

## INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600

UMI<sup>®</sup>



# **Metaball Graphics Package**

Liang Du

A major report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements  
for the degree of Master of Computer Science at  
Concordia University  
Montreal, Quebec, Canada

April 2003

© Liang Du, 2003



**National Library  
of Canada**

**Acquisitions and  
Bibliographic Services**

**395 Wellington Street  
Ottawa ON K1A 0N4  
Canada**

**Bibliothèque nationale  
du Canada**

**Acquisitions et  
services bibliographiques**

**395, rue Wellington  
Ottawa ON K1A 0N4  
Canada**

*Your file Votre référence*

*Our file Notre référence*

**The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.**

**The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.**

**L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.**

**L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.**

0-612-77986-6

**Canada**

# **ABSTRACT**

## **Metaball Graphics Package**

**Liang Du**

The goal of this project is to design a metaball graphics package for use in visualization of large object oriented software. In this technique, a metaball is used to represent a software entity and the relationship between these software entities. Appearance attributes such as size, color and texture are used to denote properties of the software entity. The software has been designed as a Java package with a well defined API to create metaballs of different radius, color and texture. This metaball API package can be seen as an extension of the Java3D API. The implementation is based on Java language with J2SDK1.4 and JAVA 3D 1.3. Hence the metaball API package can be used on any popular platform.

This report first introduces the scope of the project and then covers state of the art in software visualization, followed by a detailed description of the metaball technique. Then an overview of Java3D is provided, followed by design and implementation detail of the metaball package. Finally application to software visualization is discussed briefly and conclusion. The source code of the metaball API implementation is also enclosed in an electronic format.

## **ACKNOWLEDGEMENTS**

I am indebted to my supervisor, Professor Sudhir P. Mudur, for his guidance with patience and valuable time throughout this project.

I am grateful to Professor Juergen Rilling, who provided me the computer environment and also gave me valuable suggestions.

# TABLE OF CONTENTS

<b>LIST OF FIGURES .....</b>	<b>VII</b>
<b>CHAPTER 1: INTRODUCTION.....</b>	<b>1</b>
1.1. SOFTWARE VISUALIZATION .....	1
1.2. 3D GRAPHICS FOR SOFTWARE VISUALIZATION .....	1
1.3. METABALL TECHNOLOGY.....	2
1.4. METABALL API PACKAGE .....	2
1.5. ORGANIZATION OF THIS REPORT.....	3
<b>CHAPTER 2: A BRIEF SURVEY OF SOFTWARE VISUALIZATION TECHNIQUES.....</b>	<b>4</b>
2.1. HISTORY OF SOFTWARE VISUALIZATION .....	5
2.2. SOFTWARE VISUALIZATION SYSTEMS REVIEW .....	6
2.2.1. <i>Sorting Out Sorting</i> .....	6
2.2.2. <i>Polka-3D</i> .....	6
2.2.3. <i>ANIM</i> .....	7
2.2.4. <i>TPM</i> .....	7
2.2.5. <i>Pavane</i> .....	8
<b>CHAPTER 3: METABALL TECHNOLOGY.....</b>	<b>9</b>
3.1. WHAT IS METABALL.....	9
3.2. MARCHING CUBES ALGORITHM.....	11
3.2.1. <i>Principle of Marching Cubes Algorithm</i> .....	11
3.2.2. <i>Basic principle used to generate isosurface facets</i> .....	12
<b>CHAPTER 4 OVERVIEW OF JAVA3D .....</b>	<b>15</b>
4.1. JAVA3D GOAL.....	15
4.1.1. <i>Simplify 3D graphics application development</i> .....	15
4.1.2. <i>Make it ideal for intranet and internet visualization applications</i> .....	15
4.1.3. <i>Performance</i> .....	16
4.2. FEATURES OF JAVA 3D.....	16
4.2.1. <i>High-level scene-graph model:</i> .....	16
4.2.2. <i>Run-time loaders:</i> .....	16
4.2.3. <i>Geometry compression</i> .....	16
4.2.4. <i>Takes advantage of existing hardware accelerators</i> .....	16
4.2.5. <i>Flexible Viewing Model</i> .....	17
4.2.6. <i>Level of Detail(LOD)</i> .....	17
4.2.7. <i>Support for continuous action devices:</i> .....	17
4.3 SCENE GRAPH STRUCTURE ( SHOWN IN FIGURE 4.1).....	17
4.3.1. <i>VirtualUniverse Object</i> .....	17
4.3.2. <i>Locale Object</i> .....	17

4.3.3. <i>Node Object</i> .....	18
4.3.4. <i>Behavior Objects</i> .....	18
4.4. CONVENIENT FACILITIES PROVIDED BY JAVA 3D .....	19
4.4.1. <i>Simple KeyNavigator Behavior</i> .....	19
4.4.2. <i>Interpolators and Alpha Object</i> .....	20
4.4.3. <i>Easy expanding and across platform</i> .....	20
<b>CHAPTER 5 DESIGN AND IMPLEMENTATION OF METABALL PACKAGE</b>	<b>22</b>
5.1. UNDERSTANDING THE REQUIREMENTS .....	23
5.2. METABALL PACKAGE CLASSES.....	23
5.3. CLASS ARCHITECTURES AND IMPLEMENTATIONS .....	24
5.3.1. <i>Pt3d Class</i> :.....	24
5.3.2. <i>TRIANGLE Class</i> :.....	25
5.3.3. <i>GRIDCELL Class</i> : .....	26
5.3.4. <i>MarchingCube Class</i> : .....	27
5.4. ALGORITHMS USED IN METABALL API PACKAGE IMPLEMENTATION .....	30
5.4.1. <i>Marchingcube class design with recursion</i> .....	30
5.4.2. <i>Normal vector calculation of metaball surface used in metaball shading</i> .....	33
5.4.3. <i>Texture mapping of metaball surface</i> .....	34
5.4.4. <i>Animation of Metaball</i> .....	36
5.4.5. <i>JDBC-ODBC bridge to connect to Access database</i> .....	40
5.4.6. <i>Metaball algebra function used to represent coupling</i> .....	41
5.5. PUBLIC METHODS .....	42
<b>CHAPTER 6 CONCLUSION</b> .....	<b>44</b>
6.1. SUMMARY .....	44
6.2. FUTURE WORK.....	44
<b>REFERENCES</b> .....	<b>46</b>
<b>APPENDIX A: EDGETABLE IN MARCHING CUBES ALGORITHM</b> .....	<b>49</b>
<b>APPENDIX B: METABALL API PACKAGE CLASS HIERARCHY</b> .....	<b>50</b>
<b>APPENDIX C: SNAP SHOTS USING METABALL API PACKAGE</b> .....	<b>51</b>



## LIST of FIGURES

FIGURE 3.1: DIFFERENT ENERGY INFLUENCE FUNCTIONS .....	10
FIGURE 3.2: METABALL ISO-SURFACE EXAMPLES .....	11
FIGURE 3.3: FACET DETERMINATION IN MARCHING CUBES ALGORITHM .....	12
FIGURE 3.4: THE 15 CASES OF ISO-SURFACE DETERMINATION WITHIN A CUBE .....	13
FIGURE 4.1: JAVA 3D APPLICATION SCENE GRAPH .....	19
FIGURE 5.1: METABALL ISO-SURFACES RENDERED AT DIFFERENT RECURSION LEVELS ..	31
FIGURE 5.2: METABALL RENDERING USING FLOAT (RIGHT) OR DOUBLE(LEFT) FOR GEOMETRY SPECIFICATION.....	36
FIGURE 5.3:MEATBALL COUPLING WITHOUT ANY COLOUR INTERPOLATION .....	41
FIGURE B.1 METABALL PACKAGE HIERARCHY .....	50
FIGURE C.1: SNAPSHOT OF A METABALL CONFIGURATION OF SOFTWARE ENTITIES AND THEIR INTERRRELATIONSHPS .....	51
FIGURE C.2: ANOTHER SNAPSHOT OF A METABALL CONFIGURATION IN THE PROCESS OF SOFTWARE VISUALIZATION .....	52
FIGURE C.3: HEAVILY COUPLED METABALL CONFIGURATION IN SOFTWARE VISUALIZATION .....	53

# **Chapter 1: Introduction**

## **1.1. Software visualization**

For many years the primary and fundamental way to understand a computer program or its execution was only to examine its source code and utilize a debugger manually. In recent years, with the rapid development of large OO (object oriented) software, it has become very hard for designers to maintain and trace the relationships of software entities. Software visualization can be used to visualize large software and help in comprehending, debugging and testing of the software. Software visualization is primarily concerned with the use of computer graphics and animation to help illustrate and present computer programs, processes, and algorithms. Software visualization systems can also be used in teaching, to help students understand how algorithms work, and they can be used in program development as a way to help programmers understand their code better [5]

## **1.2. 3D Graphics for software visualization**

In recent years, most of the systems for visualizing and animating computational process have been developed as aids for program understanding. These systems have mostly been restricted to exhibiting 2D graphical imagery such as Rational Rose. 3D graphics could be used more efficiently to visualize large software. 3D graphics provide an extra dimension to encode more information. This report describes the design and

implementation of a 3D graphics package known as the metaball API package which can be used to create 3D visualizations of large software [11].

### **1.3. Metaball technology**

Metaball is a 3D modeling technique that blends and transforms an assembly of spheres into a complex shape. Metaball is defined by a so-called three-dimensional variable density field, radiating from a given center point. A point on a metaball surface is constructed at all points in the field with the same density value, which is given by the user or derived from the visualization context. The value of the field can vary linearly with distance from the center, or in any other way expressible via a mathematical formula. Metaballs, also known as blobby objects, are a type of implicit modeling technique.[16] Metaball is like equal electric potential of some electric charges or equal density surface of some density distribution. We have used metaball technology to implement a metaball package, with a suitable API that can be used to create various metaball configurations and their visualizations.[13,14]

### **1.4. Metaball API package**

Very simply stated, our metaball package implementation can be considered as being an extension to the JAVA 3D 1.3 API. It is implemented using JAVA 3D. The API includes functions to create meatballs each with the different radius, shading, color, and texture mapping. The metaball itself is used to represent the software entities (spherical surface) and relationships among these entities are represented by a metaball of a different shape (cylindrical surface).

## **1.5. Organization of this report**

Chapter 2 gives a state of the art survey of software visualization techniques. Chapter 3 presents the metaball formulation and the marching cube algorithm that is to be used for rendering a metaball configuration. Chapter 4 introduces Java 3D package and associated computer graphics concepts. In chapter 5 we present the design and implementation of the metaball package. Chapter 6 contains conclusions based on the experience in using the metaball package and also further extensions of this work. Appendices A and B include details of the metaball package, while Appendix C includes some snap shots of the use of this package in software visualization.

## **Chapter 2: A Brief Survey of Software Visualization Techniques**

Software visualization is the use of computer graphics and animation to help illustrate and present computer programs, processes, and algorithms. Software visualization systems can be used in teaching to help students understand how algorithms work, and they can be used in program development as a way to help programmers understand their code better. Traditional program understanding methods use code tracing and debugging which is one dimensional, and static. For a software engineer, the most difficult thing is to know the dynamic aspects of the program. Software visualization can provide graphical depictions to explain, illustrate, and show how computer software functions. If a visualization is designed well, software engineers can gain an understanding of the inherent process that will be extremely difficult to obtain using traditional program methods such as tracing and debugging.

One approach to build 3D visualizations would be to use an existing 3D graphics package or library such as OpenGL, DirectX, Java 3D or PHIGS+ , as all these packages provide a suite of graphical shapes including lines, rectangles, circles, polygons, arrows, blocks, spheres, and text. Furthermore, these graphical shapes are able to undergo changes in position, size, color, and visibility, providing a dynamic sense of animation. The metaball API is a class library built using Java 3D and can be easily used to create sophisticated 3D animation .[11]

## **2.1. History of software visualization**

Software visualization is by no means a new idea to people. An excellent survey can be found in the paper by Price, Baecker and Small [1], and much of this material is summarized from that paper. Back in 1947, Goldstein and von Neumann used flowcharts and demonstrate its usefulness. Later, in 1959, Haibt developed a system that could draw them automatically from Fortran or assembly language programs. In 1963, Knuth developed a system that integrated documentation with the source code and could automatically generate flowcharts. So in early stages, software visualization was based at a low level, using flowcharts to represent the source code. At that time all languages were largely procedure oriented and flowcharts could visually represent the flow of the source code. All these early approaches to software visualization were static. In 1966, Knowlton was the first to use dynamic techniques as opposed to static techniques and the first to address the visualization of the data structure. He showed visuals of list manipulation. From 1970, software visualization continues to be developed. In 1975 Ledgard describe the use of spacing, indentation and layout to make source code easier to read in a structured language. 1980's, along with the windows GUI technology, saw the beginning of modern SV (software visualization) research with the introduction of the bit-mapped display and use interface. The most important and famous system of this era was BALSAs which allowed students to interact with high level dynamic visualizations of Pascal programs.[1]

## **2.2. Software visualization systems review**

There are many software visualization systems available. But for our brief survey we chose the systems below based on their historic importance and their diversity and difference in approaches as described in [1].

### **2.2.1. Sorting Out Sorting**

Sorting Out Sorting (SOS) is the first major software visualization work in 1980's. It was produced at the University of Toronto. In the study of algorithms, sorting is very important. There are many kinds of sorting algorithms that are described in text and using diagrams. However, SOS system uses a different technique, which is animated computer graphics, to explain how nine different sorting algorithms manipulate their data. The data items are typically represented by different colors, blue or green rectangles, with each having a different height to allow users to visualize the sorted data. As soon as one or more data items are being considered by the sorting algorithm, they will be highlighted and when an element has reached its final position which means it has been sorted already, it will turn red.[1] Furthermore, SOS system also illustrates the speed differences of each algorithm. Today, SOS is widely used for introductory computer science teaching at secondary and post-secondary levels.

### **2.2.2. Polka-3D**

Polka-3D is a 3D animation software visualization system used to support the development of 3D software visualizations. It is an object-oriented animation methodology similar to Metaball API. It contains classes that model the entire animation, individual views of animations and windows, and the entities to help define a view such as graphical objects and action or motions. Polka-3D is mainly composed of AnimObject

class which provides a simple object modeling capability, Location object which is simply a user-placed marker sitting somewhere within the 3D coordinate system and a special Eye or Viewer object which controls the position and direction of the viewpoint. Polka-3D is implemented in C++ on SGI workstations using the GL graphics library. So Polka-3D is a straightforward, general purpose 3D animation methodology that could be used to build many other information visualizations and animations. Polka-3D is also one of the first systems to bring 3D graphics capabilities to many programmers who desire 3D views to apply 3D visualization technique for software comprehension.[15]

### **2.2.3. ANIM**

ANIM, which is developed at AT&T Bell Laboratories, is a simple but powerful software visualization system for producing both animated visualizations on a workstation as well as static snapshots ready for inclusion in documents. ANIM uses some script commands to generate animations such as movie or still images. There are only eight commands in total. For example, line, text, box and circle is for drawing command and view. Click, erase and clear is for control commands. So ANIM is very easy for users to learn. [1]

### **2.2.4. TPM**

TPM(Transparent Prolog Machine) which is developed at the UK's Open University, was first announced in 1986. It is one of the most successful automatic systems which is used as a graphical tracer and debugger for the declarative language Prolog. The most recent version of TPM is based on the Macintosh. TPM provides two basic views for the user which is CGV(coarse-grained view) and AORTA diagram(fine-grained view). The CGA uses the tree to show the execution space of the entire program with each node representing goals. Each node has different colors which is used to indicate the state of



the goal which represents pending, succeeded, failed or initially succeeded but failed on backtracking. AORTA diagram is the And/OR tree which allows the user to zoom in on a particular node to get details of data flow such as variable instantiation. Therefore, TPM is especially suitable for debugging large programs particularly for large Prolog programs. TPM and the AORTA notation have been used extensively by students in the Open University's distance teaching program as well as interactive classroom teaching.

[1]

### **2.2.5. Pavane**

Pavane software visualization system is designed to provide declarative three-dimensional visualizations of concurrent programs. Programming and debugging in a parallel language can be much more complex than in conventional sequential programming environment because of the greater number of computational elements interacting each other. Pavane was implemented in five parts which are a parser, run-time package, a library of routines that is used when visualizing C programs, a second run-time package and a viewing program which renders the visualization and provides the user interface. Pavane is an active research prototype and a compelling indication of the power of 3-D color animations to aid in the understanding of both sequential and concurrent programs. [1]

## Chapter 3: Metaball Technology

### 3.1. What is metaball

A metaball is an implicit representation defined by a function that takes 3D coordinates of a point as input, and forms an isosurface with some output value. Metaball is comparable to an electric potential of some electric charges or equal density surface of some density distribution. Metaballs, also known as blobby objects, are a type of implicit surface modeling technique. Implicit surface is the modeling method that represents objects and their surfaces implicitly with a mathematical function of the form  $F(x,y,z) = 0$ . [13,14]

We can think of a metaball as a particle surrounded by a density field, where the density attributed to the particle (its influence) decreases with distance from the particle location. So in algebraic function form, if we are using the implicit equation:

$f(x,y,z) = R^2 / ((x-a)^2 + (y-b)^2 + (z-c)^2)$ , where  $R$  is the radius of the spherical metaball and the metaball center is at  $[a,b,c]$ . With multiple metaballs, we can define individual algebraic functions  $f_1, f_2, f_3, \dots, f_n$ . Then the combination of metaballs is the sum of all these functions:  $f_1 + f_2 + f_3 + \dots + f_n$ . Any point for which the sum of the metaball equation is less than the given iso surface value is inside the volume. If you are drawing it and visualize this, you will see that the metaballs now attract and deform each other as you would expect them to. The more precise metaball algebra function is as below:

$$D(r) = \begin{array}{ll} a*(1-3r^2/b^2) & 0 \leq r \leq b/3 \\ 3a/2*(1-r/b)^2 & b/3 \leq r \leq b \\ 0 & b \leq r \end{array}$$

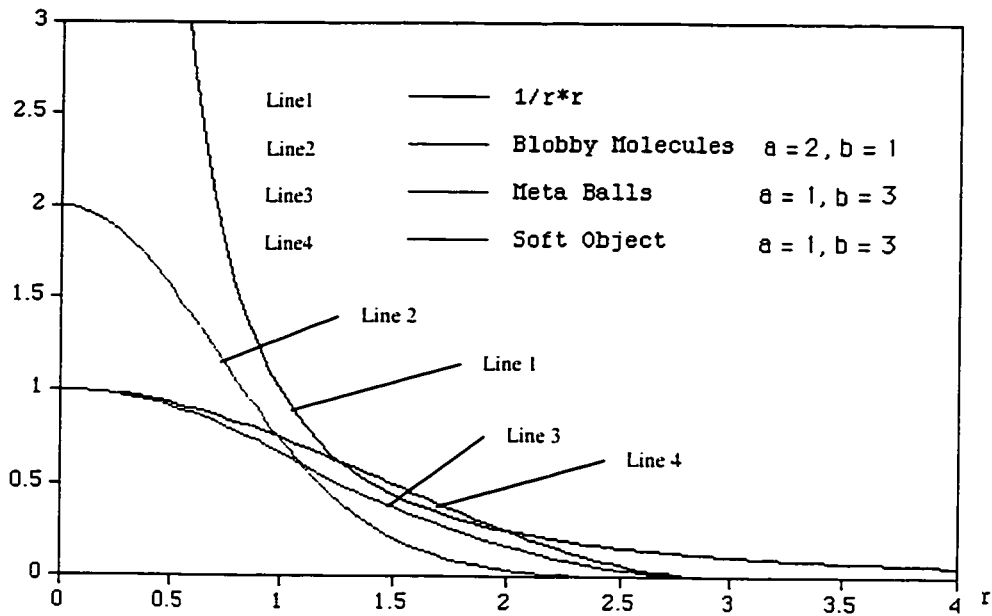


FIGURE 3.1: DIFFERENT ENERGY INFLUENCE FUNCTIONS

In Figure 3.1 , you can see that the line 3 shows the metaball algebra function for a decreasing influence function curve, which shows that (its influence) decreases with distance from the center of the particle location. [16]

A metaball surface is implied by taking an isosurface through this density field - the higher the isosurface value, the nearer it will be to the particle. The powerful aspect of metaballs is the way they can be combined. By simply summing the influences of each metaball on a given point, we can get very smooth blendings of the spherical influence fields. An isosurface is a surface in 3D space, along which some function is constant. For example, all the points that satisfy the function  $f(x,y,z) = C$  forms the Iso-surface.  $C$  is the Isovalue.

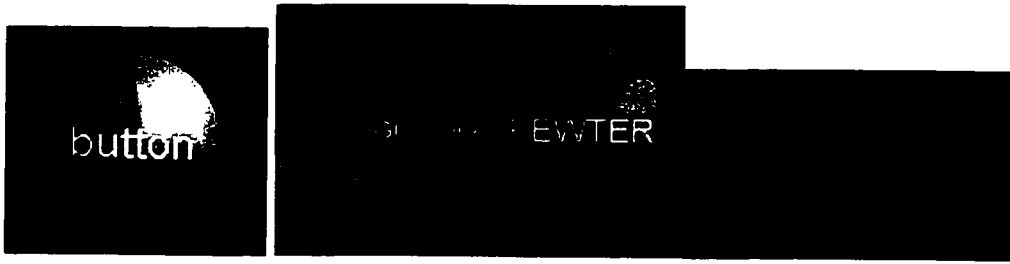


FIGURE 3.2: METABALL ISO-SURFACE EXAMPLES

Figure 3.2 shows some examples of isosurface visualization of metaball configuration.

## 3.2. Marching Cubes Algorithm

### 3.2.1. Principle of Marching Cubes Algorithm

It is difficult to render the iso-surface only by the algebra function. To get a perfect accurate metaball volume, we need to evaluate the above equations for a every pixel on the screen and for every possible depth value. This will cost us too much calculation time and take up too much of CPU processing time. One way to simplify rendering is by using approximations using the so-called marching cubes algorithm. This algorithm generates a polygonal 'skin' around the metaball volume which is very easy and fast to render. The basic idea is to create a regular 3D grid of points. You interpret this grid as an array of cubes. If a cube has vertices inside and outside the metaball volume, linear interpolation is used to estimate where the metaball iso-surface intersects the cube, and a polygon is generated at that location. The indexing convention cube for vertices and edges used in the algorithm is shown below:

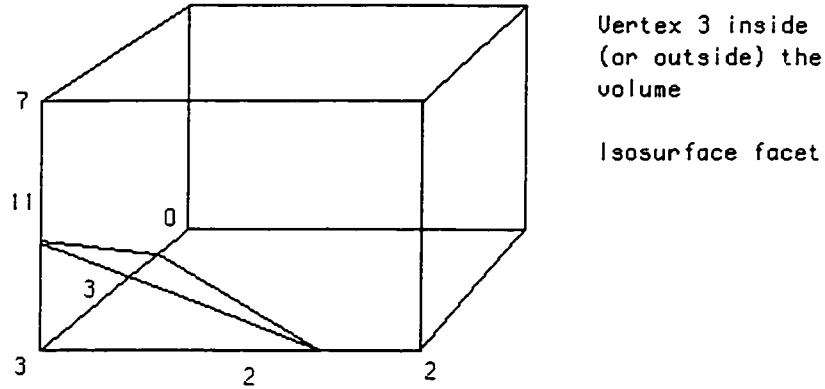


FIGURE 3.3: FACET DETERMINATION IN MARCHING CUBES ALGORITHM

For example, in figure 3.3, if the value at vertex 3 is below the iso surface value and all the values at all the other vertices were above the iso surface value then we would create a triangular facet which cuts through edges 2,3, and 11. The exact position of the vertices of the triangular facet depend on the relationship of the iso-surface value to the values at the vertices 3-2, 3-0, 3-7 respectively. For that we can use linear interpolation.  $P = P_1 + \frac{(\text{isovalue} - V_1)(P_2 - P_1)}{(V_2 - V_1)}$ .  $P_1$  and  $P_2$  are the vertices of a cut edge and  $V_1$  and  $V_2$  are the scalar values at each vertex. The precise intersection point  $P$  is given by the function above. But it is difficult to derive a consistent triangle facet combination for each solution so that facets from adjacent grid cells connect together correctly. [17]

### 3.2.2. Basic principle used to generate isosurface facets

The Marching cube algorithm describes how to derive the triangle facet combination using edgeTable that describes all kinds of facet combinations. The maximum triangle facets in each condition is 5 triangle facets. The detailed edgeTable is given in Appendix A. After presenting the basic principle of Marching Cubes algorithm, we discuss now how the principle can be made to work in 3D space. In the 3D space, we are dealing with

cube's that have 8 corners and therefore a potential 256 possible combinations of corner status. However to simplify the algorithm we can reduce the complexity by taking into account cell combinations that duplicate under the following conditions:

- Rotation by any degree over any of the 3 primary axis
- Mirroring the shape across any of the 3 primary axis
- Inverting the state of all corners and flipping the normals of the relating polygons.

Taking this into account we can reduce the original 256 combinations of cell states down to a total of 15 combinations. With this reduced number it is then easy to create predefined polygon sets for making the appropriate surface approximation. The image below gives an example data set covering all of the 15 possible combinations. The spheres denote corners that have tested as inside the shape and the arrows denote the surface normals of the relevant triangles.

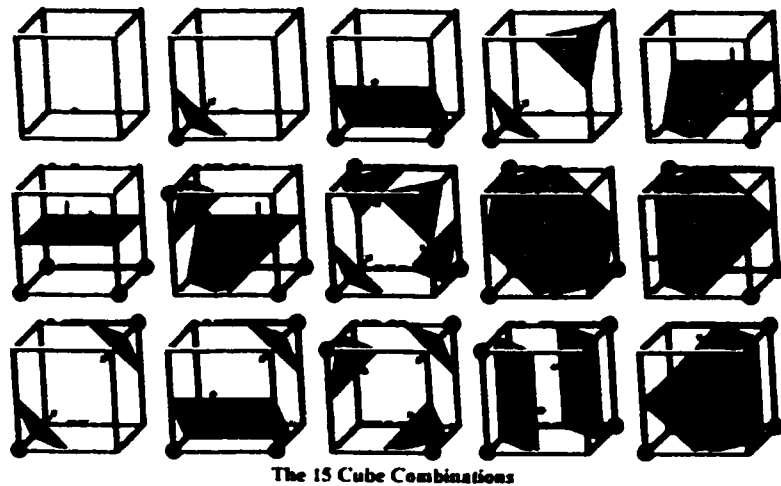


FIGURE 3.4: THE 15 CASES OF ISO-SURFACE DETERMINATION WITHIN A CUBE

As we can see the marching cubes algorithm has produced a very close approximation to the original surface, a sphere. When polygonising a field where the values are known or can be interpolated in space, we can use the resolution of the sampling grid to control the

precision and smoothness of the metaball. The smaller the grid size, the more smooth the metaball. We will describe this in more detail in Chapter 5. [21]

## **Chapter 4 Overview of Java3D**

Our metaball API package is designed and implemented using Java3D v1.3. Java 3D is a standard extension to the Java 2 JDK which is used to display and interact with three-dimensional graphics. Java 3D is composed of a bundle of high level API function calls to support the tasks of creation of imagery, visualizations, animations, and interactive 3D graphics application. The Java 3D class library provides a simpler and easier interface than most other graphics libraries, but has enough capabilities to produce good 3D visuals and animations. Java 3D builds on existing technology such as DirectX and OpenGL and the programs do not run as slowly as you might expect. Also, Java 3D can incorporate objects created by 3D modeling packages like VRML models. Java 3D is a high level API which is implemented above the OpenGL and DirectX, giving rendering hints to the 3D pipeline and controlling the objects in the scene through the data structure of a scene graph. [19,20]

### **4.1. Java3D goal**

#### **4.1.1. Simplify 3D graphics application development**

Java3D is in the high level abstraction layer which is above OpenGL and DirectX. It supports more complex and advanced APIs to simplify 3D graphics development. Java3D is easy to use by 3D graphics developers to create sophisticated 3D graphic visuals.

#### **4.1.2. Make it ideal for intranet and internet visualization applications**

Java3D is based on the JAVA platform. It is embedded into JAVA J2EE architecture. JAVA programming language is widely used in Intranet and Internet



programming. Java applets can be used in Java3D development to create intranet and Internet visualization applications.

#### **4.1.3. Performance**

Java 3D API is the hierarchy of Java classes which serve as the interface to a sophisticated three-dimensional graphics rendering and sound rendering system. All the geometric objects reside in a tree (scene graph) tracing from virtual universe, which is then rendered. This greatly enhances the performance of 3D graphics rendering.

### **4.2. Features of Java 3D**

#### **4.2.1. High-level scene-graph model:**

This allows developers to focus on the objects and the scene composition so as to free the programmer from spending time and effort designing specific geometric shapes and writing rendering code for the scene display

#### **4.2.2.Run-time loaders:**

This allows Java3D to accommodate a wide variety of file formats. The Java3D API enables a much broader range of developers to create sophisticated 3D applications.

#### **4.2.3. Geometry compression**

This allows very large 3D models to be rapidly downloaded over the network for remote viewing and manipulation, reducing the impact of potential bottlenecks in network bandwidth.

#### **4.2.4. Takes advantage of existing hardware accelerators**

Java3D does this via its use of low-level APIs such as OpenGL and Direct3D. This allows Java3D implementations to tune and scale the application's scene graph to the underlying hardware for maximum performance.

#### **4.2.5. Flexible Viewing Model**

An application or applet written using the Java 3D API view model can render images to a broad range of display devices including flat screen displays, stereo displays, portals/caves, and head-mounted displays, all without modification to the code.

#### **4.2.6. Level of Detail (LOD)**

The Java 3D API includes support for multiple levels of detail, enabling the end-user to view the nearest or most important objects at increased resolutions, thereby improving both application performance and the user experience.

#### **4.2.7. Support for continuous action devices:**

The Java 3D API can accept input from continuous action devices, such as trackers, increasing the interactive capabilities of Java 3D applications

### **4.3 Scene Graph Structure ( shown in Figure 4.1)**

#### **4.3.1. VirtualUniverse Object**

A VirtualUniverse object consists of a name and a list of Locale objects that contain a collection of scene graph nodes that exist in the named universe. Typically, an application will need only one VirtualUniverse, even for very large virtual databases.

#### **4.3.2. Locale Object**

The Locale Object acts as a container for a collection of subgraphs of the scene graph that are rooted by a BranchGroup node. A Locale also defines a location within the virtual universe using high resolution coordinates (HiResCoord) to specify its position. This HiResCoord serves as the origin for all scene graph objects contained within the Locale.

### **4.3.3. Node Object**

In Jav3D Node Objects includes Group Node Objects and Leaf Node Objects. There are four kinds of Group Node Objects which are BranchGroup objects, TransformGroup objects, OrderedGroup objects and SwitchGroup objects. The BranchGroup object is the only object that connects to the Locale object as its parent. BranchGroup objects are the root of a subgraph, or branch graph. There are two different categories of scene subgraphs: the view branch graph and the content branch. The content branch graph specifies the contents of the virtual universe. The view graph specifies the viewing parameters such as the viewing location and direction. TransformGroup Objects are the Group node that contains a transform. The TransformGroup node specifies a single spatial transformation, via a Transform3D object, that can position, orient, and scale all of its children. OrderedGroup Objects is the OrderedGroup node which is a Group that ensures its children render in increasing index order. SwitchGroup Object is the object for monitoring switches and contains a list of errors, as well as a list of all switches. LeafNode Objects in JAVA3D are the objects which can not have any child objects. It consists of Sound object, Light object, Shape Object and Fog Object.

### **4.3.4. Behavior Objects**

Behavior objects are the base for Interaction and Animation in Java 3D. The Behavior objects provide methods to process keyboard and mouse inputs, react to movements as well as to enable and process pick events.

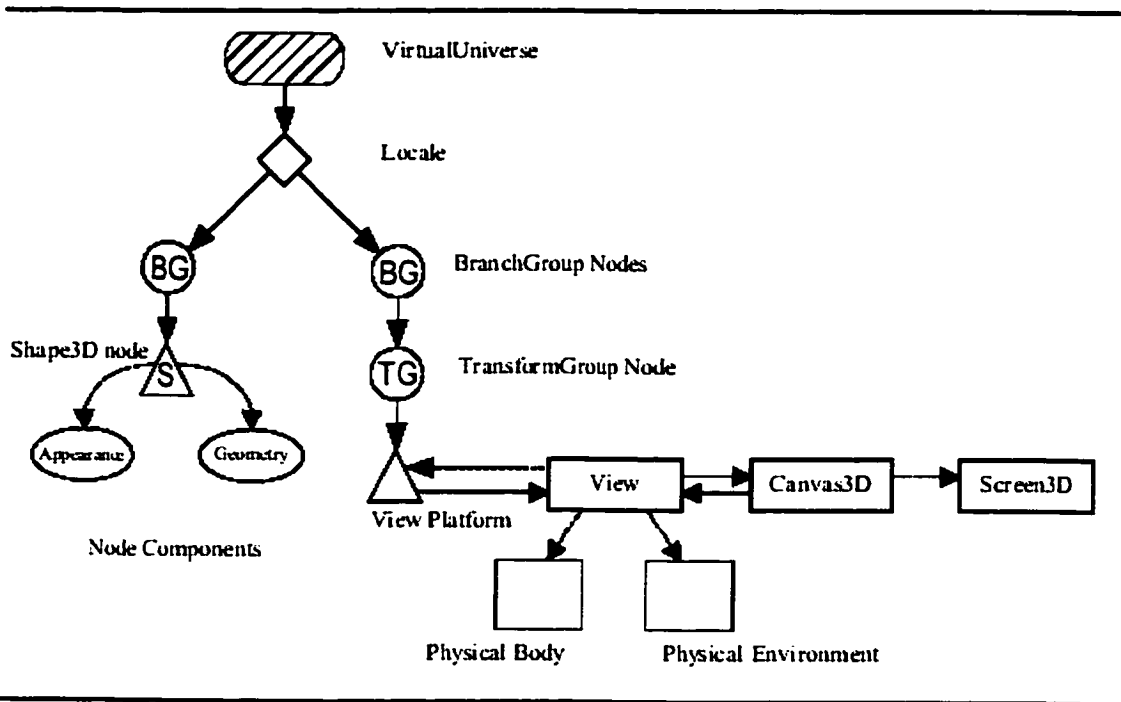


FIGURE 4.1: JAVA 3D APPLICATION SCENE GRAPH

#### 4.4. Convenient facilities provided by Java 3D

Java 3D provides high level constructs for creating and manipulating 3D geometric objects. Java 3D API is a hierarchy of Java classes, which serve as the interface to a sophisticated three-dimensional graphics engine.

##### 4.4.1. Simple KeyNavigator Behavior

In the Metaball API package implementation, we need to navigate in 3D space to change to any random viewpoint to see the different parts of the whole metaball visualization. All the KeyNavigatorBehavior movements have been already defined in JAVA 3D inherently. Even in OpenGL, we need to define these using glut function calls. Without Java 3D, we need to program the KeyNavigator function using windows keyboard event handlers and change the viewpoint, which will cost extra time because

this function is only used to test and check the integrity and correct rendering of the metaball. With KeyNavigator Behavior, we only need to use several lines to achieve these tasks in the Metaball API Package implementation:

```
/* add KeyNavigatorBehavior */  
  
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(viewTrans  
form);  
  
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0)  
);  
  
keyNavBeh.setEnabled( true );  
  
objectRoot.addChild(keyNavBeh);
```

#### **4.4.2. Interpolators and Alpha Object**

Animations in Java3D are implemented using Behavior objects. The Java3D API, Interpolator object together with Alpha object manipulates some parameter of a scene graph object to create a time-based animation. In the Metaball API package implementation, some simple animations are required. In Java3D, we can use Interpolator and Alpha object to easily finish this task. In the implementation, we use RotationInterpolator and Alpha classes to let the metaball rotate along the x, y z axis. Further details are presented in Chapter 5.

#### **4.4.3. Easy expanding and across platform**

Java3D is embedded in Java language and is object oriented. By taking advantage of Java, the Java 3D program is composed of classes and objects that create instances of Java 3D objects and places them into scene graph data structure which is an arrangement of 3D objects in a tree structure that completely specifies the content of a virtual universe.

and how it is to be rendered. So every class in this implementation is easy to be inherited and further specialized by other programmers. Furthermore, this Metaball API package can not only be used in Windows 2000 OS but also in Unix and other operating systems without changing or adapting the source code.

## **Chapter 5 Design and Implementation of Metaball Package**

The Metaball-API is similar to Polka-3D which is an animation methodology and toolkit/class library designed and implemented using an object-oriented methodology. It can be used to support 3D animation development by programmers who are engaged in large object oriented source code visualization. Metaball-API is a 3D computer graphics application with a programmer interface based on the metaball visualization approach to represent and visualize software entities and the relationship among these entities. It is implemented in Java on WINDOWS 2000 using the Java 3D graphics library. The primary software visualization task is encapsulated in a class known as MarchingCube class. The MarchingCube class mainly defines the marchingcube algorithm to recursively compute the metaball surface. Similar to Polka-3D which can create a sphere AnimObject and move it, metaball API can be used to create a metaball object and set it to any 3D space location. Further more, metaball API can also set different color and shading for different meatballs.

The Metaball package has been designed using Object-Oriented methodology. Object-Oriented Design (OOD) centers on finding an appropriate set of classes and defining their contents and behavior. It involves determining the proper set of classes and then filling in the details of their implementation. Object-oriented design is fundamentally a three-step process: identifying the classes, characterizing them, and then defining the associated actions. It is composed of a number of different classes that can be inherited and extended.

## **5.1. Understanding the requirements**

The first step to create an object-oriented design is to understand the problem. This system is about a package to be used to create meatballs. Hence after a detailed study the main requirements are concluded as the following:

1. API functions and parameters in each API functions
2. Easy to expand and inherit
3. Can be used in different platforms

Some problems are complex and they will probably be divided to more sub problems: such as “parameters in each different API functions” can be grouped by “how to set default values”, “what type of parameters”, “how to use constructor”, etc.

For these problems, testing scenarios were created so that they will tell us if the API is working:

1. This package provides an interface for the people who will do the software visualization using meatballs
2. This metaball API can be used to create different size, color and texture mapping for the metaballs
3. It can be used to create coupling between two meatballs
4. The metaball API is created as an extension of Java3D API class that can be called.

## **5.2. Metaball package classes**

1. Pt3d Class:

the point (x,y,z) coordinate in 3D space

2. TRIANGLE Class:



contains three point(x,y,z) that forms a triangle

### 3. GRIDCELL Class:

contains eight points(x,y,z) that forms a voxel used in marchingcube algorithm

### 4. MarchingCube Class:

the class implementing the marchingcube algorithm and used to cut the voxel grid to draw the metaball isosurface

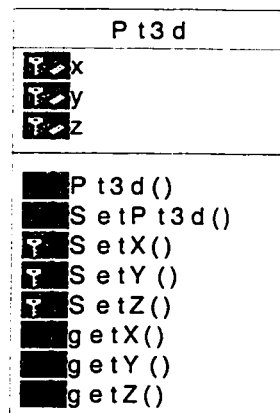
### 5. Metaball Class:

The main test class which is used to call the API functions

## 5.3. Class architectures and implementations

As part of this report, the implementation was limited to the development of an API package, therefore no IDE and user interface was required. The relationships between these classes are not very complicated and each class has its own attributes and operations. We use Unified Modeling Language(UML) to describe our classes and their relationships. UML is a standard notation for writing software blueprints and it may be used to graphically depict, specify, construct and document the artifacts of a software-intensive system.

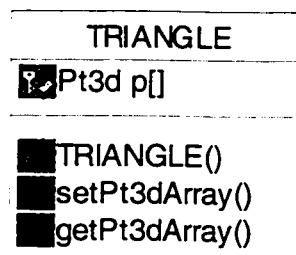
### 5.3.1. Pt3d Class:



### In Pt3d class:

- attribute x is the x coordinate value of the 3D point
- attribute y is the y coordinate value of the 3D point
- attribute z is the z coordinate value of the 3D point
- Constructor:
  1. Pt3d() { setPt3d( 0, 0, 0 ); } is to set the default 3D coordinate to (0,0,0) without parameter
  2. Pt3d( float a, float b, float c ) { setPt3d( a, b, c ); } to set the value of a 3D point with three float parameter by the programmer
- SetPt3d( *float xvalue, float yvalue, float zvalue* ) is to set the 3D coordinates of any 3D points
- getX() is to get the x coordinate value of 3D point
- getY() is to get the y coordinate value of 3D point
- getZ() is to get the z coordinate value of 3D points
- SetX() is to set the x coordinate value of the 3D point
- SetY() is to set the y coordinate value of the 3D point
- SetZ() is to set the z coordinate value of the 3D point








### 5.3.2. TRIANGLE Class:



**In TRIANGLE class:**

- p is the Pt3d array used to represent coordinates of triangles
- Constructor:  
TRIANGLE() is the constructor to set the 3D point coordinate to (0,0,0)
- setPt3dArray() is to initialize the array Pt3d value
- getPt3dArray() is to get the array Pt3d value

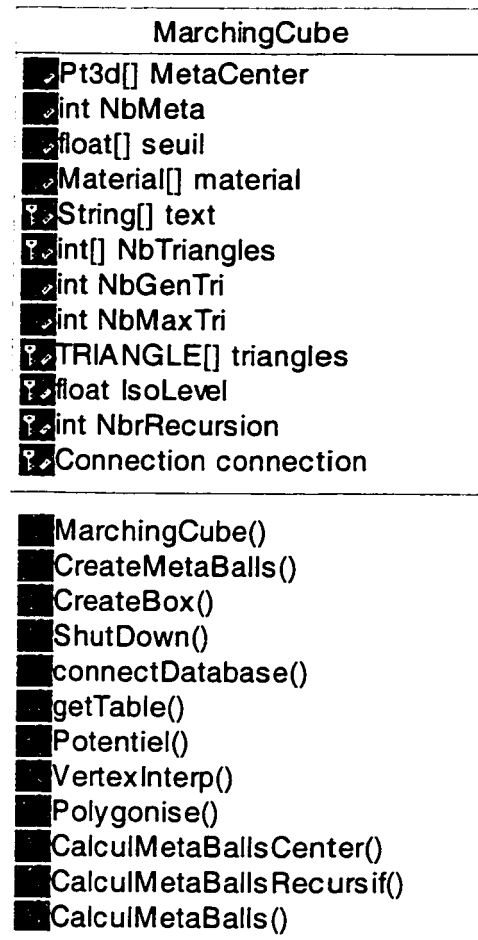
**5.3.3. GRIDCELL Class:**

GRIDCELL	
	Pt3d[] p
	float val
	GRIDCELL()
	setPt3dArray()
	setValArray()
	getPt3dArray()
	getValArray()

**In GRIDCELL class:**

- p is the Pt3d array used to represent coordinates of triangles
- val is the attribute value after interpolation used to compare with isovalue
- Constructor:  
GRIDCELL() is the constructor to set the 3D point coordinate to (0,0,0) and the val value associated with each value
- setPt3dArray() is to initialize the array Pt3d value
- setValArray() is to initialize the array Val value
- getPt3dArray() is to get the array Pt3d value
- getValArray() is to get the array Val value

### 5.3.4. MarchingCube Class:



In **MarchingCube**, there are nine main methods, one constructor and two methods for database access.

- MetaCenter is the meatball center coordinates
- NbMeta is the number of meatballs
- Seuil is the distance between the metaball center and the influence point
- material is the meatball material properties which are R, G, B values
- text is the text used to texture mapping to the meatball
- NbTriangles is the number of the triangles generated totally

- NbGenTri corresponds to the number of the triangles generated by the marchingcube algorithm
- NbMaxTri is the maximum triangle numbers which can be generated by the marchingcube algorithm
- Triangles are the coordinate of each triangle generated
- IsoLevel is the isovalue of the meatball
- NbrRecursion is the number of recursion used by the marchingcube algorithm, the larger the number, the more precise rendering the meatball. But the rendering speed will slow down when the number of recursion is increased
- Connection is used to connect to the database
- Constructor:

Constructor MarchingCube is used to initialize the default values of the meatballs

1. Initialize contents by creating an array of MetaballCenter with size 300 used to represent the center of 3D metaball coordinate
2. Initialize contents by creating an array of text with size 300 used for texture mapping to the surface of the metaball
3. Initialize contents by creating an array of Seuil value with size 300 used to represent the R2 value of metaball
4. Initialize content by creating an array of material with size 300, which contains the material information(eg. Ambient, Diffuse, Shininess) of each metaball.
5. Initialize contents of creating an array of NbTri with size 300 which represents the number of triangles generated by the marchingcube algorithm

- 6. Initialize the metaid value which is used to separate different metaballs
  - 7. Initialize the NbMaxTri value with size 50000 which represents the maximum number of triangles generated in the marching cube algorithm
  - 8. Initialize the NbrRecursion value which represents the number of recursion used in the Octree data structure in marchingcube algorithm
  - 9. Initialize the Isovalue which is used in the iso-function(metaball algebra function) at the right hand side
  - 10. Initialize 50000 TRIANGLE objects which contains the 3D coordinate of the triangles generated by the marching cube algorithm
- CreateMetaBalls() initializes the metaball location center of the metaball.
  - CreateBox() creates a transparent box object in Java3D around the metaball used to texture mapping the name of the metaball
  - connectDatabase() is to connect to the Access database through JDBC-ODBC bridge.
  - ShutDown() is used to close the database
  - getTable() fetches the the metaball information which is metaballID, metaball location, R1, R2, Ambient(R,G,B), Diffuse(R,G,B) and Shininess as well as metaball name from the Access database
  - Potentiel(Pt3d p) calculates the Iso-function(metaball algebra function:  $(Seuil[i]-l)*(Seuil[i]-l)/(Seuil[i])$ ), here Seuil[i] is the R2 value of the metaball, R1 is 0. l is the distance between the point in 3D space and the metaball center. The function is the approximate of the metaball algebra function.
1. if l is less than Seuil[i]  $fx += (Seuil[i]-l)*(Seuil[i]-l)/(Seuil[i]);$

2. if  $l$  is larger than  $Seuil[i]$  then 0, which means this point has no effects with the metball algebra function
- `VertexInterp()` linearly interpolates the position where an isosurface cuts an edge between two vertices, each with their own scalar value
  - `Polygonise()` is using a grid cell and an isolevel(isovalue), calculate the triangular facets required to represent the isosurface through the cell. Return the number of triangular facets, the array “triangles” will be loaded up with the vertices at most 5 triangular facets. 0 will be returned if grid cell is either totally above or totally below the isovalue
  - `CalculMetaBallsCenter()` is used to calculate the center position of the metaball
  - `CalculMetaBallsRecurisif()` is used to recursively cut the cubes with octree data structure.
  - `CalculMetaBalls()` initializes the cubic length and to call the `CalculMetaBallsRecurisif` function.

## **5.4. Algorithms used in Metaball API package implementation**

The main algorithm used in Metaball API package implementation is the marchingcube algorithm, which is used to render the isosurface of the metaball algebra function.

### **5.4.1. Marchingcube class design with recursion**

In marchingcube algorithm, we use octree to recursively cut the voxel grid into 8 parts. The more level of recursion, the more precision the isosurface is. Figure 5.1 shows

the same isosurface generated at different increasing levels of recursions by reducing the grid size.

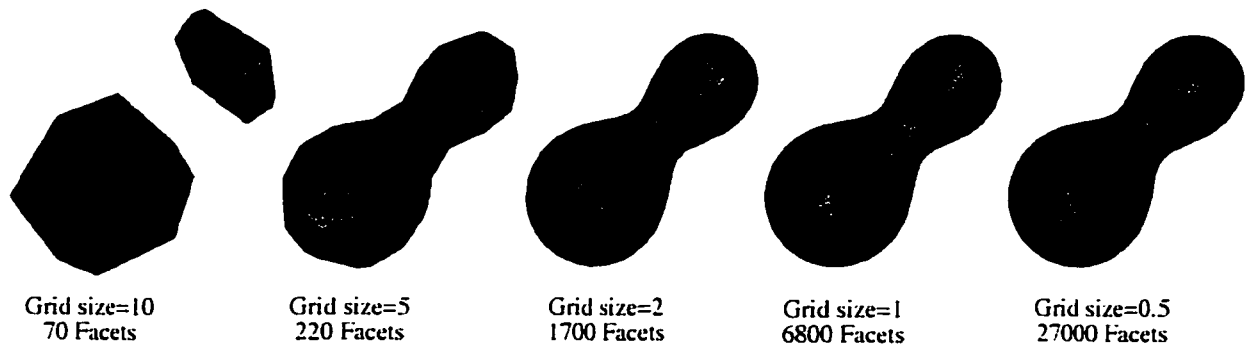


FIGURE 5.1: METABALL ISO-SURFACES RENDERED AT DIFFERENT RECURSION LEVELS

Shown below the is source code of `CalculMetaBallRecurisif` function:

```
public void CalculMetaBallsRecurisif(float xmin,float xmax,float ymin,float
ymax,float zmin,float zmax,float cz,float cx,float cy,int n,int id)
{
    GRIDCELL gc = new GRIDCELL();
    int j,flag1,flag2;
    float[] m = new float[3];
    gc.setPt3dArray( 0, xmin, ymax, zmax );
    gc.setPt3dArray( 1, xmax, ymax, zmax );
    gc.setPt3dArray( 2, xmax, ymax, zmin );
    gc.setPt3dArray( 3, xmin, ymax, zmin );
    gc.setPt3dArray( 4, xmin, ymin, zmax );
    gc.setPt3dArray( 5, xmax, ymin, zmax );
    gc.setPt3dArray( 6, xmax, ymin, zmin );
```



```

gc.setPt3dArray( 7, xmin, ymin, zmin );
if(n!=0) {
    flag2=flag1=0;
    for (j=0;j<8;j++) {
        gc.setValArray( j, Potentiel(gc.getPt3dArray(j)));
        if(gc.getValArray(j)>IsoLevel) flag1|=1<<j;
        if(gc.getValArray(j)<IsoLevel) flag2|=1<<j;
    }
    /* if the grid cell is either totally above or totally below the isovalue then exit the
    recursion
    if((flag1==255)||((flag2==255)) return;
}
if(n<NbrRecursion) {
    m[0]=xmin+(xmax-xmin)/2;
    m[1]=ymin+(ymax-ymin)/2;
    m[2]=zmin+(zmax-zmin)/2;
    /* recursively to cut the voxel grid according to the isovalue */
    CalculMetaBallsRecursif(xmin,m[0],ymin,m[1],zmin,m[2],cz,cx,cy,n+1,id);
    CalculMetaBallsRecursif(m[0],xmax,ymin,m[1],zmin,m[2],cz,cx,cy,n+1,id);
    CalculMetaBallsRecursif(m[0],xmax,m[1],ymax,zmin,m[2],cz,cx,cy,n+1,id);
    CalculMetaBallsRecursif(xmin,m[0],m[1],ymax,zmin,m[2],cz,cx,cy,n+1,id);
    CalculMetaBallsRecursif(xmin,m[0],ymin,m[1],m[2],zmax,cz,cx,cy,n+1,id);
    CalculMetaBallsRecursif(m[0],xmax,ymin,m[1],m[2],zmax,cz,cx,cy,n+1,id);
}

```

```

        CalculMetaBallsRecurisif(m[0],xmax,m[1],ymax,m[2],zmax,cz,cx,cy,n+1,id);
        CalculMetaBallsRecurisif(xmin,m[0],m[1],ymax,m[2],zmax,cz,cx,cy,n+1,id);
    }
    else {
        if ( NbMetaTri < NbMaxTri-5)
            NbGenTri += Polygonise(gc,IsoLevel,NbGenTri,id);
    }
}

```

#### 5.4.2. Normal vector calculation of metaball surface used in metaball shading

The meatball visualization is created by using the marchingcube algorithm, normal vector needs to be calculated. This is different compared to OpenGL sphere function whose normal vector is already calculated inherently. Here we use `ng[i] = new NormalGenerator()` function call to automatically generate the vertex normal vector in Java3D. The `NormalGenerator` included with the Java 3D utilities generates normals when specifying visual objects using `GeometryInfo` objects. To generate normals, we use the visual object geometry into a `GenetryInfo` object and call `NormalGenerator.generateNormals()`. Below is the source code of normal vector calculation.

```

ng[i].generateNormals(gi[i]);

gi[i] = new GeometryInfo(GeometryInfo.TRIANGLE_ARRAY);

gi[i].setCoordinates(metaballdatas[i].metaballdata);

gi[i].recomputeIndices();

ng[i] = new NormalGenerator();

```

```

ng[i].generateNormals(gi[i]);

st[i] = new Stripifier();

st[i].stripify(gi[i]);

```

### 5.4.3. Texture mapping of metaball surface

In this project, it was required to create texture map texts onto the front box surface of the meatballs. The `text2D` function that creates the texture map directly to the isosurface of the metaball will generate reflections of the text. Here we use `text2D` class in Java3D. `Text2D` objects are rectangular polygons with the text applied as a texture. A `Text2D` object is a representation of a string as a texture mapped rectangle. The texture for the rectangle shows the string as rendered in the specified color with a transparent background. Then we use function call below to automatically generate the texture coordinate.

```

tcg[i] = new TexCoordGeneration(TexCoordGeneration.OBJECT_LINEAR, TexCoord
Generation.TEXTURE_COORDINATE_2);

```

Here we use `TexCoordGeneration.OBJECT_LINEAR` attribute and its texture coordinates are generated as a linear function in object coordinates. Below is the source code of the texture mapping. Figure 5.2 shows the effects of texture mapping on the front surface of box surrounding the metaball.

```

/* create a Java 3D box object surrounding the metaball */

public Box[] CreateBox() {

    Box[] box;

    box = new Box[NbMeta];

    Appearance ap;

```

```

    ap = new Appearance();

    for (int i=0;i<NbMeta;i++) {

        float xdim = Seuil[i];

        float ydim = Seuil[i];

        float zdim = Seuil[i];

        box[i] = new Box(xdim, ydim, zdim, ap);

    }

    return box;

}

/* create texture image and texture coordinates */

text2d[i] = new Text2D(result, new Color3f(0.9f, 1.0f, 1.0f), "Courier", 12,
Font.BOLD);

text2d[i].setString(result);

text2d[i].setCapability( Shape3D.ALLOW_GEOMETRY_READ);

text2d[i].setCapability( Shape3D.ALLOW_GEOMETRY_WRITE);

text2d[i].setString(result);

appearance[i] = text2d[i].getAppearance();

tcg[i]=newTexCoordGeneration(TexCoordGeneration.OBJECT_LINEAR,
TexCoordGeneration.TEXTURE_COORDINATE_2);

tcg[i].setEnabled( true );

appearance[i].setTexCoordGeneration(tcg[i]);

appearance[i].getTexture().setBoundaryModeS
(Texture.CLAMP_TO_BOUNDARY);

```

```
appearance[i].getTexture().setBoundaryModeT
(Texture.CLAMP_TO_BOUNDARY);
```



FIGURE 5.2: METABALL RENDERING USING FLOAT (RIGHT) OR DOUBLE(LEFT) FOR GEOMETRY SPECIFICATION

#### 5.4.4. Animation of Metaball

During debugging large objected oriented software, we need to apply 3D graphic animations. In this project we used Java3D Alpha value to code the animation of metaballs. An alpha object in Java3D represents a value called the alpha value which is between 0.0 and 1.0, inclusive. The alpha value changes over time as specified by the parameters of the alpha object. Alpha is the class in Java3D for creating time varying functions. Most of the time, Alpha is used with RotationInterpolator together to create Java3D animation, such as

```
RotationInterpolator(Alpha alpha, TransformGroup target).
```

In this function, alpha is the time varying function to reference and target is the TransformGroup object to modify.

The source code part of animation is shown below( also included is the main function of Java3D) :

```
public static void main( String[] args ) {
```

```

GraphicsConfigTemplate3D tmpl = new GraphicsConfigTemplate3D ( );
GraphicsEnvironment env =
GraphicsEnvironment.getLocalGraphicsEnvironment();
GraphicsDevice device = env.getDefaultScreenDevice();
GraphicsConfiguration config = device.getBestConfiguration(tmpl);
Canvas3D canvas = new Canvas3D( config );

// Build the view
PhysicalBody body = new PhysicalBody( );
PhysicalEnvironment environment = new PhysicalEnvironment();
View view = new View();
view.addCanvas3D( canvas );
view.setPhysicalBody( body );
view.setPhysicalEnvironment( environment );

// create the universe
VirtualUniverse universe = new VirtualUniverse( );
Locale locale = new Locale( universe );

// Create the view branch
BranchGroup viewRoot = new BranchGroup( );
TransformGroup viewTransform = new TransformGroup( );
viewTransform.setCapability( TransformGroup.ALLOW_TRANSFORM_READ
);
viewTransform.setCapability(
TransformGroup.ALLOW_TRANSFORM_WRITE );

```

```

Transform3D transform = new Transform3D( );
transform.set( new Vector3d( 0, 0, 15 ) );
viewTransform.setTransform( transform );
ViewPlatform vp = new ViewPlatform( );
vp.setCapability(ViewPlatform.ALLOW_POLICY_READ);
vp.setCapability(ViewPlatform.ALLOW_POLICY_WRITE);
vp.setViewAttachPolicy(View.NOMINAL_HEAD);
view.attachViewPlatform( vp );
viewTransform.addChild( vp );
viewRoot.addChild( viewTransform );
viewRoot.compile( );
locale.addBranchGraph( viewRoot );
// Create the object branch
BranchGroup objectRoot = new BranchGroup( );
TransformGroup objectTransform = new TransformGroup( );
transform.set( new Vector3d( 0, 0, 0 ) );
objectTransform.setTransform( transform );
objectTransform.setCapability(
TransformGroup.ALLOW_TRANSFORM_READ );
objectTransform.setCapability(
TransformGroup.ALLOW_TRANSFORM_WRITE );
objectRoot.addChild(objectTransform);
createMetaball();

```

```

for (int i=0;i<NbMeta;i++)
    objectTransform.addChild( metaball[i]);
Alpha rotationAlpha = new Alpha(-1, 50000);
RotationInterpolator rotator =
new RotationInterpolator(rotationAlpha, objectTransform);
/* a bounding sphere specifies a region a behavior is active
create a sphere centered at the origin with radius of 1 */
BoundingSphere bounds = new BoundingSphere();
rotator.setSchedulingBounds(bounds);
objectTransform.addChild(rotator);
// Create the light
AmbientLight ambLight = new AmbientLight(new Color3f(1.0f,1.0f,1.0f) );
ambLight.setInfluencingBounds(new BoundingSphere(new Point3d( 0.0, 0.0,
0.0),1000));
objectRoot.addChild(ambLight);
DirectionalLight dirLight = new DirectionalLight(new Color3f(1.0f, 1.0f, 1.0f
),new Vector3f( 1.0f, 1.0f, 1.0f ) );
dirLight.setInfluencingBounds(new BoundingSphere(new
Point3d(0.0,0.0,0.0),1000.0));
objectRoot.addChild(dirLight);
objectRoot.compile( );
locale.addBranchGraph(objectRoot);
// Create the UI

```



```

javax.swing.JFrame frame = new javax.swing.JFrame( "Metaball Universe");
frame.getContentPane( ).setLayout( new java.awt.BorderLayout( ) );
frame.getContentPane( ).add( canvas, "Center" );
frame.setSize( new java.awt.Dimension( 500, 500 ) );
frame.addWindowListener( new java.awt.event.WindowAdapter( ) {
    public void windowClosing( java.awt.event.WindowEvent e ) {
        System.exit( 0 );
    }
} );
frame.setVisible( true );
}
}

```

#### **5.4.5. JDBC-ODBC bridge to connect to Access database**

We store all the metaball information such as radius, material R,G,B information, shineness value, as well as R1,R2 value of the metaball in the Microsoft Access database. In this project we use JDBC-ODBC to access database. Below is the source code for this part.

```

public void connectDatabase() {
    String url = "jdbc:odbc:metaball";
    String username = "anonymous";
    String password = "guest";
    try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
}

```

```

        connection = DriverManager.getConnection(url,username,password);
    }
    catch (ClassNotFoundException cnfex){
        System.err.println("Failed to load JDBC/ODBC driver.");
        cnfex.printStackTrace();
        System.exit(1);
    }
    catch (SQLException sqllex) {
        System.err.println( "Unable to connect");
        sqllex.printStackTrace();
    }
    getTable();
}

```

#### 5.4.6. Metaball algebra function used to represent coupling

There are many versions of metaball algebra function used currently. In this project we use a potential energy function as below:

$$f_x = (\text{Seuil}[i]-1)*(\text{Seuil}[i]-1)/(\text{Seuil}[i]);$$

Seuil[i] is the distance between the metaball center and the influence point. Figure 5.3 is the effect of the metaball coupling with different colors.

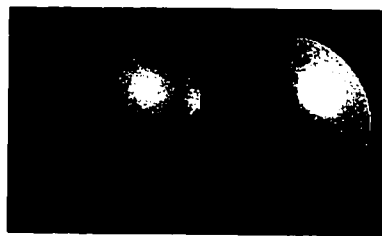


FIGURE 5.3:MEATBALL COUPLING WITHOUT ANY COLOUR INTERPOLATION

## 5.5. Public Methods

This application package includes public methods to draw different radius, shading, texture of metaballs. All the API methods are all included in the Metaball class.

Public Methods:

### **Metaball()**

Default constructor. This method will create a white shading metaball with radius 1.

### **Metaball(float radius,int r, int g, int b)**

Constructor used to create a metaball with radius and color

### **Metaball(float radius,int r, int g, int b,float x float y,float z)**

Constructor used to create a metaball with radius, color and location.

### **Setradius(float radius)**

This method sets the radius of the metaball.

### **SetColor(int r, int g, int b)**

This method sets the RGB color of the metaball, attribute r,g,b is the percentage of the red, green and blue

### ***SetMaterial(int Ambr, int Ambg, int Ambb,int Diffr, int Diffg, int Diffb ,int shininess)***

This method sets the metaball material qualities such Ambient light , Diffuse light as well as shininess, Ambient light is the light that's been scattered so much by the environment that its direction is impossible to determine because it seems to come from all directions. The diffuse light is the light that comes from one direction. The attribute shininess is the material's shininess, in the range [1.0, 128.0] with 1.0 being not shiny and 128.0 being very shiny.

**Setlocation(float x ,float , float z)**

This method sets the center of the metaball

**SetTras( float x, float y, float z)**

Sets the translation parameter of the metaball

## **Chapter 6 Conclusion**

### **6.1. Summary**

The Metaball API package was designed and implemented as part of this project providing a set of function calls that can be used to draw the metaball without the user having to know the details of metaball implementation and technology. Users call these functions in the same way as calling any Java3D API classes. The Metaball API package can be seen as the extension of the Java3D API package.

Fundamentally this package is based on the MarchingCube algorithm and Octree recursion which is used to approximately represent the isosurface of metaball algebra function

This package is written in Java and uses the Java 3D package extensively. The Java 2 platform provides Java Virtual Machine(JVM) that enables Java programs to run on any OS. Java Application Programming Interface(API) is a large collection of ready-made software components that provides many useful capabilities, such as GUI widgets.

### **6.2. Future Work**

The entire system can be used to model and visualize metaball configuration. There is considerable scope for interesting work what can be performed. We only list a few of the important work items to be addressed as soon as possible for making the package even more effective.

1. When the number of metaballs is increased, the rendering time is slow. It is because all the metaballs are within one cube. We can optimize the algorithm by

incorporating a mechanism to terminate a recursion when there are no more metaball surface parts in a cubic sub part of the Octtree structure. .

2. Texture mapping of the metaball is not very elegant. Right now we texture map the text to a transparent cube outside the metaball and not directly to the surface of metaball. This is due to a problem in the Java 3D API, because of which, there is a reflection of the text if mapped on the surface of the sphere. It can be improved in the future.
3. When the metaballs are coupled together, and the metabal colors are not the same, then color is not interpolated smoothly across the connection. The effect of color blending must be incorporated when displaying two meatballs that are coupled together.
4. The metaball coupling process is static. We can add the animation of Metaball coupling process.
5. The actual application to visualization of large software has to be studied and analyzed in depth.

## References

- [1] Blaine A. Price, Ronald M. Baecker and Ian S.Small, “ A principled Taxonomy of Software Visualization”, *Journal of Visual Languages and Computing* 4, p211-266.
- [2] Bloomenthal, Jules and Bajaj, Chanderjit “Introduction to implicit surface”, San Francisco, CA.: Morgan Kaufmann publisher, Inc., 1997
- [3] Claire Knight and Malcolm Munro “Visualizing Java Uncertainty”, Visualization Research Group, Research Institute in Software Evolution. Department of Computer Science, University of Durham, Durham, DH1 2LE, UK
- [4] Deitel Paul J., “Java How to Program”,Third Edition, Prentice Hall, 1999
- [5] Georgia Institute of Technology, “ Software Visualization” website:  
<http://www.cc.gatech.edu/gvu/softviz/SoftViz.html>
- [6] Ian Sommerville, “Software Engineering”, Fifth Edition, Addison Wesley,1995.
- [7] James Sharman, “The Marching Cubes Algorithm”, website: <http://kom.auc.dk/~zeek/kowd/mcubes/ind.html>
- [8] James Sharman, “The Marching Cubes Algorithm”, website:  
<http://www.exaflop.org/docs/marchcubes/ind.html>
- [9] Jeurgen Rilling and S. P. Mudur, Department of Computer Science, Concordia University, Canada “On the Use of Metaballs to Visually Map Source Code Structures and Analysis Results onto 3D Space”, in *Proceedings of IEEE WCRE 2002*
- [10] John Hunt and Alex McManus, “Key Java-Advanced Tips and Techniques”, Springer, 1998

- [11] John T. Stasko, "Three-Dimensional Computation Visualization", Georgia Institute of Technology Atlanta, GA 30332-0280
- [12] Jonathan I. Maletic, Jason Leigh and Andrian Marcus, "Visualizing Software in an Immersive Virtual Reality Environment" , in Proceedings of ICSE'01 Workshop on Software Visualization, Toronto, Ontario, Canada, May 12-13 2001, p49-54.
- [13] Kuroda Dycoon, "Metaball " website: <http://www.ceres.dti.ne.jp/~dycoon/program/meta/emeta.html>
- [14] Matthew Ward, " An overview of Metaballs/Blobby Objects ", website: <http://www.cs.wpi.edu/~matt/courses/cs563/talks/metaballs.html>, WPI CS Department
- [15] Neville Churcher, Lachlan Keown, Warwick Irwin "Virtual Worlds for Software Visualization", Software Visualization Group, Department of Computer Science, University of Canterbury, Private Bag 4800, Christchurch, New Zealand
- [16] Paul Bourke, "Implicit surfaces", website: <http://astronomy.swin.edu.au/~pbourke/modellig/implicitsurf/>
- [17] Paul Bourke, "Polygonising a scalar field", <http://astronomy.swin.edu.au/~pbourke/modeling/polygonise/>, May 1997
- [18] Steven P. Reiss, "A Practical Introduction to Software Design with C++", John Wiley & Sons, Inc., 1998
- [19] Sun Corporation, "Getting Started with the Java 3D API", website: <http://java.sun.com/products/java-media/3D/collateral/>



- [20] Sun Corporation, “JAVA™ Foundation Classes (JFC)”, website:  
<http://java.sun.com/products/jfc>
- [21] Tom Nuydens, “Delphi3D”, website: <http://www.delphi3d.net/articles/printarticle.php?article=metaballs.htm>
- [22] William E. Lorensen and Harvey E. Cline, “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”, Computer Graphics(Proceeding of SIGGRAPH '87), Vol. 21, No. 4, p163-169
- [23] Wim De Pauw, Richard Helm, Doug Kimelman, and John Vlissides  
“AnArchitecture for Visualizing the Behavior of Object-Oriented Systems”,  
IBM T.J.Watson Research Center, P.O. BOX 704, Yorktown Heights, NY  
10598 USA

## Appendix A: EdgeTable in Marching Cubes Algorithm

```
static int edgeTable[256]={
0x0 , 0x109, 0x203, 0x30a, 0x406, 0x50f, 0x605, 0x70c,
0x80c, 0x905, 0xa0f, 0xb06, 0xc0a, 0xd03, 0xe09, 0xf00,
0x190, 0x99 , 0x393, 0x29a, 0x596, 0x49f, 0x795, 0x69c,
0x99c, 0x895, 0xb9f, 0xa96, 0xd9a, 0xc93, 0xf99, 0xe90,
0x230, 0x339, 0x33 , 0x13a, 0x636, 0x73f, 0x435, 0x53c,
0xa3c, 0xb35, 0x83f, 0x936, 0xe3a, 0xf33, 0xc39, 0xd30,
0x3a0, 0x2a9, 0x1a3, 0xaa , 0x7a6, 0x6af, 0x5a5, 0x4ac,
0xbac, 0xaa5, 0x9af, 0x8a6, 0xfaa, 0xea3, 0xda9, 0xca0,
0x460, 0x569, 0x663, 0x76a, 0x66 , 0x16f, 0x265, 0x36c,
0xc6c, 0xd65, 0xe6f, 0xf66, 0x86a, 0x963, 0xa69, 0xb60,
0x5f0, 0x4f9, 0x7f3, 0x6fa, 0x1f6, 0xff , 0x3f5, 0x2fc,
0xdfc, 0xcf5, 0xfff, 0xef6, 0x9fa, 0x8f3, 0xbf9, 0xaf0,
0x650, 0x759, 0x453, 0x55a, 0x256, 0x35f, 0x55 , 0x15c,
0xe5c, 0xf55, 0xc5f, 0xd56, 0xa5a, 0xb53, 0x859, 0x950,
0x7c0, 0x6c9, 0x5c3, 0x4ca, 0x3c6, 0x2cf, 0x1c5, 0xcc ,
0xfcc, 0xec5, 0xdcf, 0xcc6, 0xbca, 0xac3, 0x9c9, 0x8c0,
0x8c0, 0x9c9, 0xac3, 0xbca, 0xcc6, 0xdcf, 0xec5, 0xfcc,
0xcc , 0x1c5, 0x2cf, 0x3c6, 0x4ca, 0x5c3, 0x6c9, 0x7c0,
0x950, 0x859, 0xb53, 0xa5a, 0xd56, 0xc5f, 0xf55, 0xe5c,
0x15c, 0x55 , 0x35f, 0x256, 0x55a, 0x453, 0x759, 0x650,
0xaf0, 0xbf9, 0x8f3, 0x9fa, 0xef6, 0xfff, 0xcf5, 0xdfc,
0x2fc, 0x3f5, 0xff , 0x1f6, 0x6fa, 0x7f3, 0x4f9, 0x5f0,
0xb60, 0xa69, 0x963, 0x86a, 0xf66, 0xe6f, 0xd65, 0xc6c,
0x36c, 0x265, 0x16f, 0x66 , 0x76a, 0x663, 0x569, 0x460,
0xca0, 0xda9, 0xea3, 0xfaa, 0x8a6, 0x9af, 0xaa5, 0xbac,
0x4ac, 0x5a5, 0x6af, 0x7a6, 0xaa , 0x1a3, 0x2a9, 0x3a0,
0xd30, 0xc39, 0xf33, 0xe3a, 0x936, 0x83f, 0xb35, 0xa3c,
0x53c, 0x435, 0x73f, 0x636, 0x13a, 0x33 , 0x339, 0x230,
0xe90, 0xf99, 0xc93, 0xd9a, 0xa96, 0xb9f, 0x895, 0x99c,
0x69c, 0x795, 0x49f, 0x596, 0x29a, 0x393, 0x99 , 0x190,
0xf00, 0xe09, 0xd03, 0xc0a, 0xb06, 0xa0f, 0x905, 0x80c,
0x70c, 0x605, 0x50f, 0x406, 0x30a, 0x203, 0x109, 0x0  };
```

## Appendix B: Metaball API Package Class Hierarchy

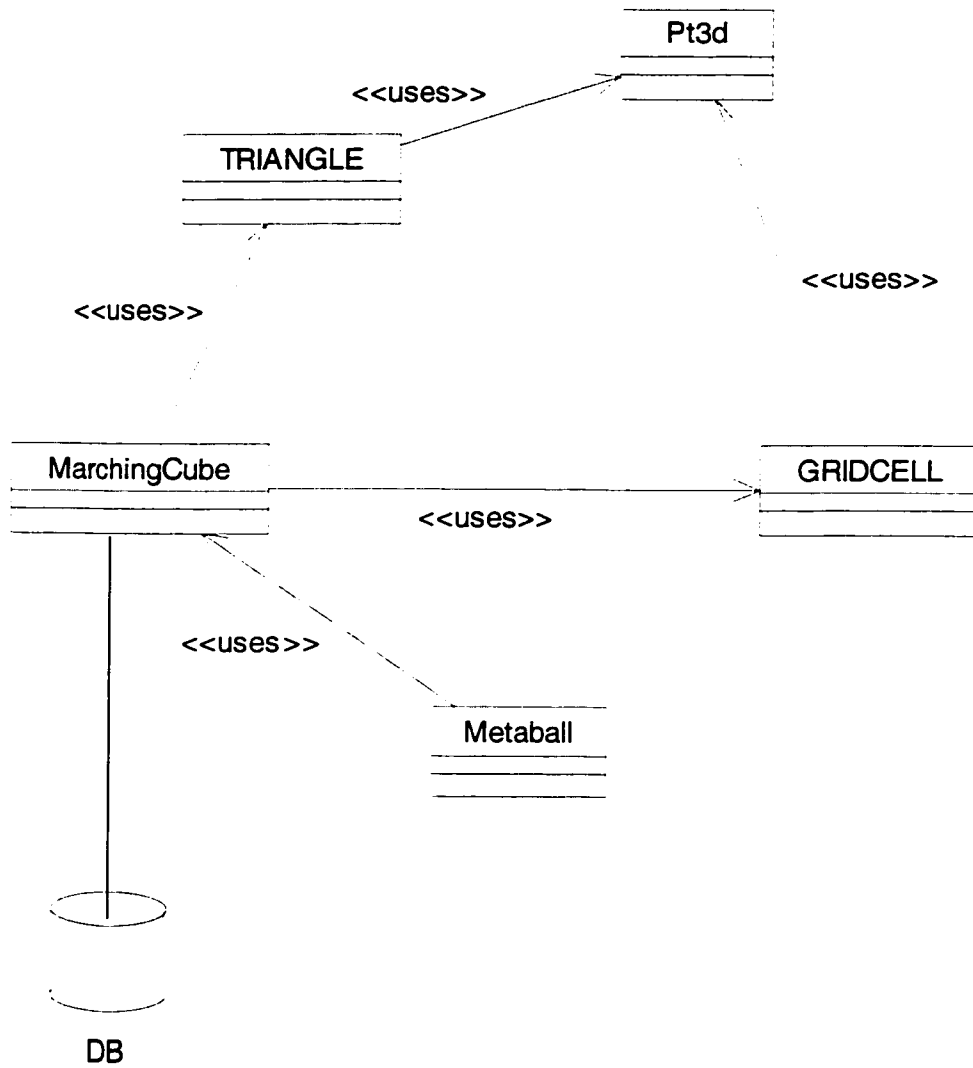


FIGURE B.1 METABALL PACKAGE HIERARCHY

## Appendix C: Snap Shots using Metaball API Package

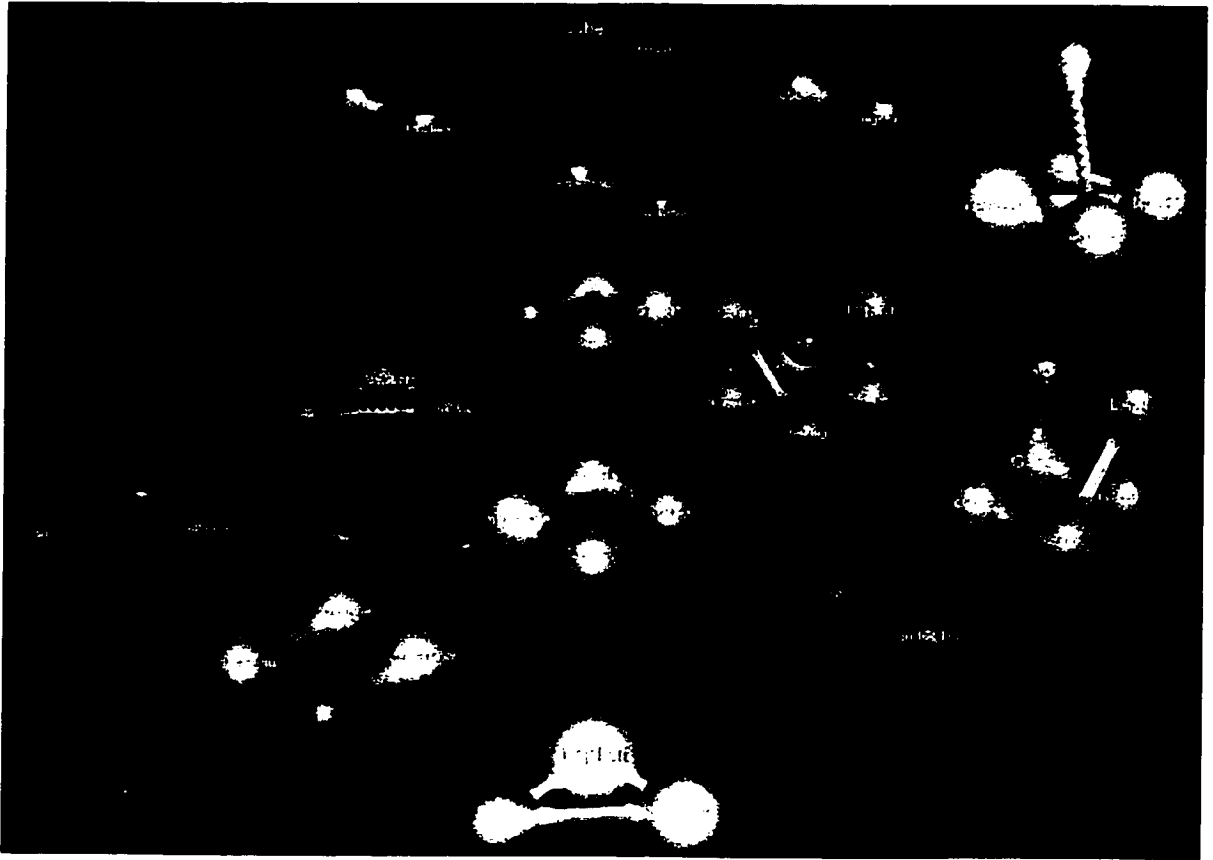


FIGURE C.1: SNAPSHOT OF A METABALL CONFIGURATION OF SOFTWARE ENTITIES AND THEIR INTERRELATIONSHPS

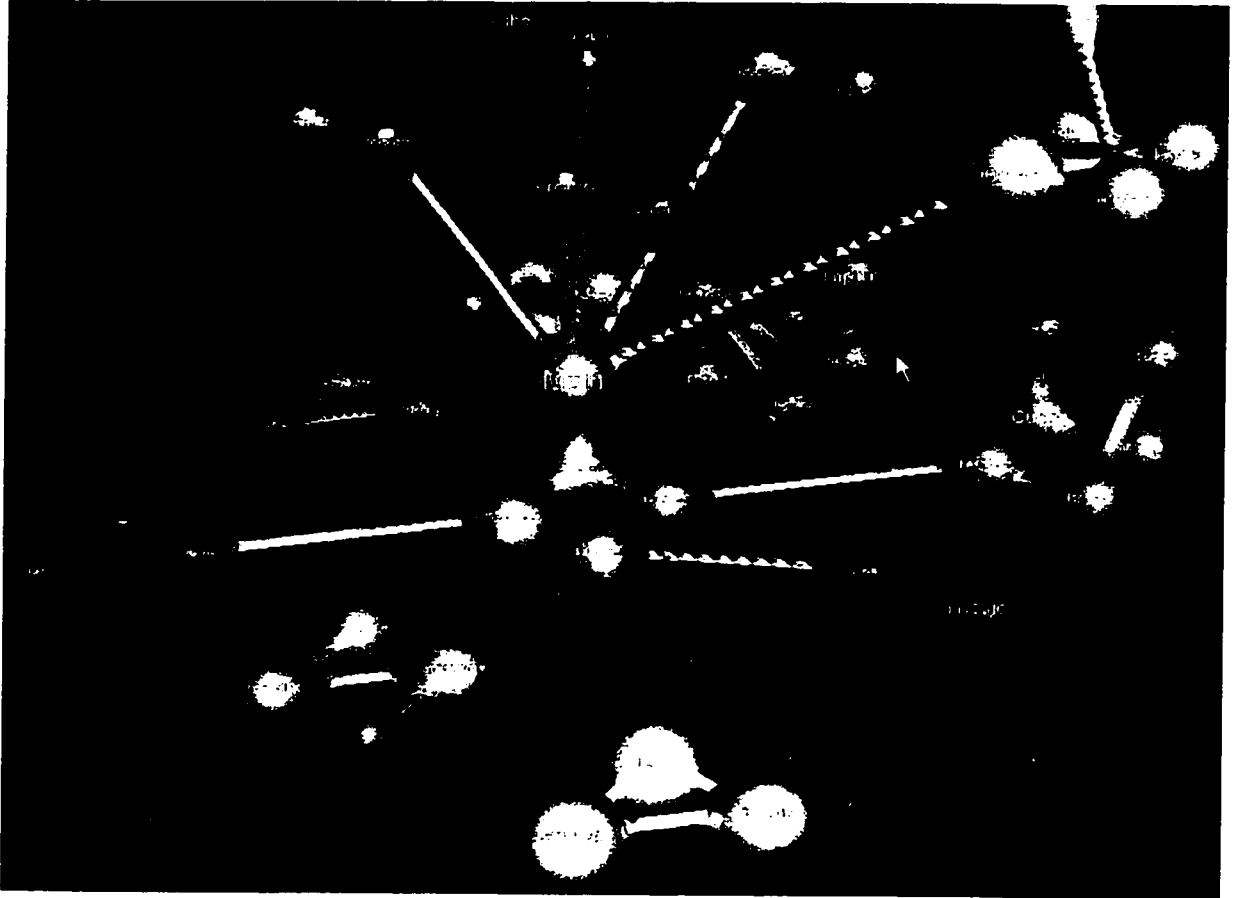


FIGURE C.2: ANOTHER SNAPSHOT OF A METABALL CONFIGURATION IN THE PROCESS OF SOFTWARE VISUALIZATION

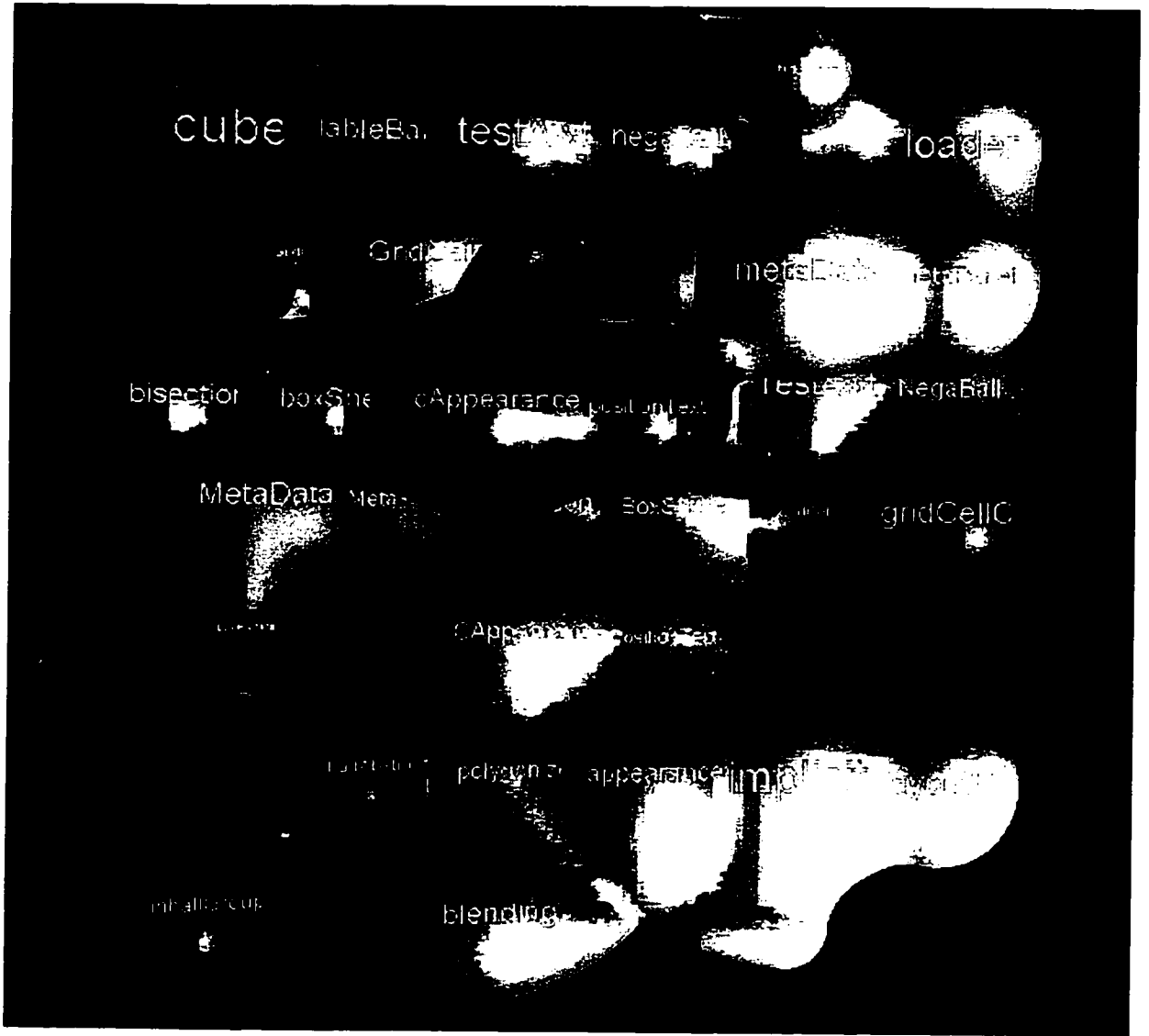


FIGURE C.3: HEAVILY COUPLED METABALL CONFIGURATION IN SOFTWARE VISUALIZATION