SEMI-AUTOMATIC AND INTERACTIVE VISUALIZATION

OF QUANTUM DOT NANO-STRUCTURES

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Xinying Ye

December, 2007
SEMI-AUTOMATIC AND INTERACTIVE VISUALIZATION
OF QUANTUM DOT NANO-STRUCTURES

Xinying Ye

Thesis

Approved:

Advisor
Yingcai Xiao

Co-Advisor
Ernian Pan

Faculty Reader
Zhong-Hui Duan

Department Chair
Wolfgang Pelz

Accepted:

Dean of the College
Ronald F. Levant

Dean of the Graduate School
George R. Newkome

Date
ABSTRACT

This thesis presents a visual analytics system for semi-automatic and interactive visualization of quantum dot (QD) nano-structures. It presents the rationale, design, implementation and testing of the system. QDs are artificial atoms where the movements of the real atoms in them are confined. Scientists have used Monte Carlo simulations to predict the growth of self-organized QDs in strained semiconductors. The visual analytics system presented in this thesis is a set of post-simulation visual analysis tools. It consists of four modules: Input, Clustering, Identification and Examination. It provides the following tools for the scientists to visualize and analyze the formation and the structures of the QDs.

• Timed animation: shows the formation and growth of the QD islands.
• Clustering: internally identifies QD islands as spatial clusters of atoms.
• Color-coding: renders each QD island in a distinctive color so that its structure can be easily visualized.
• Auto-identification: automatically identifies the islands of particular interest (IOPI);
• Auto-zooming: allows the scientists to have close examinations of the QD islands of particular interests.
• Interactive visualization: allows the scientists to view the overall picture of the QD distribution or an IOPI from different vantage points.
• Triangulation: forms an elevation surface from the QDs so that their height profile can be visualized.

• Cutting: constructs a cross section view of the triangulated elevation surface from any given cutting plane so that the scientists can further analyze the height profile at any cross sections.

Initial tests have shown that the tools are easy to use and the system is valuable to the scientists for post-simulation analysis of the nano-structures of the self-organized QDs.

Further work can be done to improve the performance and the user-friendliness of the system. For examples, Quad-trees can be incorporated into the spatial clustering algorithms to speed the clustering process by only considering the atoms in the same and neighboring quads when making distance calculations. Labeled axes can be added to the height profile display to make it easier for scientists to analyze the profile more accurately.
ACKNOWLEDGEMENT

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. I want to thank the research group of Computer Modeling and Simulation in the Civil Engineering Department for providing the topic, necessary search material and simulation data for this thesis.

I am deeply indebted to my advisor Dr. Yingcai Xiao of the Computer Science Department whose suggestions and encouragements helped me throughout the research project.

I would like to thank my co-advisor, Dr. Ernian Pan of the Civil Engineering Department, for confirming the topic, and for teaching me the knowledge of quantum dots which forms the foundation of this thesis work.

My “Advanced Algorithm” class teacher, Dr. Zhong-Hui Duan, from the Computer Science Department supported my algorithm work in this research. I want to thank her for her support and her valuable hints.

Finally, I would like to give my special thanks to my friend, Mingkun Sun, whose patient support enabled me to complete this work.
TABLE OF CONTENTS

I. INTRODUCTION ........................................................................................................... 1
  1.1 Visual Analytics.................................................................................................... 1
  1.2 QD Simulation ...................................................................................................... 1
  1.3 Motivation............................................................................................................. 4
  1.4 Thesis Outline ....................................................................................................... 6

II. DESIGN OF THE POST-SIMULATION VISUAL ANALYSIS SYSTEM .............. 7
  2.1 System Architecture.............................................................................................. 7
  2.2 Input Module......................................................................................................... 8
  2.3 Clustering Module .............................................................................................. 11
  2.4 Identification Module.......................................................................................... 11
  2.5 Examination Module........................................................................................... 14

III. ALGORITHMS AND IMPLEMENTATION ......................................................... 15
  3.1 Clustering Algorithms.......................................................................................... 15
    3.1.1 Merge Clustering (MC)............................................................................ 15
    3.1.2 Edmonds-Karp Based Clustering Algorithm ........................................... 21
    3.1.3 Edmonds-Karp Based Algorithm V.S. MC Algorithm............................ 22
  3.2 Identification Algorithm..................................................................................... 23
    3.2.1 Color Mapping......................................................................................... 23
    3.2.2 Cluster Identifications.............................................................................. 26
3.3 Examination Algorithms................................................................. 27
  3.3.1 Growth Animation ................................................................. 27
  3.3.2 Auto Stepped Zooming ........................................................... 27
  3.3.3 Triangulation .......................................................................... 30
3.4 Implementation ............................................................................... 32

IV. RESULTS ......................................................................................... 33
  4.1 Animation of QD Island Growth ................................................ 33
  4.2 Automatic Island Identification .................................................. 36
  4.3 Interactive Island Identification .................................................. 36
  4.4 Automatic Stepped Zooming ....................................................... 39
  4.5 Island Height Analysis ............................................................... 41

V. CONCLUSIONS AND FUTURE WORK .......................................... 42

REFERENCES ...................................................................................... 44
LIST OF TABLES

Table 2.1. Simulation output format................................................................. 10
Table 3.1. Cluster and link numbers in case 1..................................................... 18
Table 3.2. Link numbers after merging clusters 0 and 1.................................... 19
Table 3.3. The updated cLink after merging cluster 2 and cluster 0.................... 19
Table 3.4. The new cLink converts the old cluster numbers to the new ones....... 21
Table 3.5. Space and time complexities of the clustering algorithms.................. 23
LIST OF FIGURES

Figure 1.1. Simulated QDs: (a) atoms without strain, (b) strained QD islands. ....... 2

Figure 1.2. Illustration of up or down jump of an edge atom during 3D QD self-assembly .................................................................. 3

Figure 1.3. Illustration of QD island height profiles (b,c) on given cutting sections during 3D QD self-assembly (a). .................................................. 4

Figure 2.1. Modular architecture of the visual analytic system ......................... 8
Figure 2.2. Post-process in 2D ........................................................................ 10
Figure 2.3. The visualization pipeline for rendering QDs ..................................... 12

Figure 3.1. Pseudo-code of MC algorithm for finding the cluster for an atom. ...... 16
Figure 3.2. Initial cluster distribution in case 1. ................................................. 17
Figure 3.3. Cluster distribution before (a) and after (b) merging cluster 1 and cluster 0. ................................................................................. 18
Figure 3.4. Cluster distribution before (a) and after (b) merging cluster 2 and cluster 0. ................................................................................. 19
Figure 3.5. Pseudo-code of MC algorithm for renumbering the clusters. ......... 20
Figure 3.6. The distribution of clusters after merging and renumbering .......... 21
Figure 3.7. Pseudo-code for assigning a unique color to each cluster and assigning the highlight red color to the IOPI. ................................. 24
Figure 3.8. The animation of a bouncing ball consists of 6 motion pictures. ........ 27
Figure 3.9. Pseudo-code for finding the center of the IOPI and gradually moving to it. ................................................................................. 29
Figure 3.10. Pseudo-code for zooming into the focal point and allowing interactions. ................................................................................. 29
Figure 3.11. A Delaunay triangulation. (a) Top view. (b) Side view. .......................... 31
Figure 3.12. Delaunay triangulation rendering pipeline. ........................................ 32
Figure 4.1. QD growth and island distribution at time step 2................................. 34
Figure 4.2. QD growth and island distribution at time step 4............................... 34
Figure 4.3. QD growth and island distribution at time step 8............................... 35
Figure 4.4. QD growth and island distribution at time step 10............................ 35
Figure 4.5. Overall view depicting the distribution of the QD islands identified with different colors. The IOPI is in red. ......................................................... 36
Figure 4.6. SII: all atoms are being selected......................................................... 37
Figure 4.7. SII: some atoms are being selected..................................................... 37
Figure 4.8. Rotated view of the selection the box widget...................................... 38
Figure 4.9. Average size and height of selected QD islands............................... 39
Figure 4.10. Auto-zoomed view of the identified island with the largest number of QDs. .......................................................... 40
Figure 4.11. Rotated view showing layered structures of QD islands.................... 40
Figure 4.12. (a) Delaunay triangulation without cutting. (b) Delaunay triangulation with cutting. .................................................. 41
Figure 4.13. Height profile at an arbitrary cutting position (example: y = -x). ....... 41
CHAPTER I

INTRODUCTION

1.1 Visual Analytics

Visual analytics is the science of analytical reasoning facilitated by interactive visual interfaces [1]. It has a wide range of applications in Home Land Security, wireless network protection, digital library query, productive reading, collaborative sense-making, linguistics algorithm study, large dataset exploration, and numerous other application areas [2-10]. Due to the important applications of visual analytics, the National Visualization and Analytics Center (NVAC) were created in May 2004 [2]. NVAC’s mission is to provide strategic leadership and coordination for the development of visual analytics technology and tools. One of the focus areas identified by NVAC is the development of visual representations and interaction techniques to allow users to see, explore, and understand large amounts of information [1].

1.2 QD Simulation

The purpose of this project is to develop visual analytics tools for the study of quantum dot (QD) nano-structures. QDs are artificial atoms where the movements of their electrons are confined. Due to this quantum confinement, nanostructures made of QDs can have unique electronic and optoelectronic properties for device applications [11, 12].
Monte Carlo simulations have been performed to simulate the growth of self-organized QDs in strained semiconductors [13-15]. In this kind of simulations, atoms are deposited onto a substrate (Figure 1.1) and are induced to move by their interactions with the substrate. Eventually, the atoms reach an equilibrium state, forming a nano-structure of QD islands (Figure 1.1) [13, 14].

![Figure 1.1. Simulated QDs: (a) atoms without strain, (b) strained QD islands.](image)

Zhu, Pan and Chung [15] used a three-dimensional kinetic Monte Carlo (KMC) model to simulate the growth of self-assembled QD islands. The KMC model is a multiscale model that includes two aspects of the simulation: the long-range strain energy contribution from a fast continuum Green’s function calculation and an up/down ratio describing the relative probability for atoms to jump out of the plane of the surface during the growth process.

In the 3D QD self-assembly growth system, the edge atoms always try to move to the upper layers, which forms 3D islands, to decrease the free energy. The material properties and growth conditions control the probability for an atom to jump up and down [26]. Figure 1.2 shows that activity of keeping balance between surface energy increase and strain energy decrease [15].
Zhu, Pan and Chung concluded from their 3D KMC model experiments:

1) The flux rate, the deposition and interruption time effect the islands’ shape and ordering. Lower flux rate and longer growth time make a better island distribution.

2) The relationship between the island height and the up/down ratio shows that the island height increases with the increase of up/down ratio when the ratio is between 1-20. When the value of up/down ratio is beyond 20, the height of the islands approaches to a constant value [15].

Further research is being conducted to study how other growth parameters, such as temperature, surface coverage, can affect the structure of the islands.

Scientists also conduct physical experiments to grow QDs in strained semiconductors [27]. Figure 1.3(a) is a photo of such a semiconductor of self-organized QDs. Scientists are interested in the uniformity of the QD distribution and their internal structures. Height profiles, as illustrated in Figure 1.3(b) and Figure 1.3(c), are very helpful to visualize the uniformity of the QD structures.

The purpose of the Monte Carlo simulations is to find optimal grow parameters to guide the physical experiments. In order to find optimal grow parameters, the simulation data needs to be visualized and analyzed. Those grow parameters which yield uniform...
QD distribution and uniform QD structures are optimal and are candidates for physical experiments.

1.3 Motivation

In the process of study how growth parameters affect the structure of the islands, scientists found great needs of visual tools to help them to analysis the structures of the QD islands. The lower the flux rate is and the longer the growth time is the more time steps are need to simulate the formation of the QD islands and the more data will be generated by the simulation. The scientists need to visualize, based on the simulation data, how the islands are formed, how they are distributed spatially after the formation and
how they change through time. They need to identify islands of special interests (for example, islands with the largest number of atoms) to examine their structures, such as their shape and height profile. They not only need to visualize the simulation data but also need to interact with the data and visually analyze the data.

Visual analytics is perfect to serve such need. Unlike information and scientific visualization, which concentrate on displaying the data, visual analytics fills the gap between the machine and human mind. Visual analytics focuses more on human interaction with the display. We are not satisfied with just presenting the data to the users, we also want to automatically extract important information from massive amount of data and allow the users to interact with them.

In this project, a visual analytics system is developed with visual representation and interaction techniques to allow users to see, explore, and understand large amounts of simulated QD data. More specifically, in order to help the scientists to analyze the simulation data, this project aims to create a set of post-simulation visual analysis tools that:

- allows the scientists to animate the formation of the QD islands;
- allows them to view the overall pictures of the QD distribution;
- automatically identifies the islands of particular interest;
- automatically zooms into the identified islands;
- allows the scientists to interactively explore the identified islands.
1.4 Thesis Outline

This thesis reports the design, implementation and the preliminary result of the visual analytics system which helps scientists to visually analyze the nano-structures of self-organized QDs. Chapter II discusses design of the overall system. Chapter III provides a detailed analysis of the algorithms used in the system. Chapter IV presents some test results. Chapter V contains conclusions and a list of future work.
2.1 System Architecture

According to the analysis in section 1.3, it is desired to develop a powerful visual analytics system for scientists to see, explore, and understand large amounts of simulated QD data. The system should support the current needs of the scientists by providing specific features like automatic identification of QD islands of particular interest and automatic zoom into the identified islands, etc. The system should have a flexible architecture such that it can be easily upgraded to accommodate future needs of the scientists. To accomplish both, a modular approach is adopted for this project. The modular approach allows us to implement and experiment with different algorithms for each module without too much impact on the other modules. Figure 2.1 shows the modular architecture of the visual analysis system for the QD nano-structure study. The Input module reads the QD distribution data generated by the simulation program. The Clustering module creates the internal presentation of the QD islands as clusters of spatial points (atoms). The internal cluster representation makes it possible for the Identification module to automatically and visually identify the islands of particular interests to the user. The user should be able to dynamically specify the selection criteria for the islands of particular interest. For each identified island, the Examination module automatically
zooms into it and gives the user a close look at it. The Examination module also includes utilities for scientists to view the islands at different vantage point and query the properties of the islands.

![Modular architecture of the visual analytic system](image)

**Figure 2.1. Modular architecture of the visual analytic system.**

2.2 Input Module

In order for the Input module to read the QD distribution data generated by the simulation program, we need to understand the simulation data. There are a total of 10 parameters required for running the simulation program, they are:

- **Material and growth direction**: isotropic and anisotropic materials are considered in the simulation program. For anisotropic material, three different growth directions can be chosen.
- **Coverage (0-1)**: total coverage of the atoms.
- **Lx, Ly**: x dimension and y dimension.
- **Flux rate (Ml/s)**: flux rate to be applied to deposit the atoms.
- **Temperature (K)**: simulation temperature.
- **Simulation time (t)**: time used for atom diffusing.
- **Bonding energy to surface & bonding energy to neighbor**: they are assumed to be
constant in the simulation.

The contents of the output files of the simulation program are as follows:

- **Initial atom coordinates.** These are the coordinates of the atoms immediately after their random deposition. The first column is the atom number; the second column is the x-coordinate of the atom; the third column is the y-coordinate of the atom.

- **Coordinates which change within required simulation time.** They are exported to the output file every 10 minutes of simulation time. In this part, the format is a little bit different from the original coordinates. The first column is the x-coordinates of the atoms; the second column is the y-coordinates; the third column indicates the strain pattern.

- **For each output step, total simulation time, total moving steps are revealed at the beginning of each output step.** For example,

  “Simulation Time = 10
  Total Steps =2818297
  Atom Coordinates
  COORDINATE and STRAIN PATTERN
  ……”

- **Final coordinates:** In this section, the final coordinates of the atoms at required simulation time are presented. Similarly, the first column is the x-coordinates of the atoms; the second column is the y-coordinates; the third column indicates the strain pattern. See Table 2.1.
Table 2.1. Simulation output format.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>52</td>
<td>2</td>
</tr>
<tr>
<td>44</td>
<td>81</td>
<td>1</td>
</tr>
<tr>
<td>56</td>
<td>66</td>
<td>1</td>
</tr>
<tr>
<td>65</td>
<td>53</td>
<td>2</td>
</tr>
<tr>
<td>75</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>84</td>
<td>2</td>
</tr>
</tbody>
</table>

One can use some simple tools, such as excel or Matlab to depict the locations of atoms according to the coordinates in the output files. See Figure 2.2. [15]

Figure 2.2. Post-process in 2D.
2.3 Clustering Module

The clustering module groups the atoms into clusters such that each cluster represents a visible QD island. The atoms in each cluster are spatially close to each other than the atoms in other clusters. Therefore the clustering algorithms in this project should be distance based and they should group the atoms into clusters based on the distance between them. Furthermore, the scientists are more interested in the formation and distribution of the QD islands on the 2D substrate. This requires the clustering algorithms to operate in 2D. On the other hand, 3D display is needed for scientists to study the height profiles of the clustered DQ islands, therefore, we need to cluster in 2D and render in 3D.

Visual analytics requires speedy interactions for scientists to operate the visualization system. Clustering is the most time consuming module of system. The challenge is that how can one find a good algorithm with ideal space and time complexity. Since the number of atoms in this application is around 10000, which is relatively large, the efficiency of the clustering algorithms is critical. The design and implementation of the algorithms will be discussed further in Chapter III.

2.4 Identification Module

To study the properties of individual QD island, the scientists need tools to identify it. Three identification methods, namely, color-coding, IOPI and SII, are implemented in this project.
To distinguish QD islands from one another, each QD island and the atoms in it need to be colored with a distinctive color so that the atoms can be visually identified from atoms in the other islands and the structure of the QD island can be easily visualized. Figure 2.3 depicts the visualization pipeline for rendering QDs and their atoms in this project. The basic steps are as following.

Step 1. Acquire the simulation data.

Step 2. Create the visualization dataset and store the data into a 3D point array.

Step 3. Map each atom as a rendering actor.

Step 4. Create atom actors as spheres.

Step 5. Display the atom actors.

**Figure 2.3. The visualization pipeline for rendering QDs.**
Knowing how to render the QD islands and their atoms is just the starting point of the examination module. According to the discussion in section 1.3, the islands’ shape and ordering are greatly affected by the flux rate, deposition and interruption times. With different flux rate and growth time, the distribution of atoms and the shapes of islands will keep changing until a set of relatively stable conditions is achieved. Simulation results also show that island heights increase sharply with the increase of up/down ratio when the ratio is between a certain ranges of values, otherwise the height of islands come towards to a stable value [12]. Therefore, island size and height are the two main factors which the scientists most interested. The term “Islands of Particular Interest” (IOPI) is created to define the specific islands selected according to the criteria of special interests to the scientists. In this thesis, the two criteria are island size and island height, accordingly, IOPI means the largest island or the highest island.

To assists the scientists analyzing the nano-structures of the IOPIs, the Identification module need to provide tools to automatically find the IOPIs and to guide the attention of the scientists to the automatically identified IOPIs.

Scientists may also identify other QD island to examine in detail. “Self-identified Islands (SIIs)” allow scientists to identify and analyze any QD island by mouse clicking on the screen. The user interactive feature makes island identification more flexible and complements the auto-identification provided by IOPI identification. The implementation of the two different identification features will be further discussed in Chapter III.
2.5 Examination Module

The main goal of this module is to provide tools for the scientists to visually examine the QD nano-structures from all different angles. The following functions are provided to scientists for in-depth exploration of islands in this project.

1) Timed animation: shows the growth of the islands from the initial deposition of the atoms onto the substrate to the time of equilibrium.

2) Zooming: allows the scientists to have close examinations of the QD islands of particular interests,

3) Interactive viewing: allows the scientists to view the identified QD islands from different vantage points,

4) Triangulation and cutting: constructs a continuous elevation (height) surface from the QD distribution data and cut through the surface to reveal the height profiles of QD islands.

The implementation of the listed features will be discussed in the next chapter.
CHAPTER III

ALGORITHMS AND IMPLEMENTATION

The visual analytics system in this project is implemented in C++ using the Visualization Toolkit [18]. The algorithms used in the system and their implementation code are discussed in this chapter.

3.1 Clustering Algorithms

The clustering module groups the atoms into clusters such that each cluster internally represents a visible QD island. Two clustering algorithms were used in this project: the Merge Clustering algorithm and the Edmonds-Karp max-flow based algorithm. As discussed in section 2.3 that the cluster algorithms should operate in 2D even though the display is in 3D.

3.1.1 Merge Clustering (MC)

Merge Clustering (MC) is originated from the concept of region-growing [16]. This algorithm checks each atom, the atom will be assigned to an existing cluster if it is within a tolerant distance to the cluster, and it will be given a new cluster number if it is not close to any existing cluster. An atom could join multiple clusters together if it is within the tolerant distance to more than one existing clusters. The name, Merge Clustering, is
due to the fact that clusters could be merged together to become a larger cluster during the clustering process. The tolerant distance is specified by the user and can be adjusted.

The corresponding pseudo-code of the MC algorithm is shown in Figure 3.1.

```plaintext
// For each atom i located at coordinate (cx, xy), find clusters it is in.
found = 0;
for (j=0; j<i; j++) // for each atom already clustered
{
    //distance between points
    cx0 = objPointData[j].xLoc;
cy0 = objPointData[j].yLoc;
cz0 = objPointData[j].zLoc;
dis=(cx-cx0)*(cx-cx0)+(cy-cy0)*(cy-cy0);
    if (dis < tol*tol) // found a cluster within range
    {
        found++;
        if(found == 1) {
            // Found the first cluster the atom is in.
            // Use its cluster number for the atom.
            objPointData[i].cNum = objPointData[j].cNum;
cNum = objPointData[j].cNum;
        } else if (cNum != objPointData[j].cNum) {
            // This atom links to another cluster,
            // the two clusters should be merged.
            // Find the smaller cluster number.
            if(cNum < objPointData[j].cNum) {
                cNum1 = cNum;
cNum2 = objPointData[j].cNum;
            } else {
                cNum2 = cNum;
cNum1 = objPointData[j].cNum;
            }
        }
    }
}
// Merge all atoms in the clusters into one and
// use the smaller cluster number.
for (k = 0; k <= i; k++) {
    if(objPointData[k].cNum == cNum2)
        objPointData[k].cNum = cNum1;
}
```

Figure 3.1. Pseudo-code of MC algorithm for finding the cluster for an atom.
cNum = cNum1;
cLink[cNum2] = cNum1;
} // found > 1
} // Found clusters atom i is in.
} // for each clustered atom j

// If the atom is not in any existing cluster, create a new cluster.
if(found == 0)
{
cLink[mcNum] = -1; // A new cluster links to no other clusters.
objPointData[i].cNum = mcNum;
mCNum++;
}

Figure 3.1. Pseudo-code of MC algorithm for finding the cluster for an atom (continued)

To demonstrate how the algorithm works, let’s consider case 1, where we have already created 7 clusters after processing 14 atoms. The distribution of clusters at this moment is shown in Figure 3.2.

Figure 3.2. Initial cluster distribution in case 1.

Table 3.1 shows the array of cluster numbers created in case 1. The link number, recorded in cLink, represents where a cluster links to after merging with another cluster. -1 indicates no link at this moment and no merge has occurred.
Table 3.1. Cluster and link numbers in case 1.

<table>
<thead>
<tr>
<th>cNum</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>cLink</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Now atom 15 is being processed. It is found firstly connected to cluster 1 as shown in Figure 3.3(a). The grey dot stands for the new atom and it was grouped into cluster 1. But at the same time, the distance between the grey dot and an orange dot in cluster 0 is smaller than the distance tolerance. This indicates that the two clusters are connected and they should be merged into one. The algorithm renumbers all atoms in cluster 1 to cluster 0 as shown in Figure 3.1(b). Note the smaller cluster number is reused during a merge. The reason will be explained shortly. After the merge, the cLink for cluster 1 is changed to 0 as shown in Table 3.2 to indicate it has been merged to cluster 0.

![Figure 3.3](image-url)

Figure 3.3. Cluster distribution before (a) and after (b) merging cluster 1 and cluster 0.
Table 3.2. Link numbers after merging clusters 0 and 1.

<table>
<thead>
<tr>
<th>cNum</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>cLink</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

In case a new atom connects to a cluster already merged into another cluster, the algorithm still works as demonstrated in Figure 3.4. The grey dot in the figure represent atom 16. It was found to be connected to cluster 2 first, hence was numbered 2 in Figure 3.4(a). Afterward, it was found to be connected to a green dot which was in cluster 1 but later merged into cluster 0. Since all atoms in cluster 1 have been renumbered to cluster 0, the new atom, atom 16, was actually found to be connected to cluster 0 not cluster 1. Figure 3.4 shows the process where atom 16 caused the merge of cluster 2 into cluster 0 with cluster 1 already merged in. Table 3.3 shows the new link numbers of the clusters.

Figure 3.4. Cluster distribution before (a) and after (b) merging cluster 2 and cluster 0.

Table 3.3. The updated cLink after merging cluster 2 and cluster 0.

<table>
<thead>
<tr>
<th>cNum</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>cLink</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
It is noticeable in the algorithm and the figures that \( mcNum \) is the largest cluster number being used not the total number of clusters found. There is a gap between them. For example, in Figure 3.4, there are only 5 clusters but the largest cluster number used is 6. For the color coding algorithm to be presented in section 3.2.1, we need the cluster numbers to be contiguous so that a distinctive color can be assigned to each cluster. To resolve the issue, the MC algorithm reuses the lower cluster numbers after merging and renumbers the clusters and their atoms as shown Figure 3.5.

```c
// Compress cluster numbers and renumber all atoms
cNum = 0;
for(i=0; i<mcNum; i++) {
    if(cLink[i] == -1) { // Found a cluster, assign the lowest available cluster number to it.
        cLink[i] = cNum;
        cNum++;
    } else { // This cluster has been merged to cluster cLink[i], use its cluster number.
        cLink[i] = cLink[cLink[i]];
    }
}

// Change cluster numbers for all atoms to the new numbers.
for(i=0; i<totPnts; i++) {
    cNum = objPointData[i].cNum;
    objPointData[i].cNum = cLink[cNum];
}
```

Figure 3.5. Pseudo-code of MC algorithm for renumbering the clusters.

The new \( cLink \) actually maps old cluster numbers \( i \) into new cluster numbers \( cLink[i] \). In the case presented above, Table 3.4 shows the old \( cNum \) and new (New \( cLink \)) for every processed cluster. Figure 3.6 shows the distribution of new clusters after merging and renumbering.
Table 3.4. The new cLink converts the old cluster numbers to the new ones.

<table>
<thead>
<tr>
<th>cNum</th>
<th>cLink</th>
<th>New cLink</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.6. The distribution of clusters after merging and renumbering.

3.1.2 Edmonds-Karp Based Clustering Algorithm

The second clustering algorithm used in this project is based on the Edmonds-Karp max-flow algorithm [17]. This algorithm defines a distance matrix measuring the distance between the atoms and creates a spatial similarity matrix with the similarity defined as the inverse of the distance between two atoms. The algorithm then groups the atoms based on their spatial similarity.

The Edmonds-Karp max-flow based algorithm is implemented in the following four steps:
• Read atom locations.
• Define the distance matrix.
• Compute the similarity matrix
• Use the maximum flow algorithms to find clusters.

For the reasons presented below this algorithm is not usable for clustering a large number of atoms, hence not usable for this application. Consequently, the details of the algorithm will not be presented in this thesis.

3.1.3 Edmonds-Karp Based Algorithm V.S. MC Algorithm

In order to compare the two clustering algorithms, we conduct some complexity analyses here. Assuming N is the total number of atoms to be clustered. Edmonds-Karp based algorithm has a space complexity of \(O(N^2)\) and a time complexity of \(O(NE^2)\), where E is the number of edges in the graph [17]. The \(O(N^2)\) space is needed to store the distance matrix and the similarity matrix which are NxN. Since all the distance between all the atoms are not infinite, there are edges between every pair of them, therefore, \(E = N*N\). The time complexity of the Edmonds-Karp based algorithm in our application is then \(O(N^5)\), which makes it take too long to process large number of atoms. On the other hand, the space complexity for the Merge Clustering is \(O(N)\) which is basically the space needed to store the coordinates of the atoms. The time of MC algorithm is spent mostly in the three nested loops of i, j and k as shown in Figure 3.1. Since each of the indexes runs from 0 to possibly N-1, the time complexity of the algorithm is then \(O(N^3)\). Further more, the loop indexed by k is only used for reassigning a cluster number to the processed
atoms. If we treat the time for reassigning numbers are negligible, the time complexity of MC becomes $O(N^2)$.

Table 3.5 summarizes the space and time complexities of the two algorithms. Since $N$, the number of atoms, in our application is around 10,000, the Edmonds-Karp based algorithm is clearly at disadvantage. Compare to the MC algorithm, it uses 10,000 times more space and takes 100,000,000 times longer to cluster 10,000 atoms. Obviously, it is not a good candidate for our application.

Table 3.5. Space and time complexities of the clustering algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Space Complexity</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edmonds-Karp</td>
<td>$O(N^2)$</td>
<td>$O(N^5)$</td>
</tr>
<tr>
<td>MC Algorithm</td>
<td>$O(N)$</td>
<td>$O(N^3)$</td>
</tr>
</tbody>
</table>

3.2 Identification Algorithm

As outlined in section 2.4, the visual analytic system implemented in this project provides three identification methods: color-coding, IOPI and SII. The implementations of the three are presented below.

3.2.1 Color Mapping

In this project, the first visual information that should be provided to the scientists is the display of the QD islands and the atoms belong to them. The atoms are displayed as 3D spheres. The islands they form are rendered in various colors, one unique color for each island. The color mapping (CM) algorithm implemented in this project is shown in Figure 3.7.
// Define maxcolor, the total number of colors to be used,  
// It is the same as the total number of clusters.  
maxcolor = mcNum;

// Define color table to make sure no cluster number is out of range.  
r3 = paw (maxcolor,0.33333333333333333333);  
m1 = r3;  
m2 = r3*r3;  
m3 = r3 * r3 * r3;

// Comput the RGB color for each atom j based on its cluster number.  
rgb[0] = 1.0 – ((objPointData[j].cNum - 1) % m1) / m1;  
rgb[1] = 1.0 – ((objPointData[j].cNum - 1) % m2) / m2;  
rgb[2] = 1.0 – ((objPointData[j].cNum - 1) % m3) / m3;  
AtomSphere->GetProperty()->SetColor (rgb[0],rgb[1],rgb[2]);

// Render the spheres in the cluster with the largest number of atoms red.  
// iopi is the cluster number for the cluster of the largest size.  
if (objPointData[j].cNum==iopi) AtomSphere->GetProperty()->SetColor (1,0,0);

Figure 3.7. Pseudo-code for assigning a unique color to each cluster and assigning the  
highlight red color to the IOPI.

Because the goal of island coloring is to give a unique color to each island for  
identification, obviously, the number of colors we need is at least equal to or more than  
the number of the islands what we have. Meanwhile, the number of the islands generated  
is based on the result from the clustering algorithm. With the max number of colors we  
need, we can find out what are the values should be assigned to the three elements of a  
color (R, G, B). The following equation is used to get a cardinal number for calculating  
the number of values we need to assign to R, G, B separately.

\[ r_3 = \sqrt[3]{\text{max color}} \]  

(1)

Where, \( r_3 \) is the cardinal number, \( \text{max color} \) is the maximum number of colors we need.

The number of values needed for R, G, B channels are calculated using the following  
set of equations:
\[ m_i = r_3 \]
\[ m_2 = r_3^2 \]
\[ m_3 = r_3^3 \]  

(2)

\( m_1, m_2, m_3 \) are the maximum number of values we can assign to R, G, B channels individually.

The following three equations are used to calculate the intensity values of R, G, and B for atom \( j \) and put the values in an array where \( \text{rgb}[0] \) stores red, \( \text{rgb}[1] \) stores green and \( \text{rgb}[2] \) stores blue.

\[
\text{rgb}[0] = 1.0 - \frac{((\text{objPointData}[j].cNum - 1) \mod m_1)}{m_1} \quad (3)
\]

\[
\text{rgb}[1] = 1.0 - \frac{((\text{objPointData}[j].cNum - 1) \mod m_2)}{m_2} \quad (4)
\]

\[
\text{rgb}[2] = 1.0 - \frac{((\text{objPointData}[j].cNum - 1) \mod m_3)}{m_3} \quad (5)
\]

In the equation, \( \text{objPointData}[j].cNum \) means the cluster number of the atom, \( m_1, m_2, m_3 \) separately means the number of intensity values in red, green, and blue.

As we know, the RGB system represents colors based on their prime components R, G and B [18]. The combine color will be different if any of its prime components is different. When \( c\text{Num} \) changes from 1 to \( \text{maxcolor} (m\text{Num}) \), equations 2-5 tells us each value in the R channel is used \( m_2 \times m_3 \) times, each value in the G channel is used \( m_3 \) times and each value in the B channel is used only once. This is because \( m_3 \) is the same as \( \text{maxcolor} (m\text{Num}) \). Since each value in the B channel is only used once, the colors generated by equations 2-5 will never be the same. For example, if there were 27 islands need to be colored, we get \( r_3 = 3 \) from equation (1). Then we get from equation 2:

\[
m_1 = 3
\]

\[
m_2 = 9
\]

\[
m_3 = 27
\]
When a color is defined for an atom, it is assigned to the related actor by the VTK function VtkActor->GetProperty()->SetColor() method.

3.2.2. Cluster Identifications

Color coding is actually the first visual identification clue for the users. It gives the overall picture of the distribution of the QD islands on the substrate.

The second identification algorithm is IOPI. An IOPI is identified based on a selection criterion specified by scientists. Actually, scientists of this visual analytics system give the definition of the IOPI as one of the two parameters: island size or island height. Once a definition criterion is specified, the system automatically finds the related cluster and color code it in red as shown in Figure 3.7. The system then brings the attention of the users to the IOPI by automatically zooming into the IOPI. Auto zooming is discussed in the next section.

SIIs are the islands selected by the scientist themselves. This way the scientist can visually identify and analyze any island interactively. To implement this function, an interactive box widget is created by the following method VTK method:

```cpp
vtkBoxWidget *boxWidget = vtkBoxWidget::New()
```

The widget is displayed as a box along with QD islands. The size and location of the box can be interactively changed by the user to isolated different clusters. The average height and the number of atoms in the box are computed and displayed to the user.
3.3 Examination Algorithms

Three Five algorithms are included in this section. They are Growth Animation, Auto Stepped Zooming, and Triangulation.

3.3.1 Growth Animation

Animation is the fast and sequential display of a sequence of images or objects positions. It is an illusion of movement which can generally be presented in two ways: motion picture or video although [21]. See Figure 3.8.

![Figure 3.8](image_url)

Figure 3.8. The animation of a bouncing ball consists of 6 motion pictures.

In this project, the growth animation of QDs greatly helps scientists to visualize the QD self-assembly process. I used the motion picture method to display the movement of QDs at different interruption time. The algorithm reads in the position data of the atoms in the QDs one interruption time at a time and displays the atoms at their locations on the screen after each read. The visual effect is that the atoms are moving as time passes, and they organize themselves to form the DQ islands.

3.3.2 Auto Stepped Zooming

In this project, the viewing window is set in parallel projection to avoid the foreshortening distortion of a perspective projection [18]. Automatically zooming into an
IOPI is achieved through resetting the focal point to the center of the related cluster and reducing the parallel scale of the camera. The Auto Stepped Zooming (ASZ) implemented in this project includes the following four main steps.

- Finding out the focal point on the screen.
- Move to the focal point.
- Zoom into the focal point a small step at a time.
- Enable interaction with the IOPI.

First of all, we should find out the center location of the IOPI on the window’s screen. The center coordinates of an IOPI are computed as the average coordinates of all atoms in it. The coordinates are the world coordinates and need to be converted into screen coordinates by multiple appropriate scaling factors (sx, sy).

To make it easier for the users to follow the change of focal point on the screen, we move the focal point gradually from its current location (cx, cy) to the new location. This is done in a pre-specified number of steps. The pseudo code for implementing this part of ASZ is listed in Figure 3.9.

```cpp
// Create the rendering window and its renderer.
vtkRenderWindow *renWin = vtkRenderWindow::New();
vtkRenderer *ren = vtkRenderer::New();
renWin->AddRenderer(ren);

// The world coordinates of the center of IOPI has been computed and stored
// cxm = the average x or the center x of the IOPI
cxm = the average x or the center x of the IOPI
// cym = the average y or the center y of the IOPI
cym = the average y or the center y of the IOPI

// Find out the screen coordinates of the focal point.
fx = cxm * sx; // relative x to the screen
fy = cym * sy; // relative y to the screen

// Get the camera from the renderer.
vtkCamera* camera = ren->GetActiveCamera();
```
// Move focal point gradually.
for(i=0; i<max_step; i++)
{
    camera->SetFocalPoint(cx + fx*i/(float)max_step,
                           cy + fy*i/(float)max_step, 0);
    renWin->Render();
}

Figure 3.9. Pseudo-code for finding the center of the IOPI and gradually moving to it.

Similarly, the ASZ algorithm gradually zooms into the focal point after the focal point is set to the center of the IOPI. This is accomplished by gradually reset the parallel viewing scale from its current value (cs) to the desired closer value. The related pseudo-code is listed in Figure 3.10 along with the code for setting up interactivity, which is done using a vtkRenderWindowInteractor.

// Get the camera from the renderer.
vtkCamera* camera = ren->GetActiveCamera();

// Move focal point gradually.
for(i=0; i<max_step; i++)
{
    camera->SetParallelScale(cs*(float)(max_step-i)/(float)(max_step);
    renWin->Render();
}

// Create the interactor and start the interaction
vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
iren->SetRenderWindow(renWin);
iren->Initialize();
iren->Start();

Figure 3.10. Pseudo-code for zooming into the focal point and allowing interactions.
3.3.3 Triangulation

When scientists examine the nano-structures of the QD islands, they need to view the height profiles of the structure as shown in Figure 1.3(b) and Figure 1.3(c). In order to construct the height profiles, we need first to create an elevation surface representing the height of the deposited atoms on the substrate. We use a triangular method to build such a surface.

Triangulation [18] is a set of nonintersecting triangles sharing common vertices and edges. This technique builds a topological surface from a set of points. The topological surface is a collection of triangles which do not intersect with one and another. In this project, Delaunay triangulation [18] is used. It has a specific property that the circum-circle of any triangle contains no other points. Figure 3.11 shows an example of a triangulation from top and side views. One can see the height variations of the atoms which are presented as the vertices of the triangulation.
The rendering pipeline of the triangulation is shown in Figure 3.12. After the creation of a triangulated elevation surface, we can cut through the surface using a vtkPlane to reveal the cross section height profile at any location and any orientation.
3.4 Implementation

Implementation of the project was done in Microsoft Visual C++, using the Visualization Toolkit (VTK) [18]. VTK is a powerful object-oriented interactive graphics and visualization programming library. A C++ class library and some interpreted interface layers like Java, Tcl/Tk, and Python were included in VTK. It supports a wide variety of visualization algorithms from basic modeling knowledge including scalar, vector, tensor, texture, and volumetric methods to advanced modeling techniques such as implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation [18].
CHAPTER IV

RESULTS

The visual analysis system designed and implemented for this project has been tested. This chapter presents some of the test results.

4.1 Animation of QD Island Growth

Figures 4.1-4.4 are snapshots of animations of QD island growth from a sequence of test simulation datasets. Each figure displays island distribution at a specific interruption time. This animation uses 11 sets of the simulation data at 10 different interruption times. Through the animation, one can see the self-organizing nature of the atoms through the formation of the QD islands.
After 4 time steps, we can see most of random QDs have found an island they belong to.
At the end of time step 12, the atoms reach an equilibrium state, forming a collection of QD islands.
4.2 Automatic Island Identification

Figure 4.5 demonstrates how each QD island is coded with a unique color. Color coding makes it easier to visualize the distribution of the atoms and the structures of the QD islands. In addition, the IOPI is highlighted in red for further examination.

4.3 Interactive Island Identification

Figure 4.6-4.8 shows how the user can interactively identify QD islands for analysis. A box widget is used to allow the user to select any number of QD islands to study. The users can choose any QD islands at any location by controlling the size and location of box widget.

Figure 4.5. Overall view depicting the distribution of the QD islands identified with different colors. The IOPI is in red.
Figure 4.6. SII: all atoms are being selected.

Figure 4.7. SII: some atoms are being selected.
Figure 4.8. Rotated view of the selection the box widget.

To assist the users to analysis the SIIs, the following information of the SIIs are displayed: (a) the total number of atoms being selected and (b) the average height of atoms being selected. Figure 4.9 shows the display screen for a sequence of interactive selections by manipulating the box widget.
4.4 Automatic Stepped Zooming

After automatic identification of an IOPI, the system automatically zooms into the IOPI as shown in Figure 4.10. The user is then allowed to interactively manipulate the viewing vantage point and examine the IOPI in different angles. For example, one can see the layered structure of the islands from a rotated view shown in Figure 4.11.

Figure 4.9. Average size and height of selected QD islands.
Figure 4.10. Auto-zoomed view of the identified island with the largest number of QDs.

Figure 4.11. Rotated view showing layered structures of QD islands.
4.5 Island Height Analysis

Figure 4.12 shows the Delaunay triangulation of a test dataset with and without cutting. With double-sided cutting, we can reveal the height profile at any given cutting position and any orientation as shown in Figure 4.13.

Figure 4.12. (a) Delaunay triangulation without cutting. (b) Delaunay triangulation with cutting.

Figure 4.13. Height profile at an arbitrary cutting position (example: $y = -x$).
CHAPTER V

CONCLUSIONS AND FUTURE WORK

In this thesis, I have designed and implemented a visual analytics system for semi-automatic and interactive visualization of QD nano-structures. The system consists of four modules: Input, Clustering, Identification and Examination and many useful tools. The tools include timed-animation of the formation and growth of the QD islands, clustering of the QDs as spatial islands, color-coding of the islands for easy visualization, automatic identification of IOPIs, ASZ for close examinations of IOPIs, interactive analysis of SII's, interactive visualization of the overall picture of the QD distribution, interactive visualization of an IOPI or SII from different vantage points, triangulation of the QDs to reveal their height profiles, cross section display of the triangulated elevation surface for further analyzing the height profile at particular locations and orientations.

Tests have been conducted to check the performance of the system. The Edmonds-Karp-based clustering is too time consuming and is not suitable for applications of large number of atoms. Other than that the system has been proven to be very valuable to the scientists to analyze their simulation data on self-organizing QDs in strained semiconductors.

Future work can be done to improve the performance and the user-friendliness of the system. The speed of the clustering algorithms can be improved by using quad-trees.
This allows the spatial clustering algorithms to consider only the atoms in the same or neighboring quads when making distance calculations, therefore to save processing time. Labels need to be added to the displays so that scientists can more accurately visualize the height and distribution profiles of QDs.
REFERENCES


[22] VTK: http://www.vtk.org/


[26] Poloczek, P. Sek, G., Misiewicz, J., Loffler, A., Reithmaier, J. P. and Forchel,