longer than  $\mu$ .

In order to correctly report the collision, particles have to be moved closer to the collision point,  $c$ , which can reduce the  $r$  value to be smaller than  $\mu$ . However, the

movement can be occurred only if  $r$  is shorter than the effective radius of attraction ( $R_{eff}$ ) as show in this equation.

$$\text{Movement is occurred when: } r < R_{eff} \quad (5-5)$$

For this case,  $r$  can be replaced by  $\mu + \varepsilon$  as shown in Equation 5-1 which implies,

$$\begin{aligned} \mu + \varepsilon &< R_{eff} \\ \therefore R_{eff} &> \mu \end{aligned} \quad (5-6)$$

This can be concluded that the effective radius of attraction ( $R_{eff}$ ) should be set not smaller than the collision distance value ( $\mu$ ). #

However, using too long  $R_{eff}$  is also not a good answer. Forces have to be computed even when objects are very far away from each other which can unnecessarily cost much computational time.

#### ***5.1.4 Problem of the improper collision distance value ( $\mu$ )***

When there is a collision between objects, particles on its corresponding object do not have to be collided since particles can not stay on edges. Therefore, a collision distance value ( $\mu$ ) is set in order to determine a distance between particles on the verge of collision. If it is too small, a collision might not be detected.

The collision distance ( $\mu$ ) should not be set smaller than half of the average edge length ( $\bar{d}$ ). The proof can be discussed as follow.

**Proof:** Given two touching objects, object *A* and object *B*, as shown in Figure 5-6.

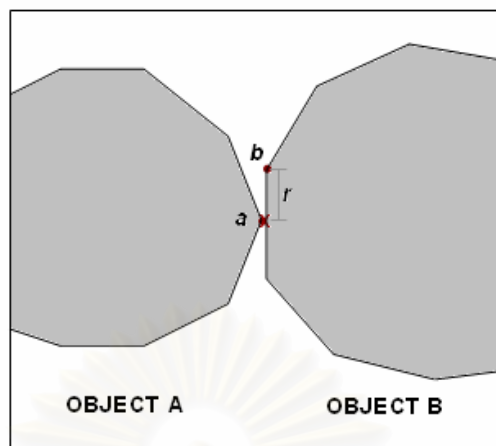


Figure 5-6. A collision on object edge.

Suppose a vertex of object *A* is collided to the edge of object *B* at point *a*. Particle of object *A* is at point *a*, and particle of object *B* is at point *b* which is the vertex closest to collision point *a*.

Assume that every edge has the same length which equal to  $\bar{d}$ . The distance between these particles (*r*) is therefore not more than half of the edge length.

$$r \leq \frac{\bar{d}}{2} \quad (5-7)$$

As mention before, a collision can be detected when the distance between particles (*r*) is shorter than a collision distance value ( $\mu$ ).

Collision is detected when:  $r < \mu$

As shown in Equation 5-7, *r* can be replaced by  $\frac{\bar{d}}{2}$  which gives

$$\mu > \frac{\bar{d}}{2} \quad (5-8)$$

Therefore, a collision can be detected when setting the collision distance ( $\mu$ ) not smaller than half of the edge length. #

However, if it is set to a very high value, the process might incorrectly report a collision. As shown in Figure 5-7, a collision is reported even though there is no collision.

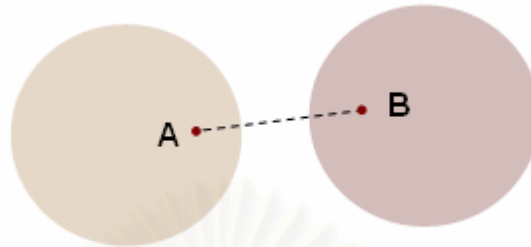


Figure 5-7. An incorrectly report of collision due to a high value of collision distance.

#### ***5.1.5 Problem of the improper acceptable area ( $A^*$ )***

In the repartition checking process, the acceptable area is set in order to determine whether a repartition is required. A decision on repartitioning is directly depended on the value of acceptable area. If the acceptable area is set to a very high value, a repartition might not occur even when some areas become too large. Consequently, a particle is not enough to cope with the extensive area. However, if the acceptable area is very small, the process might perform a repartition almost every time the deformation occurred. This can cost a very high computational time which is not desired for the process.

#### ***5.1.6 Problem of object deformation***

A repartition is determined by considering object deformation. If the deformation affects the correctness of detection, then a repartitioning process should be applied in order to be sure that object is always suitably partitioned. The proposed algorithm considers only on the area of deformed partition. A repartition is required when there is at least one area larger than a specified value which only one particle is not enough to detect many collisions at the same time. However, this algorithm might have less efficiency when there is an object part changes its topology into concave as shown in Figure 5-8. If there are more than one object collide that part at the same time, a collision can not be detected due to the insufficient of particle.

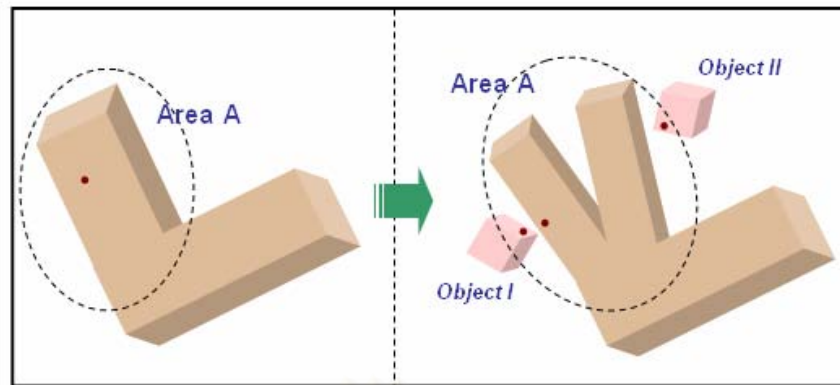


Figure 5-8. Deformation case that causes the particle deficiency to detect the collision.

Moreover, a collision might not be detected when the object deforms until few edge become longer than a collision distance value ( $\mu$ ). As shown in Figure 5-9, particle A can not move to detect a collision since it can not stay on the extensive edge.

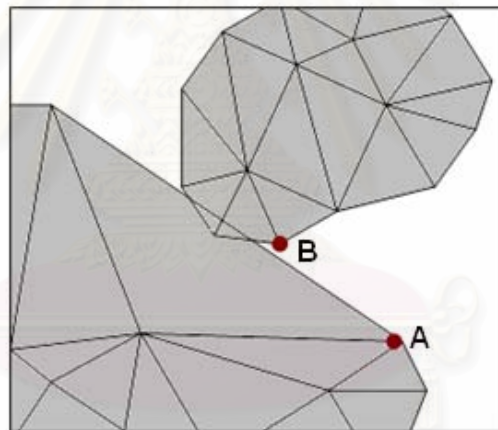


Figure 5-9. Undetected collision due to a very long edge.

In addition, deformation by changing the number of vertices is also not considered in this algorithm. Increasing number of vertices can cause some area to lack of particles. As the increased vertices have not been classified into any partition, particles therefore can not move to these vertices. Hence, detection might fail when there is a collision at one of these increased vertices. Also, decreasing number of vertices can cause a problem. If there is a particle positions on the removed vertex, this particle will also be eliminated. Therefore, there is no longer being any particle to detect collision in this corresponding area.



### 5.1.7 Complexity discussion

As described in Section 4.2, the complexity to perform a detection process in this algorithm is  $O(N^2)$  when  $N$  is the number of particles, while the complexity of a brute force collection detection is  $O(M^2)$  when  $M$  is the number of vertices. The required number of particles rather depends on the number of vertices as shown in the following equation.

$$N = \gamma M \quad (5-9)$$

When  $\gamma$  is a relation variable can vary from 0 to 1 depends on size and topology of the object. It is obvious that, the efficiency of this algorithm is reverse to the value of  $\gamma$ . In the case which  $\gamma$  is nearly 0, the required number of particles in this algorithm is therefore much lower than the number of vertices. Also, this can cost much lower computational time. However, this  $\gamma$  value can be nearly to 1 if its topology is extremely complex as described in Section 5.1.1. A great number of particles are added into the process which can cause very high computational time. Therefore, it can be concluded that the proposed algorithm might not appropriate for very complex structure.

## 5.2 Conclusion

This research presents a collision detection algorithm for deformable objects using particles as collision checkers. The proposed algorithm provides a particle separately to each corresponding area by applying LBG quantization to partition object surface. Attractive force is assigned between particles on different object in order to pull them into each other when there tend to be a collision. A distance between each pair of particle is investigated and compared with a tolerable value on the verge of collision. After the deformation of objects, the acceptance area is monitored so that the proper number of particles can be altered by completing the repartitioning process.

The algorithm analysis gave in the previous chapter shows that this particle-based collision detection algorithm can efficiently detect a collision with the complexity of  $O(N^2)$ , when  $N$  is the number of particle.

## 5.3 Future Work

The proposed collision detection algorithm is just a preliminary step and still need development especially in the repartition checking process. The repartition checking of this algorithm is merely based on the size of deformed area while other factors can also affect the appropriated partition as described in Section 5.1.6. In order to handle topological deformation, the considering of topological transformation should be applied. Also, the number of vertices and edge length should be investigated so that an appropriate partition can be achieved at all time.

Moreover, applying the splitting method to find initial code vector for the partitioning process can cause the number of partitions restricted to  $2^n$  value, where  $n$  is a counting number. Therefore, the partitioning process can be developed by trying another method to initiates code vector which can gives more adaptive number of partitions.

## REFERENCES

- [1] M. C. Lin and S. Gottschalk; Collision detection between geometric models: a survey, Proceedings of IMAConference, Mathematics of Surfaces VIII, 1998.
- [2] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, P. Volino; Collision Detection for Deformable Objects, *EUROGRAPHICS* 2004.
- [3] M. Senin , N. Kojekine , V. Savchenko and I. Hagiwara; Particle-based Collision Detection, *EUROGRAPHICS* 2003. Short Presentations.
- [4] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Boston, MA: Kluwer, 1991.
- [5] H. Abut, R.M. Gray and G. Rebolledo, "Vector Quantization of Speech and Speech-Like Waveforms," *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. ASSP-30, pp. 423-435, June 1982.
- [6] M. Qasem; Vector Quantization [[www.geocities.com/mohamedqasem](http://www.geocities.com/mohamedqasem)]
- [7] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design", *IEEE Trans. Communications.*, vol. 28, pp. 84-95, Jan. 1980.
- [8] A. Martin; Particle System
- [9] R. Parent, Computer Animation: Algorithms and Techniques, *Morgan Kaufmann*, 2001.
- [10] P. Hubbard; Approximating polyhedra with spheres for time-critical collision detection, *ACM Transactions on Graphics (TOG)*, 15(3):179–210, 1996.
- [11] I. Palmer and R. Grimsdale; Collision detection for animation using sphere-trees, *Computer Graphics Forum*, 14(2):105–116, 1995.
- [12] S. Quinlan; Efficient distance computation between non-convex objects, Proceedings of IEEE International Conference on Robotics and Automation '94), 3324–3329, 1994.

- [13] N. Bobic; Advanced Collision Detection Techniques by, 2000 [www.gamasutra.com]
- [14] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi; I-COLLIDE: an interactive and exact collision detection system for large-scale environments, Proceedings of the Symposium on Interactive 3D Graphics, 189–196, 1995.
- [15] M. Held, J. T. Klosowski, and J. S. B. Mitchell; Evaluation of collision detection methods for virtual reality fly-throughs, Proceedings Seventh Canadian Conference on Computational Geometry, 205–210, 1995.
- [16] J. Lander; When Two Hearts Collide: Axis-Aligned Bounding Boxes ,2000 [www.gamasutra.com]
- [17] S. Gottschalk, M. C. Lin, and D. Manocha; OOBTree: A hierarchical structure for rapid interference detection, ACM Computer Graphics (Proc. of SIGGRAPH'96), 171–180, 1996.
- [18] J. H. Conway and N. J. A. Sloane; "The Kissing Number Problem" and "Bounds on Kissing Numbers." §1.2 and Ch. 13 in *Sphere Packings, Lattices, and Groups, 2nd ed.* New York: Springer-Verlag, pp. 21-24 and 337-339, 1993.
- [19] J. Shen and D. Yoon: A New Scheme for Efficient and Direct Shape Optimization of Complex Structures represented by polygonal meshes, International Journal for Numerical Methods in Engineering, Vol 58, No. 4. pp. 2201-2223, 2003.

## BIOGRAPHY

**Name-Last Name:** Miss Nida Saenghaengtham

**Birth:** June 21, 1982, Bangkok, Thailand.

**Education:** B. Eng., Chemical engineering, Chulalongkorn University (2003)

**Publications:**

1. Nida Saenghaengtham and Pizzanu Kanongchaiyos. 2004. Collision Detection Algorithm for Deformable Objects using Particle. The 1<sup>st</sup> Thailand Computer Science Conference (ThCSC 2004), December, Kasetsart University, Bangkok, Thailand.
2. Nida Saenghaengtham and Pizzanu Kanongchaiyos. 2006. A Collision Detection Algorithm Using Particle Sensor. 2006 IEEE International Conference on Robotics, Automation & Mechatronics (RAM2006), June, Bangkok, Thailand.
3. Nida Saenghaengtham and Pizzanu Kanongchaiyos. 2006. Using LBG Quantization for Particle-based Collision Detection Algorithm. Journal of Zhejiang University SCIENCE, June, Zhejiang, China.

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย