MORPHOLOGICAL EVOLUTION OF STRAINED ISOTROPIC THIN SOLID FILM ON RIGID SUBSTRATE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SANAM HADDADIAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
METALLURGICAL AND MATERIALS ENGINEERING

JULY 2014

Approval of the thesis:

MORPHOLOGICAL EVOLUTION OF STRAINED ISOTROPIC THIN SOLID FILM ON RIGID SUBSTRATE

submitted by **SANAM HADDADIAN** in partial fulfillment of the requirements for the degree of **Master of Science in Metallurgical and Materials Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Hakan Gür _____
Head of Department, **Metallurgical and Materials Eng.**

Prof. Dr. Kadri Aydınol _____
Supervisor, **Metallurgical and Materials Eng. Dept., METU**

Prof. Dr. Tarık Oğurtanı _____
Co-Supervisor, **Metallurgical and Materials Eng. Dept., METU**

Examining Committee Members:

Prof. Dr. Turgut Baştuğ _____
Material Science and Nanotechnology Eng. Dept., TOBB ETU

Prof. Dr. Kadri Aydınol _____
Metallurgical and Materials Eng. Dept., METU

Prof. Dr. Tarık Oğurtanı _____
Metallurgical and Materials Eng. Dept., METU

Assist. Prof. Dr. Ersin Emre Ören _____
Biomedical Eng. Dept., TOBB ETU

Assoc. Prof. Dr. Arcan Dericioğlu _____
Metallurgical and Materials Eng. Dept., METU

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Sanam Haddadian

Signature :

ABSTRACT

MORPHOLOGICAL EVOLUTION OF STRAINED ISOTROPIC THIN SOLID FILM ON RIGID SUBSTRATE

Sanam Haddadian
M.Sc., Department of Metallurgical and Materials Engineering
Supervisor: Prof. Dr. Mehmet Kadri AYDINOL
Co-Supervisor: Prof. Dr. Tarik OGURTANI

July 2014, 134 pages

In quantum dots (QD), the excitons are spatially confined and their energy spectrum, which controls many physical properties of interest, can be adjusted over a wide range by tuning composition, density, size, lattice strain and morphology. The formation of QDs joined by a thin flat wetting layer, known as the Stranski-Krastanow (SK) morphology, is a general growth mode observed in many epitaxially-strained thin solid films. These features make semiconductor QDs attractive for the design and fabrication of novel electronic, magnetic and photonic devices. The success of this endeavor has mainly been enabled by research, leading to reliable means for estimating forces in small material systems and by establishing frameworks, in which the integrity and/or functionality of the systems is satisfied.

The material failure continues to be a main technology-limiting barrier and thus, the subject of capillary-driven morphological evolution of surfaces and interfaces; especially under the action of applied force fields e.g., electrostatic and thermo-mechanical, is still a challenging materials problem.

Here we demonstrate the effects of strain relaxation on morphological evolution of QDs and occurrence of wetting layer. Our study based on continuum level dynamical simulations will be presented for the spontaneous evolution of an isotropic isolated thin solid droplet on a rigid substrate under various stress fields. The simulations showed that there is a threshold value for the stress level under which the formation of isolated islands observed; whereas at higher stress levels we observed the formation of SK-type islands connected with a very thin wetting layer. *Supported by TUBITAK grant no 111T343 and TUBA GEBIP*.

Keywords: Thin films, thin film growth modes, quantum dots, surfaces and interfaces, numerical modeling

ÖZ

SERT ALTLIK ÜZERİNDE GERGİN İZOTROP İNCE KATI FİLMİN MORFOLOJİK EVRİMİ

Sanam Haddadian
Yüksek lisans, Metalurji ve Malzeme Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Mehmet Kadri AYDINOL
Ortak Tez Yöneticisi: Prof. Dr. Tarik OĞURTANI

Temmuz 2014, 134 sayfa

Kuantum noktalarda (QD), eksitonlar mekansal olarak sınırlanmıştır. Bu malzemelerde, birçok fiziksel özelliği kontrol eden enerji spektrumu, kompozisyon, yoğunluk, boyut, kafes gerginlik ve morfolojide değişikliklerle geniş bir aralıkta ayarlanabilir. İnce düz bir ıslatma tabakasıyla birbirine bağlı olan kuantum noktaların oluşumu, Stranski-Krastanow (SK) büyüme modu olarak adlandırılmış ve bir çok epitaksiyel gergin ince katı filmde gözlemlenen genel bir durumdur. Bu özellikler, yarı iletken quantum noktaları yeni elektronik, manyetik ve fotonik cihazların tasarımı ve imalatı için cazip kılmaktadır. Bu çalışmaların başarısı, ağırlıklı olarak bilimsel araştırmalara ve bu araştırmaların ortaya çıkardığı nano boyutlu malzeme sistemlerindeki kuvvetleri tahmin ederek bu sistemlerin bütünlüğü ve / veya işlevleri sağlamak amacıyla çerçeve oluşturulmasıyla sağlanmıştır.

Malzeme kusurları, teknolojinin ana sınırlayıcı engeli olmaya devam etmektedir. Yüzeyler ve arayüzeylerin kapilar odaklı morfolojik evrimi, özellikle uygulanan kuvvet alanlarının etkisi altında, hala çözülmesi gereken önemli bir malzeme sorunudur.

Bu araştırmada, sis-temdeki gerginliğin gevşemesinin quantum noktaların morfolojik evrimi ve ıslatma tabakasının ortaya çıkması üzerine etkilerini incelenmiştir. Süreklilik seviyedeki dinamik simülasyonlara dayalı bu çalışmamız, katı bir altlık üzerinde izole edilmiş bir izotropik ince katı madde damlacığının çeşitli gerilim alanları altında kendiliğinden evrimini göstermektedir. Simülasyon bulgularının gösterdiği sonuca göre stres düzeyinin bir eşik değerin altında izole adaların oluşumu gözlenmiştir. Daha yüksek gerilme seviyelerinde ise ince bir ıslatma tabaka ile bağlantılı SK-tipi adaların oluşumu gözlenmiştir. Bu çalışma, 11T343 nolu TÜBİTAK 1001 projesi ve Dr. Ören'in TÜBA GEBİP projesi tarafından desteklenmiştir.

Anahtar Kelimeler: İnce filmler, ince film büyütme modları, kuantum noktalar, yüzeyler ve arayüzler, sayısal modelleme

To My Family

ACKNOWLEDGMENTS

I wish to express my deepest gratitude to my supervisor Prof. Dr. Kadri Aydinol and to my Co-supervisors Prof. Dr. Tarik Ogurtani and Assist. Prof. Dr. Ersin Emre Oren for their guidance, advice, criticism, encouragements and insight throughout the research.

I also would like to thank to supports by the Turkish Scientific and Technological Research Council, TUBITAK through a research Grant No. 111T343 and Turkish Academy of Sciences TUBA GEBIP Grant to Assist. Prof. Dr. Ersin Emre Oren.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

LITERATURE SURVEY

1.1. Overview

Quantum dots are nano-size semiconductors in which holes and electrons are confined in all three spatial dimensions. Due to some similarity, sometimes, they are called artificial atoms, however, the size is much bigger (1-100 nm for QD versus 0.1 nm for atoms). These nanostructures represent unique optical and electrical properties, which are due to their small size, and are different in character to those of the corresponding bulk material. In these particles, the addition or subtraction of an electron changes its properties.

To know what is special with QDs, we need some background on semiconductors. In bulk semiconductors, the presence of many atoms causes splitting of the electronic energy levels, giving continuous energy bands separated by a Band gap. Band gap is the forbidden zone for the electron existence. The portion of energy levels above the band gap is called conduction band and below the band gap is called valance band (Figure 1.1). Electrons gain the additional energy from external sources such as electromagnetic radiation, electricity or heat and this, results in electron jump to an unstable higher energy level. In an atom, the excited electron loses its absorbed energy when falls back down to its original ground state. This energy is released in the form of light (Figure 1.2). In bulk materials, due to existing of large number of atoms, energy levels are close to each other; this makes

continuous energy bands. In a quantum dot, relatively few atoms are present, which leads to discrete and quantized energy levels more like those of an atom. This is the reason of why quantum dots are called “artificial atoms”.

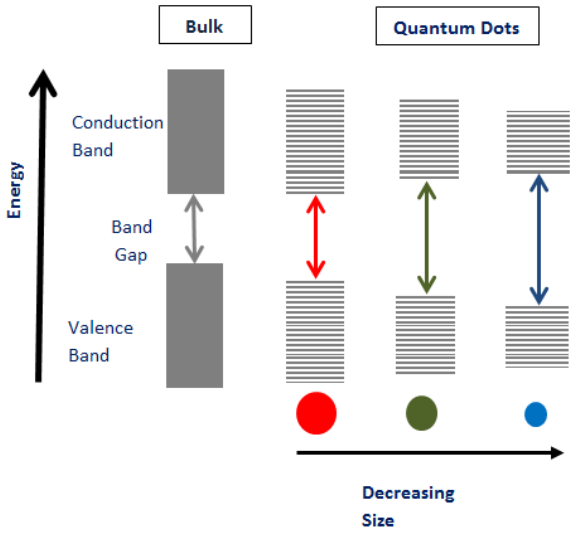


Figure 1.1: The effect of QD size on the band gap zone

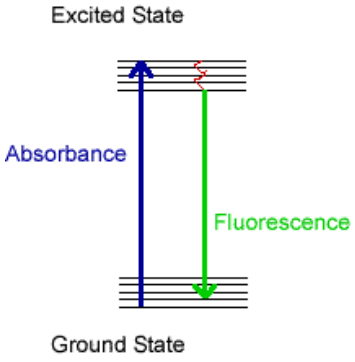


Figure 1.2: The fluorescence effect of an excited electron

Traditional semiconductors have many limitations. Their band gaps and energy levels are not easily changed and their emission frequencies cannot be easily manipulated as desired, which leads to expensive procedure for optical and electrical quality adjustment.

Quantum dot's band gap changes with its size or composition and thus, the addition or removal of just a few atoms could affect the band gap energy. Nowadays, these nanostructure materials have become an interesting issue both in commercial and scientific areas due to their special functions. However, there are a lot of unknown about them.

The efficiency of QD based devices is extremely dependent on the positioning, density and size of quantum dots. Therefore, in order to produce desirable self-organized QDs, there should be great scientific information about QD nucleation, growth and morphological change mechanisms. The information that is required for understanding the mechanism will be helpful for developing new QD formation techniques and designing of new QD based devices.

1.2. The effect of size on the electron confinement regime

The combination of the electron in the conduction band and the hole in the valance band, which are held together by the electrostatic Coulomb force, is called exciton. The quantum effect occurs if the dimension of the nano-crystal is smaller compared to the natural length scale of the exciton (a_B), which we call it the Bohr radius of the particle.

$$a_B = \frac{a_0 \varepsilon}{m^*/m_e} \tag{1.1}$$

where

a_0 is the Bohr radius of the hydrogen atom

ε is the size dependent dielectric constant

m^* is the mass of the particle

m_e is the rest mass of electron

And Bohr Radius of the hydrogen atom is

$$a_0 = \frac{4\pi\varepsilon_0\hbar^2}{m_e e^2} = \frac{\hbar}{m_e c\alpha} = 0.529\,\mathring{A} \tag{1.2}$$

where

ε_0 is the permittivity of free space

$\hbar$ is the reduced plank constant

e is the electron charge

c is the speed of light in vacuum

α is the fine structure constant

For a nano-crystal, we calculate three individual Bohr radii as a_e, a_h and a_{exc} for electron, hole and exciton successively that each can be evaluated using the equation 1.1. In the case of exciton, the reduced mass of the electron-hole pair is used. In bulk

semi-conductors, the electron can move freely in all three directions. When the length of material is reduced to the same order as the exciton radius, i.e. to a few nanometers, quantum confinement effect occurs and the exciton properties are modified. Depending on the dimension of the confinement, three types of confined structures appears: two-dimensional quantum well (QW), one-dimensional quantum wire (QR) and zero dimensional quantum dot (QD). If the size of a nano-crystal is smaller than a_e, a_h and a_{exc} $(a < a_e, a_h, a_{exc})$, this is called 'strong confinement regime'. If a is larger than both $a_e$and a_h but smaller than a_{exc} $(a_e, a_h < a < a_{exc})$, only the center of the mass motion is confined and it is called 'weak confinement regime' and finally if a is between $a_e$and a_h $(a_e < a < a_h)$, it is said to be in 'intermediate confinement regime'. Hence, if the size of the quantum dot is small enough that quantum confinement dominates (typically smaller than 10 nm), the electronic and optical properties change and the fluorescent wavelength alters relevant to the size of the nano-particle [1].

As discussed earlier, in bulk materials, electrons in conduction band are free to move in all three spatial directions. Fabricating quantum well, quantum wire or quantum dot could be done by encircling the material with a larger band gap material, which leads to confining the electron either, in one, two or three dimensions respectively.

The density of states of quantum dots are discrete and hence they behave like real atoms unlike to the bulk material, where the density of states is continuous (Figure 1.3).

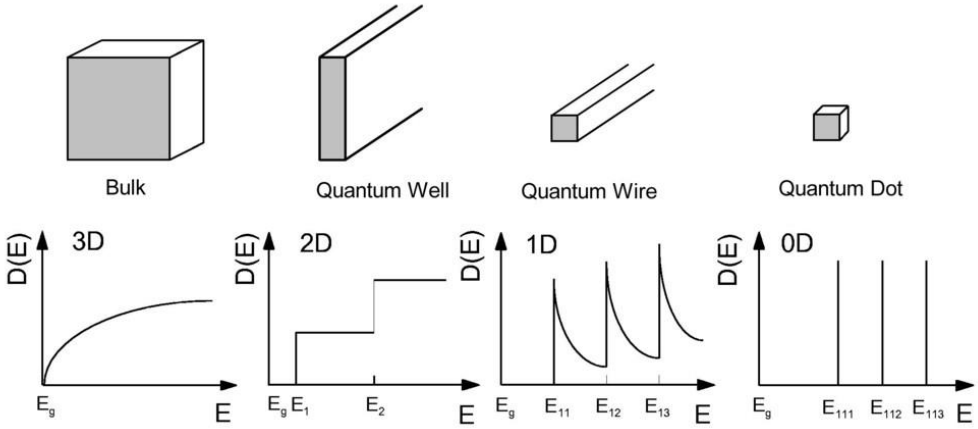


Figure 1.3: The density of states for bulk material, quantum well, quantum wire and quantum dot

1.3. Quantum dots applications

QDs with their special characteristics and electronic properties have become an important candidate for development of many medical, electronic, photonic and magnetic devices [2-4].

The thin wetting layer between QDs, which is grown within SK growth mode (this mode will be explained later), provides a path for electrons to move freely between islands. This transmission mechanism of SK mode make it suitable for electronic and optoelectronic applications.

Some examples of this wide application area are photovoltaic devices that transform the solar energy into electrical energy, Quantum computers, single electron transistors, laser fabrication, etc. [5-7].

Some particular advantages of single quantum dots are:

- Durability
- High efficiency
- Wide spectral range
- Compatible with chip-technology
- Electrical Pumping
- Strong interactions

1.4. Quantum dot production methods

Generally, there are two different approaches in the case of fabrication of quantum dots, “top- down” and “bottom-up” approaches. The “top- down” approach is the method of shaping the desired nanostructures by removing the extra material from the bulk, where this method is costly and time consuming specially for the production of nanostructures with large surface area. On the contrary, “bottom-up” approach starts from atomic level and can produce nanostructures with new chemical bonds [8].

There are several methods to produce quantum dots for different applications:

- ✓ Colloidal Synthesis
- ✓ Lithography
- ✓ Epitaxy:
  - Patterned Growth
  - Self-Organized Growth

1.4.1. Colloidal synthesis

The solution based synthetic chemistry for fabrication of QD is a powerful approach to control the size and composition of nano-crystals. Typically, colloidal nano-materials are synthesized by reacting inorganic salts or organometallic compounds that follow consecutive stages: growth of the preferred nuclei, isolation of particles reaching the desired size and post preparative treatments. This method is more flexible in fabrication comparing to other methods like Atmospheric pressure chemical vapor deposition (APCVD), which needs complex equipment and show less productivity.

Colloidal nanostructures is produced in the form of nano-crystals, nano-rods, cubes, prisms, which are suitable for biological and engineering applications such as light emitting devices (LED s), photo detectors and biological imaging agents [8, 9].

In order to improve the stability and application of colloidal QDs, they are often coated with a layer of organic passivation ligands (shell) with wider band gap called core-shell model to isolate the exciton from non-radiative relaxation via surface states that leads to wave length tuning and lifetime increasing [10].

1.4.2. Lithography

This is a top-down method, where quantum dots can be formed from two dimensional quantum wells. Photolithography, scanning probe lithography, electron beam lithography and nano-imprinting are among the most common methods for producing nano-crystals with precise shape and size used for electronic applications.

The nano-imprint method is typically suitable for flat surfaces because the mold mostly is made with rigid material. Haixiong Ge *et al*. fabricated a mold combining a rigid polymer containing silicon as a patterning template on a flexible polydimethylsiloxane polymer to allow the nano-scale imprinting on the curved substrate, which can imprint features smaller than 30 nm [11].

Vikas Nandwana *et al.* utilized direct patterning lithography by using the functional material as a negative resist. They claim that the final configuration have the same optical properties as the un-patterned film. The fluorescence intensities and lifetime were also unchanged [12].

1.4.3. Epitaxy

Semiconductor quantum dots can be fabricated by various epitaxial techniques such as molecular beam epitaxy (MBE) or Metalorganic vapor phase epitaxy (MOVPE), in which semiconductor compounds with smaller band gaps are grown on a surface of a host material with larger band gap.

If the thickness of deposited material exceeds a critical value, the system refrains to bear the extending stress, which results in the generation of coherently strained islands with a wetting layer in between. This energy favorable process is known as Stranski-Krastanov growth mode. These islands can be transformed subsequently to make quantum dots.

The self-assembled island nucleation process and the position of quantum dots cannot be controlled precisely. Since deterministic positioning of QDs is required for many applications, several techniques have been under investigation in this

respect to improve the functionality of the epitaxial quantum dots. The most advanced approach is to make holes on the substrate using lithographic methods and growth of islands on the substrate. The holes alter the chemical potential of the surface and increase the local growth rate of quantum dots [13].

1.5. Epitaxial growth modes

Epitaxial growth is a process of growing of a crystal on an underlying crystalline surface, in which the deposited crystal is oriented by the lattice structure of the substrate. When the substance of the substrate differs from the growing crystal, the process is called hetero-epitaxy and if the substances are similar, it is called homo-epitaxy.

The word “epitaxy” derived from the Greek word “epi” meaning “on” and “taxis” meaning “arrangement”, was introduced by Louis Royer, in 1928 to define this type of growth occurring in nature or imitated in laboratories and to separate epitaxial growth from non-crystalline and amorphous growth.

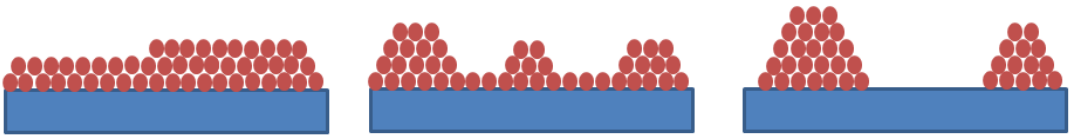
The morphology of the film depends on a number of factors, including the deposition rates of the species, the surface temperature, the surface material, and its crystallographic orientation [14-16].

It is generally accepted that, there are three possible modes of film surface morphology that can appear through epitaxial growth process. These are referred to as Frank-Van der Merwe (FM) morphology, Volmer-Weber (VW) morphology and Stranski-Krastanov (SK) morphology.

There are also another two types of growth modes known as columnar growth mode (CG mode) and step flow mode (SF), which are not in our field of interest.

In Frank-Van der Merwe (FM) mode, the atoms are strongly bounded to the substrate than to each other. A new layer nucleation begins only after the completion of the underneath layer. FM growth mode forms continuously and it can spread growth steps over macroscopic distances. A Volmer-Weber (VW) growth mode initially consists of a large number of surface nuclei, which grow into islands of condensed phase. This happens when atoms of deposit are more strongly bounded to each other than to the substrate.

The layer plus island or Stranski- Krastanov (SK) mode is considered as an intermediate between the FM and VW modes and it is caused due to the significant lattice mismatch between the film and the substrate. This lattice mismatch generates an internal strain, which happens due to increasing the elastic energy caused by increasing the layer thickness. The first deposited layer is smooth (FM growth mode) but if the thickness of the layer exceeds a critical thickness, the system needs to release the energy and subsequent island nucleation rapidly takes place (VW growth mode) [17, 18], where the stress is released by the generation of misfit dislocations. However, there is another state of Stranski-Krastanov morphology, where the energy barrier for formation of coherent islands (islands without dislocations) is very small compared to the energy need for dislocation nucleation [19, 20].



Layer-by-Layer Stranski-Krastanov Island Growth

Figure 1.4: Three different epitaxial growth modes

Epitaxy occurs in such a way that the total energy of the system consisting of substrate-crystal phases becomes minimal. The growth mode of the system also depends on interfacial free energy and lattice mismatch.

In lattice matched systems there is a relation between interfacial energy γ_{fs}, epitaxial layer surface energy γ_f and surface energy γ_s as $\gamma_f + \gamma_{fs} < \gamma_s$, which leads to formation of Frank-van der Merwe growth mode. Alternatively, if there is $\gamma_f + \gamma_{fs} > \gamma_s$ then Volmer-Weber growth mode is observed. In a system with small interfacial energy but large lattice mismatch, the system initially begins with FM mode but as layers become thicker, the system prefers to reduce its large strain energy. This leads to formation of isolated thick islands. Controlling the morphology during the hetero-epitaxy process requires understanding the atomistic mechanisms [19, 21].

The characteristic features of strain induced 2D (layer by layer) to 3D (layer + island) transformation during evolution of islands depend on the lattice misfit between the epilayer and the substrate. During the SK growth of Ge on Si (001), the formation of {105} facets that are elongated along the <100> directions with rectangular bases having up to 8:1 aspect ratio, make a hut like morphology. The evolution kinetics includes meta-stability, since they dissolve during annealing.

The self-limiting growth mechanism slows down the growth kinetics of the larger islands compared to the small islands. Thus, smaller islands convert to bigger ones, while larger islands stay constant, which makes uniform island size distribution [22].

The Stranski-Krastanow (SK) growth is a desired mode for various applications due to less imperfection in their structure that decreases the loss of performance compared to other modes. This info has attracted significant attention in recent years. Despite of this huge interest, the structure and distribution during morphological evolution of QDs grown by SK mode is not understood exactly.

1.6. Experimental observations

Undoubted, there should be a quit good information and capability to produce QD with desirable properties to put it into use. To follow this job, many experimental [23,24], theoretical and modeling and simulation [25-27] efforts have been performed in the subject of heteroepitaxial growth of self-organized quantum dots. However, there are still many unsolved issues regarding the kinetics and thermodynamics of the formation and evolution of QDs. In many of these investigations, Ge and Si/Ge on Si(001) or InGaAs on GaAs(001) systems have been commonly used as the model systems. Figure 1.5 depicts the SEM image of heteroepitaxial QDs formation in $Si_{0.75}Ge_{0.25}$/Si(001) system. Here, quantum dots have 135 nm width and 80 nm height.

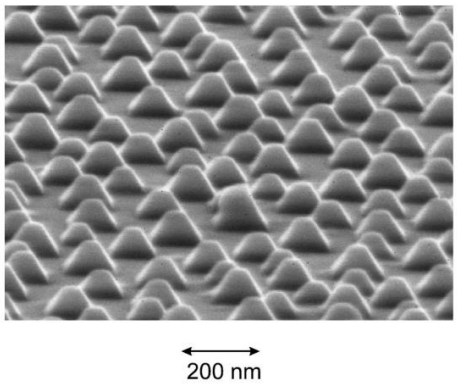


Figure 1.5: The SEM image of $Si_{0.75}Ge_{0.25}$/Si (001) system [28].

Over years, ordering and arranging of QDs was a challenging concept [29-31]. The overgrowth procedure of QDs can play an important role for spatial ordering of other QDs. Kiravittaya *et al* observed nano-holes, which start to appear due to strain effect, after 6 mL deposition of GaAs, in the middle of nanostructures. These holes have dimensions around 20-30 nm width and 1.5 nm depth and could be used as templates to guide the formation of closely spaced QDs [32]. However, the absolute positioning of QDs could be achieved by growth of QDs on a patterned substrate [33,34].

1.6.1. Nucleation and growth

The surface roughening process and its subsequent island formation can be understood by a simplified energetic argument [35-37]. During island formation, the total strain energy of the thin film decreases while the surface total energy increases. In an isotropic film and substrate, for $\lambda > \lambda_{cr}$, surface roughening is energetically favorable, where λ is the perturbed wavelength and λ_{cr} is the critical wavelength depending on the surface energy and strain energy density of the initially flat film. However, when $\lambda < \lambda_{cr}$, the perturbation will disappear and the surface will remain flat.

In other words, when the volume of an island is small, the surface energy of the side facets of the island plays an important role in preventing the island formation due to high ratio of the surface area to the volume. However, as the volume of the island exceeds a critical value, its significance diminishes and cannot stop the growth.

When the surface energy anisotropy and elastic anisotropy are included, the critical wavelength may change with the elastic and surface energy anisotropies. Thus, tuning these anisotropy strengths will change the roughening kinetics and therefore affect the island morphology. However, in this scheme of growth, the driving force of instability at low misfits is still under debate [38].

The size distribution of self-assembled QDs varies approximately in the range of$\pm$10% [21]. However, size homogeneity can be improved by tuning the growth conditions [39]. In InAs/Ga system, the lower growth rate induces larger QDs with better size homogeneity, which can be explained with the migration length of adatoms [39,40]. At low growth rate, the adatoms prefer to incorporate into existing QDs rather than forming new QD due to large migration length. In this case, better size homogeneity could be attained since adatoms can migrate longer so they have a higher chance of finding a suitable position with lower energy to be incorporated. Since larger QDs produce higher strain barriers, the adatoms prefer to incorporate into smaller QDs, which is called self-limiting growth.

The kinetic of growth is also another unresolved question. Studies show that, the growth velocity of quantum dots (QDs) and other structures by selective area epitaxy, may strongly be depend on crystallographic orientation [41, 42].

At low temperatures, the growth of InAs islands on GaAs substrate is simply due to accumulation of deposited material. However, at high temperatures, the volume of the islands exceeds the volume of deposited InAs on the substrate. Thus, the additional material comes from the wetting layer. This is a proof of the active role of wetting layers in the final stage of SK growth mode. Under the fixed amount of deposition, the growth of island needs the decrement of WL (wetting layer). As the surface energy density of the WL has a strong dependence on its thickness especially for thin WL, the growth of large islands becomes difficult and finally the system achieves equilibrium. Accordingly, it can be concluded that the thickness dependence surface energy of WL prevents the islands from growing up without limit and finally islands will have steady size [43, 44].

1.6.2. Quantum dots composition

In the case of equilibrium for a given shape, size and average composition, the composition profile (CP) is calculated by minimizing the total free energy consists of elastic energy, entropic and chemical mixing energies.

In the SiGe/Si system, the larger alloy component (Ge) segregates on the tensile region of top apex and upper corners, where strain is almost relaxed. On the other hand, the Si element segregates to corner of the base. In this case, Ge concentration decreases from the top to the base and base corners, which could be observed for all QDs independent of the size and shape. By increasing the θ, which creates steeper sidewalls, QDs decomposition degree increases compared to shallower QDs. Considering the negligible diffusion in bulk due to high-energy barrier at typical growth temperatures, Medhekar *et al* showed that even when the phase separation is thermodynamically favorable, the complete segregation cannot be achieved and entire dot could not reach the equilibrium [45]. However, the increased diffusion also occurs in subsurface regions since energy barrier is greatly reduced at surfaces.

This allows local equilibrium CPs to be established in the near surface regions during growth. Similarly, complete mixing is only seen for very shallow dots. Consequently, the kinetic of the growth mode, which dictates the surface mass transport and alloy mixing via surface diffusion at the growth process, becomes a key factor in determining the kinetically limited CP.

Strain induced segregation in quantum dots can substantially reduce the critical size for transition between the shapes with different facets. The critical volume for shape transition depends on the surface energies of the facets. The total energy of the decomposed dots is lower than the energy of the dots with uniform composition. However, the reduction in the energy is greater for the steeper dot, resulting in smaller critical size for transition in shape.

The composition profile depends strongly on the shape of the QD (slope, curvatures and other geometric features). However, the temperature change does not make any significant change on qualitative composition and just make slight change on quantitative composition profile [46].

Shenoy *et al*. used irreversible thermodynamic for their composition and morphological evolution model for fully faceted crystals. They established a two-component system with individual velocity for each component. Material flow and accordingly, the surface morphology could be determined based on the competition between the mass exchange in bulk and surface due to differences in bulk and surface chemical potentials including the size of the component and local stress field [47].

Shaleev *et.al* [48] have investigated the $Si_{1-x}Ge_x$ film growth with different Ge concentrations on Si (001) surface. The results show that the Ge concentration affects the misfit parameter between the film and the substrate and alters the critical thickness for initiation of island formation.

1.6.3. Shape transition of quantum dots

Shape transition has been detected during island forming generally when the amount of deposited material increases or the system evolves during *in situ* annealing. Experimental observations revealed that during deposition, small QDs with pyramid shapes start to make a morphological evolution to finally become a dome with a much larger size [49, 50].

By using Scanning Electron Microscopy (SEM) [51, 52], High-Energy Energy Diffraction (RHEED) [53] and Transition Electron Microscopy (TEM) [54] within experimental investigations, it have been observed that for InAs/GaAs(001) systems as well as SiGe/Si(001), the small InAs islands have pyramid shape including {137} facets, while larger islands have {101} facets with dome shape. After transition from pyramid to dome shape, the {101} facets with larger area and {111} facets with smaller area at the base are observable. However, {137} facets still exist at the top and the bottom of the dome [31, 32, 55] (Figure 1.6).

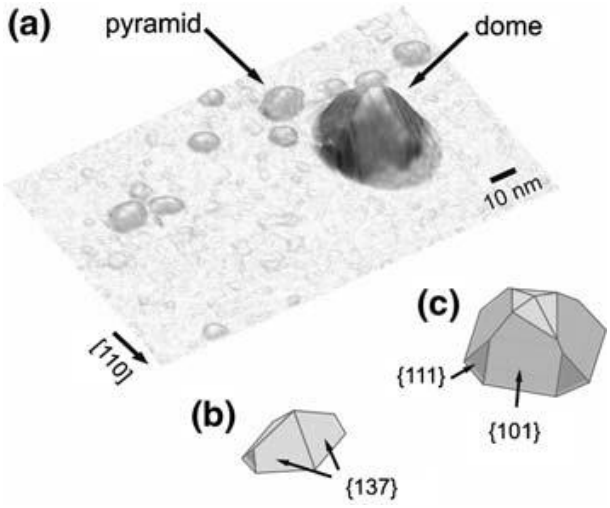


Figure 1.6: (a) 3D view STM image of Pyramid and Dome shape InAs QDs on GaAs(001) flat substrate. Schematic representation of **(b)** a pyramid and **(c)** a dome [32]

This Pyramid-Dome transition is also observed by Montalenti *et.al.* [56] and Baribeau *et.al.* [57] in $Si_{1-x}Ge_x$/Si (001) system (Figure 1.7).

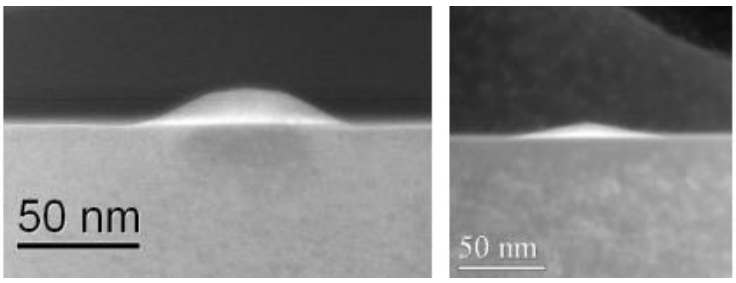


Figure 1.7: The figure in the right is the pyramid shape QD and the figure in the left is the same QD after transition into dome shaped island (6ML Ge/Si(001) is deposited in temperature of 650^{o}C)

Spencer *et al*. investigated the transition path from pyramid to dome. For any given volume, they minimized the E (energy) numerically of all possible shapes with respect to motion of each facet normal to itself. They showed that although the

transition evolves from symmetric pyramids to symmetric domes, however, the transition passes through a high asymmetric transition states pathway. The reason for nucleation of steeper facet in one side instead of two side of the pyramid is due to lower activation energy for this kind of transition instead of symmetric transition. They also found that the kinetic of the transition depends on the size of the island and it becomes faster with increasing the size [58].

1.6.4. Effect of substrate on the evolution of quantum dots

The substrate orientation is responsible for the appearance of specific bonding facets on the islands that determine their shape, which in turn defines the electronic structure.

On the other hand, the substrate structure and orientation influences the kinetic of adsorption, migration, and incorporation of atoms in heteroepitaxy as well as can directly influence the mechanism and velocity of strain relief, producing a change in the island sizes, shape and distributions [59- 61].

The structure and orientation of the substrate certainly play a serious role in the heteroepitaxial growth of highly mismatched systems. By changing the substrate, we also expect a change in the symmetry of QDs if the growth is epitaxial. It has been shown experimentally that on the high-symmetric GaAs (001) surface, the shape of InAs QDs exhibit the same two mirror-symmetry planes as the bulk (001) substrate [51,62].

Substrate affects the growth kinetics and morphological evolution of islands. As an example, GaN island nucleation on AlN substrate shows a small delay compared to nucleation on GaN substrate. Both form uniformly with hexagonal pyramidal shape. However, in the case of highly lattice mismatched sapphire substrate, the nucleation starts with a long delay also larger and fewer nuclei with prismatic shape form. This effect of substrate can be understood as the effect of strain caused lattice mismatch [63].

Studies done recently to investigate the effect of substrate strain on adatom binding and Ehrlich-Schowoebel (ES) barriers for FCC metals show that the compressive strain slows down the atomic exchange diffusion while increases the ES barrier. However, tensile strain decreases the ES barrier and promotes the layer-by-layer growth. Here, the ES barrier refers to the case when an atom that approaches the step on the top side meets a barrier that can be even greater than the diffusion barrier on the terrace $E_{Diffusion}$.

This additional barrier ΔE_{ES}, known as Ehrlich–Schwoebel barrier, comes from the case when crossing a step edge, an atom passes through the area with a low number of nearest neighbors. This ES barrier makes the layer-by-layer growth difficult while promoting the 3D island formation [64].

The investigation about the effect of substrate temperature during GaAs overgrowth by photoluminescence spectroscopy (PL) shows that PL spectra of the QD over grown at lower temperature (460 C) is significantly narrower and red shift compared to QD over grown at 500 C.

The PL line width obtained from a QD ensemble is generally attributed to the inhomogeneous broadening produced by the size and composition fluctuations of the QD. The larger QDs with narrower size distribution provide longer wavelength emission with narrower line width [40, 65, 66]. In addition, the low growth temperature is expected to preserve the shape of buried QDs [40, 67].

Studies performed by Schmidt *et al.* [68] and Wang *et al.* [69] show that during growth of QDs on Nano-wire surface below certain diameters, unlike in a traditional planar growth substrate, misfit strain can propagate along the length of the wire and cause self-organize growth in a periodic pattern to minimize the strain energy. Such misfit strain guided epitaxial islands not only present a new type of periodic nanostructures but also serve as periodic Nano-stresses providing a unique opportunity toward strain-engineered materials with novel mechano-electronic properties for applications such as thermo electronic and optoelectronic devices [70,71].

1.6.5. Dislocation formation during the evolution of quantum dots

Dislocation formation is a phenomena reported in literature and observed as thickness increases during deposition, which is due to strain relaxation [72, 73]. These defects are undesirable in electrical and optical applications as they act as scattering sires, recombination centers and leakage paths. Considerable efforts have been done to minimize the strain relaxation in heterostructure growth and to minimize the probability of dislocation and other faults formation due to this reason [19, 74, 75].

Katsuno *et al.* [76] established a table with different regions that present the proper growth mode from 1ML to 3ML considering the dislocations as strain relaxation sites. They believe that continuum elasticity may not give accurate results for small length scales so they established a new 2D elastic lattice model to investigate the lowest energy configuration and determine the equilibrium shapes by comparing energies of various surface configurations with misfit parameter (f) and binding energy. The result is given in Figure 1.8.

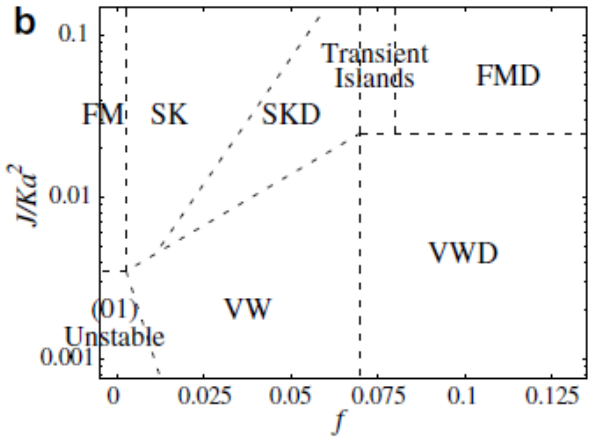


Figure 1.8: Different regions for the proper growth mode according to misfit parameter (f) and binding energy

Here, SKD, FMD and VWD are various growth cases, where the dislocations are observed. The authors claim that for the binding energy of the ordinary semiconductor systems, small misfits lead to the island formation, since introduction of a dislocation is energetically unfavorable. However, large misfits lead to generation of dislocations.

In the region between the SK growth mode and FM growth mode with dislocation, a special growth mode appears. When the layer thickness exceeds a critical thickness

needed for introduction of dislocations, islands forms on a dislocated layer. If the layer thickness increases, it leads to change the equilibrium configuration from three dimensional island growths on a wetting to a two dimensional flat layer growth, which could be due to thickness dependent dislocation formation energy. The energy gain by introducing the dislocation increases with increasing the thickness. Therefore, at high coverage, layer-by-layer growth with dislocation is more favorable than island formation.

1.6.6. Diffusion and stabilization of quantum dots

Wetting potential with the effect of limiting the valley-to-peak mass transport, which leads to limiting the island height growth, plays an important role on morphology stabilization of the surface as well as island coarsening and island size saturation.

In the coarsening process, as long as the wetting layer is not depleted, the rearrangement of mass between different islands takes place, which cannot be inhibited by the wetting effect. Therefore, in order to explain the island stabilization, there must consider another factor that is taking role. For an undulated surface (i.e., in the region of islands surface), the strain energy is concentrated at surface valleys but released at peaks; thus the diffusion would be the result of surface elastic energy density gradient from valleys to peaks, which leads to island growth. On the other hand, the stabilization effect of surface energy is in competition with this morphological destabilization process, where domination of each of them could determine the dynamics of island growth. Although the wetting potential still has the function of sustaining the wetting layer between islands and restricting the diffusion process, however, with increasing the dominance of the destabilization process, the effect of wetting potential becomes weaker. Consequently, the wetting effect cannot prevent the mass transport between islands, which is related to island migration or coarsening effect. This process can be controlled by the higher-order elastic energy

terms describing island interaction and correlation. Thus, at late-times, the islands height increases rapidly, which leads to appearance of islands with large aspect ratio between height and width.

Strained surface islands form as a consequence of the evolution of surface undulations, which occur due to the film morphological instability. Note that the rate of formation and growth of these islands varies at different surface locations, which is due to the nonlinear effects of elastic interaction. At the next stage, island coarsening occurs, means the growth of some islands is at the expense of other shrinking ones and hence the decrease of island density on the film surface. As time passes, such coarsening processes become much slower, and the system would approaches stable morphology with steady arrays of strained quantum dots. As expected, this late-time state of film surface morphology is significantly dependent on the value of film-substrate misfit strain, which results an increase of island density and a decrease of island spacing for larger misfits [77].

1.7. Theoretical and modeling efforts

Many theoretical works have been performed hitherto for the aim of prediction of QD size, number and density. However, the accuracy of these calculations was limited due to lack of knowledge on the QD shape and formation processes [78-80].

Phase field simulation is one of the most popular simulation methods used for surface behavior modeling. The significant advantage of this method is that the explicit tracking of surface is unnecessary; furthermore, the long-range elastic interactions of various structural defects are automatically taken into consideration during evolution.

Although several phase field models have been developed to investigate the surface evolution of stress induced islands, their stress fields could be solved using the mechanical equilibrium equations considering the approximation of small perturbation of the shear modulus [81, 82] or a proposed simple relaxation method [83, 84].

Most of the numerical works done in order to simulate the morphology evolution of surfaces and interfaces are based on nonlinear formulation. Both two and three-dimensional Galerkin finite element methods are used for studying the surface diffusion and displacement field for the modeling of the surface evolution behavior [85-88].

The growth or formation of epitaxially strained solid films is represented in the scope of problems so-called capillary and stress-driven shape and microstructural evolution in solids. The first serious attempt to approach this issue was performed by Asaro and Tiller [89]. By incorporating the elastic strain energy density (ESED) with the so-called chemical potential, they developed an equilibrium thermostatic model in order to express the morphology behavior of surfaces and interfaces during stress corrosion cracking. The Asaro/Tiller (AT) theory is partly accountable in isochoric systems, where the ESED is incorporated correctly with a positive sign in the Helmholtz free energy density.

Grinfeld [90] applied the Gibbs-Duhem stability theory of thermodynamic equilibrium for the isothermal and isochoric systems, characterized by the second variance in the total Helmholtz free energy denoted as $\delta^2 f > 0$ for the infinitesimal perturbations on the surface morphology associated with the surface acoustic waves generated in the non-hydrostatically stressed linear elastic solids in contact with their

melts. There are two common points in these theories; they both concern isochoric systems, implicitly or explicitly and they both propose the existence of a critical wave length, which above it, the flat free surface becomes unstable under the sinusoidal perturbations if certain conditions prevail [35].

Spencer [91] and Tekalign and Spencer [92, 93] present very successful analyses in order to explain the morphological instability of growing epitaxially strained dislocation-free solid films. These analyses were based on the surface diffusion driven by the capillary forces and misfit strains by elaborating various type of wetting potentials associated with the thickness dependent surface specific free energy considering the ESED parameter. They applied periodic boundary conditions to all numerical and analytical studies reported in the literature according to the work done by Kukta and Freund [94], on the equilibrium morphologies, in order to obtain steady state solutions of the nonlinear free moving boundary value problem. Their study is relied mostly on the instabilities initiated by white noise or small amplitude initial perturbations, where the film thickness is smaller than the wavelength of surface variations.

The present study is accomplished based on the theoretical development and modeling procedure performed by project group (Oren ve Ogurtani) [95-98], which is specified with prominent studies in investigating the evolution of surfaces and interfaces. They demonstrated that without even imposing any external perturbations, otherwise smooth surface of droplets, this isochoric composite system (film/substrate) simultaneously evolves towards the stationary state in the absence of the growth term by creating the SK islands or other proper morphologies depending on the imposed external and internal parameters. Unfortunately, the application of the rigid boundary conditions of any type to the computation domain restricts the natural motions of the triple junction (TJ) that lines between the isolated islands and

the substrate, and thus, the spontaneous evolution kinetics of the ensemble towards the possible stationary state morphologies are partially hindered. In our work, this restriction on the TJ motion is eliminated by employing an irreversible thermodynamic connection obtained by using the internal entropy production (IEP) hypothesis [96].

Accordingly, a specific formulation is developed for temporal velocity of the TJ singularity considering the wetting parameter, which depends only on the specific surface Helmholtz free energies of the thin film, substrate and the interface between them.

However, in the stationary regime, SK island describes strictly monotonic decrease in the profile while approaching to the perfectly flat and highly extended platform with a relatively sharp turn. This plateau corresponds to the wetting layer, which has almost uniform thickness and has direct contact between individually formed SK islands. This layer is very close to the prescribed thickness of the boundary layer, namely fraction of a nanometer.

Thus, according to required unknown information in the scope of formation of strained heteroepitaxial thin films, the purpose of this survey is to investigate the film/substrate interface (in) stabilities and evolution of quantum dots through computer simulation by utilizing the theoretical works of Ogurtani and Oren. Consequently, in this project, it is aimed to precisely control the size, shape, position and distribution of nano-crystalline quantum dots formed during molecular-beam epitaxy (MBE) technique. The results are expected to be a guide for fabrication of self-organized quantum dots that are suitable for technological devices.

CHAPTER 2

PHYSICAL AND MATHEMATICAL MODEL

A continuum theory based on the irreversible thermodynamics of surfaces and interfaces developed by Ogurtani [96] and Ogurtani and Oren [98] is used to simulate the evolution of epitaxial films, especially, the formation of the Stranski-Krastanow islands through computer modeling. In this theory, the thickness dependent surface Helmholtz free energy (for the isochoric systems) is taken into account.

In our model, a single random droplet is taken to generate our initial system. The droplet is presented with a symmetrical halve-wave length Cosine-function. The height (i.e., amplitude) and width of the droplet is symbolized by h_p and 2L, respectively. Since simulations are performed in 2D space, no profile variation and displacement in the film and the substrate takes place along the *z* axis (perpendicular to the plane of the schematics in Figure 2.1.

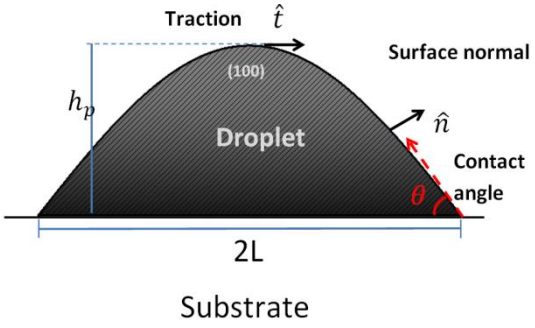


Figure 2.1: The side view of the droplet

To simplify the numerical computations, the following assumptions are made: The interface between the film and the substrate is assumed to be coherent. The top surface is exposed to vapor environment with a vapor pressure that can be neglected. Additionally, it is assumed that the film/substrate interface is smooth and the substrate is rigid. These assumptions guarantee that the initial displacement along the interface associated with the misfit strain ε_0 stays constant during the evolution process (i.e., Dirichlet boundary condition).

The droplet aspect ratio, β is defined as the value of the width over the value of the height $\beta = 2L/h_p$, θ_w is the temporal wetting contact angle between the film and the substrate and varies in the range of $\theta_w \in (0^\circ - \pi)$; while the zero degree corresponds to perfect wetting, 180^o resembles to no wetting conditions.

In the normalized and scaled length space, the initial height of the peak is chosen as the normalization parameter and thus $\bar{h}_p = 1$. Therefore, in the normalized system the initial shape of the droplet can be fully-defined by the aspect ratio β only. Similarly, to complete the predetermination of the initial morphology of the droplet, another additional parameter is needed to determine the size of the droplet, which corresponds to the intensity of the elastic strain energy density denoted by $\sum = \ell_0/\ell^*$. Here, ℓ_0 is the height of the droplet in real space and ℓ^* is the characteristic

length for the isochoric systems that depends only on the material properties of the film and the substrate including the misfit strain. If one takes ℓ^* as a length scaling parameter (ℓ_0), then, according to our definition $\sum = \ell_0/\ell^*$, we will come up with the result of $\sum \to 1$. In real space, ℓ_0 may be introduced as $\ell_0 = h_p$ for a given value of β. Thus, h_p can be calculated simply for a known value of $\sum$ and ℓ^* in real space as follows: $\sum = \ell_0/\ell^* \to h_p/\ell^*$ so $h_p = \sum \ell^*$. Therefore, this dimensionless parameter $\sum$ exclusively specifies the size while keeping the shape invariant (i.e., zooming); As a result of our normalization procedure, in the absence of the growth term, the aspect ratio β (i.e., shape) and the $\sum$ (i.e., size) are two basic values to describe the morphology of the final stationary states.

According to this micro-discrete formulation based on the irreversible thermodynamics of surfaces and interfaces, established by Ogurtani and Oren, the evolution kinetics of the surfaces or interfaces can be defined in terms of normal displacement velocities by solving the following free moving boundary value problem in 2D space. $\bar{V}_{ord}$ describes the surface normal displacement velocities for ordinary points and longitudinal velocity $\bar{V}_{Long}$ is related to the natural motion of the triple junction points, which are the points that droplet, substrate and the vapor phase encountered. During the evaluation, only normalized and scaled parameters and variables are used, which are indicated by bar signs (the normalization procedure will be represented later in this study).

$$\bar{V}_{ord} = \frac{\partial}{\partial \bar{\ell}}\left[\bar{D}_\sigma(\theta,\varphi,m)\frac{\partial}{\partial \bar{\ell}}\left(\Delta \bar{f}^0_{dv} - \sum(\bar{\sigma}_h)^2 + \Xi\bar{\sigma}_h + \bar{f}_{d/s}(\bar{y})\bar{\kappa} + \bar{\omega}(\bar{y})\right)\right] - \bar{M}_{dv}\left(\Delta \bar{f}^0_{dv} - \sum(\bar{\sigma}_h)^2 + \bar{f}_{d/s}(\bar{y})\bar{\kappa} + \bar{\omega}(\bar{y})\right) \qquad \text{Ordinary points} \qquad (2.1)$$

$$\bar{V}_{Long} = -\bar{M}_{Long}\bar{\Omega}^{-1}\{\lambda - \cos\theta_w\} \qquad \forall\ \lambda \geq 1 \qquad \text{Triple junction} \qquad (2.2)$$

Here, $\overline{D}_{\sigma}$ corresponds to the normalized anisotropic surface diffusion coefficient with respect to the minimum surface diffusivity. $\Delta\bar{f}_{dv}^{0}$ represents the thermal part of the Helmholtz free energy of transformation for a flat interface assuming that the isothermal processes are taking place in an isochoric system. In real space, it is defined as $\Delta\hat{f}_{dv}^{0}(T) = (\hat{f}_{v}^{0} - \hat{f}_{d}^{0})$. The positive value corresponds to condensation of the vapor phase or the growth of the droplet. $\hat{f}_{v}^{0}$ and $\hat{f}_{d}^{0}$ are the volumetric Helmholtz free energy densities for the realistic vapor and bulk droplet phases, respectively. The normalized hoop stress is denoted by $\bar{\sigma}_{h} \equiv (\sigma_{h}/\sigma_{0})$, where σ_{0} is the exerted stress (misfit or uniaxial). The hoop stress in plane strain condition is described by $\sigma_{h} = \hat{t}.\underline{\sigma}_{s}.\hat{t}$ where $\underline{\sigma}_{s}$ is a 2D-stress tensor evaluated at the region just adjacent to the surface layer and $\hat{t}$ is the unit surface tangent vector.

Dimensionless parameters $\sum$ and Ξ are respectively, the intensities of the Elastic Strain Energy Density (ESED) and the Elastic Dipole Tensor Interaction (EDTI) contributions on the stress-driven surface drift diffusion. $\bar{\ell}$ is the curvilinear coordinate along the surface (arc length) in 2D space normalized with respect to ℓ_{0} (arbitrary length scale). As told previously, ℓ_{0} may be selected as the peak height of the droplet or the ratio of the surface Helmholtz free energy of the film in the bulk to the elastic strain energy density [93] such as $\ell^{*} = f_{d}/w_{0}$. Here, $w_{0} = (1 - \vartheta_{d}^{2})\sigma^{2}/2E_{d}$ denotes ESED, which is associated with the nominal biaxial misfit stress considering the third dimension.

The film thickness h_{0} is defined as the *integrated* film thickness, and it may be given by $\bar{h}_{0} = 2h_{p}/\pi \rightarrow 0.637$ for the scaled halve wave length Cosine-shape flat droplets, where $\overline{h}_{p} \rightarrow 1$.

$\bar{f}_{d/s}(\bar{y})$ is the height dependent surface free energy of the droplet and for an isochoric system, it depends on the local distance y between the surface layer and the substrate:

$$\bar{f}_{d/s}(y) = \frac{(f_d + f_s)}{2f_d} + \left(\frac{(f_d - f_s)}{f_d}\right)\frac{1}{\pi}\arctan(y/\delta) \tag{2.3}$$

f_s, and f_d are the surface energy of the surface and droplet in the bulk form respectively and δ is the characteristic length scale that determines the size of the transition region. The wetting potential $\bar{\omega}(\bar{y})$ is defined as

$$\bar{\omega}(\bar{y}) = \frac{1}{\sqrt{1+\bar{y}_x^2}}\frac{(f_s - f_d)}{\pi f_d}\frac{\bar{\delta}}{\bar{\delta}^2 + \bar{y}^2} \tag{2.4}$$

$\bar{\kappa}$ is the normalized local curvature and is taken to be positive for a convex solid surface (rounded). Similarly, the positive direction of the surface displacement is assumed to be towards the vapor. The second group of terms in the equation (2.1) is related to the growth or phase transformation (*condensation or evaporation*) kinetics, which is not considered in this study but detailed information is given in Reference [95].

By applying the following assumptions, the equation (2.1) is converted to the form below. We assume that there is no anisotropy in diffusion and surface stiffness regime in the system. Additionally, there is also no transformation or growth phenomena considered during the evolution process.

$$\bar{V}_{ord} = \frac{\partial}{\partial \bar{\ell}}\left[\frac{\partial}{\partial \bar{\ell}}\left(-\sum(\bar{\sigma}_h)^2 + \Xi\bar{\sigma}_h + \bar{f}_{d/s}(\bar{y})\bar{\kappa} + \bar{\omega}(\bar{y})\right)\right] \quad (2.5)$$

We can write this equation as

$$\bar{V}_{ord} = \frac{\partial}{\partial \bar{\ell}}\left[\frac{\partial}{\partial \bar{\ell}}(\Psi)\right] \quad (2.6)$$

where

$$\Psi = -\sum(\bar{\sigma}_h)^2 + \Xi\bar{\sigma}_h + \bar{f}_{d/s}(\bar{y})\bar{\kappa} + \bar{\omega}(\bar{y}) \quad (2.7)$$

If we consider

$$\bar{\omega}_{cal} = \bar{f}_{d/s}(\bar{y})\bar{\kappa} \quad (2.8)$$

Then

$$\Psi = -\sum(\bar{\sigma}_h)^2 + \Xi\bar{\sigma}_h + \bar{\omega}_{cal} + \bar{\omega}(\bar{y}) \quad (2.9)$$

Again if we write

$$\omega et = \bar{\omega}_{cal} + \bar{\omega}(\bar{y}) \tag{2.10}$$

we obtain the following equation

$$\Psi = -\sum(\bar{\sigma}_h)^2 + \Xi\bar{\sigma}_h + \omega et \tag{2.11}$$

The wetting parameter λ is defined as $\lambda = \left[(f_s - f_{ds})/f_{d/s}\right]$, where f_s is the Helmholtz surface free energy of the substrate, f_{ds} is the interfacial free energy between the droplet and the substrate, and $f_{d/s}$ is the height dependent surface free energy of the droplet. θ_w is the temporal dihedral or wetting contact angle and varies in the range of $\theta_w \in (0^\circ - \pi)$,. $\bar{\Omega} \cong 10^{-3}$ is the normalized atomic volume in the particle representation by assuming tentatively that the scaling length is in the range of 10 atomic spacing (more details can be found in the papers [96, 98]).

$\bar{M}_{Long}$ is the ratio of the triple junction points mobility denoted by $\widehat{M}_{Long}$ to the surface mobility, $\widehat{M}_d$. Similarly, $\bar{M}_{dv}$ is the normalized growth mobility, which in general may depend on the temperature and the surface stress [97].

In the present computer simulations similar to work done by Spencer [91] and his coworkers [92], we assumed that $f_{d/s} \cong 0$ for the wetting potential, which is acceptable for the coherent boundaries such as the interface between epitaxially grown film and the substrate.

We also scaled the time and space variables $\{t, \ell\}$ in the following manner: the normalized time scale is introduced by $\tau_0 = \ell_0^4/(\Omega^2 \widehat{M}_d^0 f_d)$ where, $\widehat{M}_d^0$ is an atomic mobility correspondent for the mass flow at the surface layer.

$$\bar{t} = t/\tau_0\,, \bar{\ell} = \ell/\ell_0\,, \bar{\kappa} = \kappa\ell_0\,,\ \ \bar{L} = L/\ell_0\,, \Delta\bar{f}_{dv}^0 = \frac{\Delta\hat{f}_{dv}^0}{f_d}\ell_0,\ \ \bar{\sigma}_h = \frac{\sigma_h}{\sigma_0}$$

$$w_0 = \frac{(1-\vartheta_d^2)}{2E_d}\sigma_0^2\,, \sigma_0 = \frac{E_d}{(1-\vartheta_d)}\varepsilon_0\,, \sum = \frac{(1-\vartheta_d^2)\ell_0}{2E_d f_d}(\sigma_0)^2 \equiv \frac{\ell_0}{\ell^*} \qquad (2.12)$$

The misfit strain ε_0 at the film/substrate interface is introduced as a *Dirichlet boundary* condition by specifying the displacement vector in 2D space as $u \to \hat{\imath}\varepsilon_0 x$ (i.e., in 3D space $u \to [\hat{\imath}\varepsilon_0 x\,, \hat{k}\varepsilon_0 z]$), and taking the droplet center at the film/substrate interface as the origin of the coordinate system to avoid shifting. The applied stress is chosen as the biaxial stress $\sigma_0 = E_d\varepsilon_0/(1-\vartheta_d)$, where, E_d and ϑ_d are Young modulus and Poisson ratio of the droplet shape film respectively. ε_0 is the misfit strain at the film/ substrate interface. This subject is very suitable for the Indirect Boundary Element Method (IBEM) solution of the plain strain isotropic elasticity problems [99]. If we take $E_d \to 1; \varepsilon_0 \to 1$ as the initial scaling data, only the actual value of the Poisson's ratio of the film is required for the computation of the normalized stress distribution. The values of $\{E_d, \varepsilon_0\}$ are embedded in the definition of $\sum$.

In Chapter 3, the numerical methods applied to solve the partial differential equations, developed here, will be described in detail.

CHAPTER 3

NUMERICAL PROCEDURES

3.1. Preparation of the initial system

The initial system is defined as a two-dimensional droplet shape film, which is introduced by using finite numbers of nodes on the outer surface with predetermined segment length. The positions of the nodes are determined using Cartesian coordinate according to the reference point. Although the model is two dimensional, to take the advantage of vector algebra, it is considered three-dimensional with zero value in z-axis. The vector is represented as $\vec{r}^{\langle i \rangle} = \begin{vmatrix} x \\ y \\ 0 \end{vmatrix}$

Using vectors for determination of node positions simplifies the calculation of the segment length s, and centroid vectors, $\vec{r}_c$ defined as:

$$s_i = \left|\Delta\vec{r}^{\langle i \rangle}\right| \quad \text{Where} \quad \Delta\vec{r}^{\langle i \rangle} = \vec{r}^{\langle i+1 \rangle} - \vec{r}^{\langle i \rangle} \tag{3.1}$$

and

$$\vec{r}_c = \frac{\vec{r}^{\langle i+1 \rangle} + \vec{r}^{\langle i \rangle}}{2} \tag{3.2}$$

3.2. Calculation of the turning angles at the nodes

The turning angles at the nodes are the angles between two vectors that connect three successive nodes as shown in the Figure 3.1. It is calculated using the definition of two vectors.

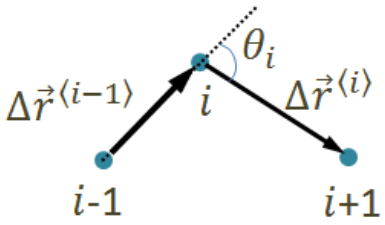


Figure 3.1: The turning angle at the node i

$$\theta_i = \begin{cases} \arcsin\left[\dfrac{\Delta\vec{r}^{\langle i-1 \rangle} \times \Delta\vec{r}^{\langle i \rangle}}{|\Delta\vec{r}^{\langle i-1 \rangle}||\Delta\vec{r}^{\langle i \rangle}|}\right] & \text{if} \quad \Delta\vec{r}^{\langle i-1 \rangle} \times \Delta\vec{r}^{\langle i \rangle} \geq 0 \\ \pi - \arcsin\left[\dfrac{\Delta\vec{r}^{\langle i-1 \rangle} \times \Delta\vec{r}^{\langle i \rangle}}{|\Delta\vec{r}^{\langle i-1 \rangle}||\Delta\vec{r}^{\langle i \rangle}|}\right] & \text{Otherwise} \end{cases} \tag{3.3}$$

3.3. Calculation of node curvatures

The curvature at each node may be assessed using discrete geometric relationship based on the fundamental description of radius of curvature and normal vector. The procedure is based on the given geometric relationships;

Three distinct points in a plane determine a unique circle, where the curvature of a circle with radius ρ_i (radius of curvature) is calculated as $1/\rho_i$. Figure 3.2 indicates such a circle that crosses through three consecutive nodes $i-1, i, i+1$. To evaluate the local curvature at node i, the following identities can be written down using the known values of the segment lengths, s_i and the segment turning angles θ_i.

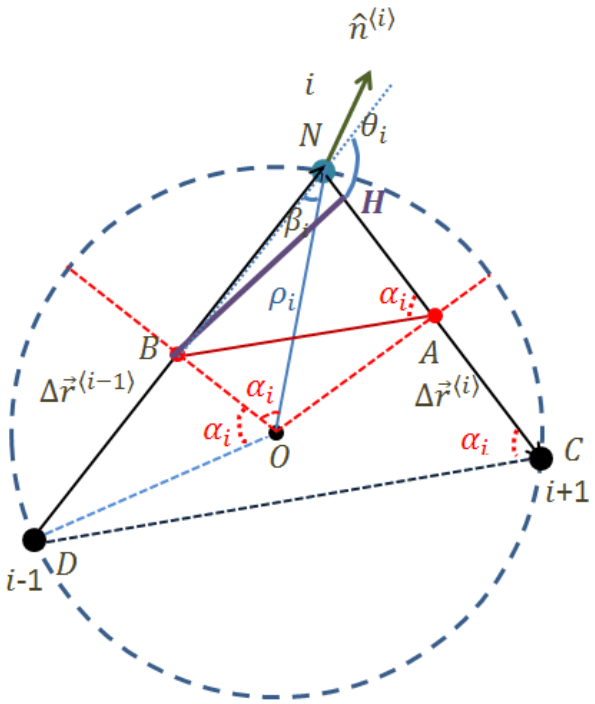


Figure 3.2: The unique circle that crosses through three successive nodes, O is the center of the circle, $s_{i-1} = |DN|$, $s_i = |CN|$, $OB \perp\perp DN, OA \perp\perp CN$ and $AB \parallel CD$

$$\rho_i = \frac{s_{i-1}}{2\sin(\alpha_i)} \tag{3.4}$$

Where α_i is the angle $B\hat{O}N$ and it is very easy to see that this angle also equals $B\hat{A}N$. Then, the curvature at the node i is given by:

$$\kappa_i = \frac{1}{\rho_i} = \frac{2\sin(\alpha_i)}{s_{i-1}} \tag{3.5}$$

The tangent of the angle α_i can be formulated as follows:

$$\tan(\alpha_i) = \frac{|BH|}{|AH|} = \frac{|BH|}{|AN| - |HN|} = \frac{|BN|\sin(\pi - \theta_i)}{|AN| - |BN|\cos(\pi - \theta_i)} = \frac{|BN|\sin(\theta_i)}{|AN| + |BN|\cos(\theta_i)}$$
$$= \frac{\frac{1}{2}s_{i-1}\sin(\theta_i)}{\frac{1}{2}s_i + \frac{1}{2}s_{i-1}\cos(\theta_i)} = \frac{\sin(\theta_i)}{\frac{s_i}{s_{i-1}} + \cos(\theta_i)} \quad (3.6)$$

Using the equation (3.5), the local curvature is calculated as:

$$\kappa_i = \frac{2\sin\left(\operatorname{atan}\left(\frac{\sin(\theta_i)}{\frac{s_i}{s_{i-1}} + \cos(\theta_i)}\right)\right)}{s_{i-1}} \quad (3.7)$$

3.4. Calculation of the local line normal vectors

To evaluate the normal line vector at each node, we need the value of angle $O\widehat{N}D$; from Figure 3.2, one can simply write $\beta_i = (\pi/2) - \alpha_i$.

Multiplying the $\Delta\vec{r}^{\langle i \rangle}$ (vector that connects the successive nodes) by anticlockwise rotation matrix will give us the vector along the local line normal vector as shown below:

$$\vec{n}^{\langle i \rangle} = \begin{vmatrix} \cos(\beta_i) & -\sin(\beta_i) & 0 \\ \sin(\beta_i) & \cos(\beta_i) & 0 \\ 0 & 0 & 1 \end{vmatrix} . \Delta\vec{r}^{\langle i \rangle} \quad (3.8)$$

3.5. Calculation of the hoop stresses by using the Indirect Boundary Element Method (IBEM)

The quantum dots dynamics is driven by the capillary-induced surface drift-diffusion. In this study, we applied the simplest implementation of the IBEM to evaluate the hoop stress at the free surface of the droplet, as well as along the interface between droplet and the substrate. In fact, it is also possible to generate the complete stress distribution field in the interior region of the sample as a byproduct. Here, Neumann (i.e., traction free boundary condition) and Dirichlet boundary conditions (i.e., prescribed displacements) are utilized, respectively, along the free surface of the droplet and at the interface between droplet and the substrate. Therefore, we have assumed that the substrate is rigid, and Dirichlet boundary conditions are applied all along the interface. Hence, the displacements are calculated from the misfit strain ε_0, by $u_x(0) = \varepsilon_0 x$.

In solid mechanics, there are some known relationships introduced between stress and strain in the material, where stress is defined as force per unit area inside a solid.

The traction vectors acting on three plains, which are parallel to three axes, are defined as follows:

$$\vec{t}_1 = \begin{bmatrix} \sigma_x \\ \ddot{\tau}_{xy} \\ \ddot{\tau}_{xz} \end{bmatrix} ; \vec{t}_2 = \begin{bmatrix} \ddot{\tau}_{yx} \\ \sigma_y \\ \ddot{\tau}_{yz} \end{bmatrix} ; \vec{t}_3 = \begin{bmatrix} \ddot{\tau}_{xz} \\ \ddot{\tau}_{zy} \\ \sigma_z \end{bmatrix} \quad (3.9)$$

Infinitesimal strains are calculated in the x, y, and z directions (u_x,u_y and u_z), by using stress components as follows:

$$\varepsilon_x = \frac{\partial u_x}{\partial x} \tag{3.10}$$

$$\varepsilon_y = \frac{\partial u_y}{\partial y} \tag{3.11}$$

$$\varepsilon_z = \frac{\partial u_z}{\partial z} \tag{3.12}$$

$$\gamma_{xy} = \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \tag{3.13}$$

$$\gamma_{yz} = \frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y} \tag{3.14}$$

$$\gamma_{zx} = \frac{\partial u_z}{\partial x} + \frac{\partial u_x}{\partial z} \tag{3.15}$$

The stress-strain behavior in elastic material can be formulated by Hooke's law. For an isotropic material, calculations can be done in three dimensions.

$$\varepsilon_x = \frac{1}{E}[\sigma_x - \nu(\sigma_y + \sigma_z)] \tag{3.16}$$

$$\varepsilon_y = \frac{1}{E}[\sigma_y - \nu(\sigma_x + \sigma_z)] \tag{3.17}$$

$$\varepsilon_z = \frac{1}{E}[\sigma_z - \nu(\sigma_y + \sigma_x)] \tag{3.18}$$

$$\gamma_{xy} = \frac{1}{G}\ddot{\tau}_{xy}\,, \gamma_{yz} = \frac{1}{G}\ddot{\tau}_{yz}\,, \gamma_{zx} = \frac{1}{G}\ddot{\tau}_{zx} \tag{3.19}$$

Here, E is the elasticity modulus, ν is the Poisson's ratio and G is the shear modulus. The relationship between them is given as

$$G = \frac{E}{2(1+\nu)} \tag{3.20}$$

The governing differential equations are acquired from the condition of equilibrium. For plane strain conditions, the following can be written:

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \ddot{\tau}_{xy}}{\partial y} + b_x = 0 \tag{3.21}$$

$$\frac{\partial \sigma_y}{\partial y} + \frac{\partial \ddot{\tau}_{xy}}{\partial x} + b_y = 0 \tag{3.22}$$

Here, b_x and b_y are body force components in x and y directions. Substitution of the equations, results in the following equations

$$G\left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2}\right) + \left(\frac{\nu E}{(1+\nu)(1-2\nu)} + G\right)\left(\frac{\partial^2 u_y}{\partial x \partial y} + \frac{\partial^2 u_y}{\partial x \partial y}\right) + b_x = 0 \tag{3.23}$$

$$\left(\frac{\nu E}{(1+\nu)(1-2\nu)} + G\right)\left(\frac{\partial^2 u_x}{\partial x \partial y} + \frac{\partial^2 u_x}{\partial y \partial x}\right) + G\left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2}\right) = 0 \tag{3.24}$$

The fundamental solution for two-dimensional plane strain problem, is calculated for point unit loads in x and y directions of magnitude 1, which are extended to infinity in both $\pm z$ directions. The solution was first performed by Lord Kelvin.

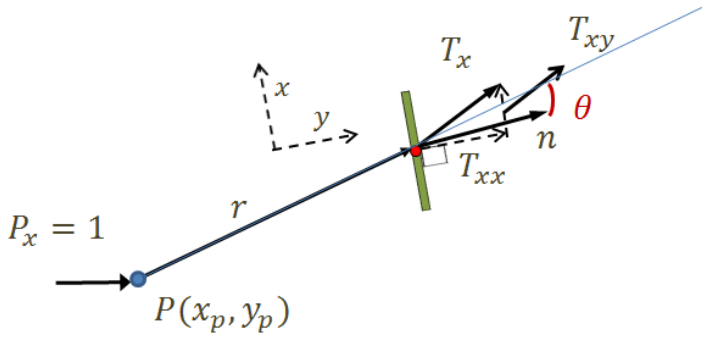


Figure 3.3: The symbolization for 2D Kelvin solution

The solutions for the displacements in x and y directions due to a unit load in x direction are calculated as follows:

$$\mathrm{U_{xx}(P,Q) = C\left[C_1 \ln(\frac{1}{r})r_x^2\right]} \tag{3.25}$$

$$\mathrm{U_{xx}(P,Q) = Cr_x r_y} \tag{3.26}$$

$$\mathrm{C = 1/(8\pi G(1-\nu)\ , C_1 = 3-4\nu} \tag{3.27}$$

To use the IBEM, the solutions for the boundary stresses (tractions) are also required. These tractions act on the surface with an outward normal direction of *n*. By taking the derivative of the displacement solution, the fundamental solutions for strains are calculated. Accordingly, by applying the Hooke's law, the fundamental solutions for tractions also can be evaluated.

According to IBEM, the tractions at point Q due to a unit load at P in x direction arecalculated as:

$$\vec{T}_{xx}(P,Q) = \frac{C_2}{r}[C_3 \ln 2r_x^2]\cos\theta \tag{3.28}$$

$$\vec{T}_{xy}(P,Q) = \frac{C_2}{r}\left[2r_x r_y \cos\theta + C_3\left[n_y r_y - n_x r_x\right]\right] \tag{3.29}$$

$$C_2 = 1/(4\pi(1-\nu),\ C_3 = 1-2\nu, \cos\theta = \frac{1}{r}r.n \tag{3.30}$$

Where θ is illustrated in Figure 3.3. If we assume that there is no body force acting in the domain, then we can write:

$$u_x(P) = \int\left[\vec{t}_x(Q)U_{xx}(P,Q) + u_y(Q)U_{xy}(P,Q)\right]dS$$

$$-\int\left[u_x(P)\vec{T}_{xx}(P,Q) + u_y(Q)\vec{T}_{xy}(P,Q)\right] \tag{3.31}$$

Using matrix algebra, we obtain

$$u(P) = \int U(P,Q)\vec{t}(Q)dS - \int \vec{T}(P,Q)u(Q)dS \tag{3.32}$$

Consequently, the hoop stresses are calculated according to equation (3.32), since through this two integral equations system, the tractions $\vec{t}$ are directly related to the displacements in the boundary u. This achievement removes the need to compute fictitious forces.

3.6. Explicit Euler's Method

Using Explicit Euler's method helps to perform the time integration of equation (2.1) in order to predict the surface evolution behavior at any run.

The initial time step is selected in the range of (0.0005, 0.05). However, the time step does readjust at any run step according to the maximum surface velocity and minimum segment length such that at each run step, the displacement is kept constant, which leads to recalculation of the time step for the specific maximum node velocity. This so-called adapted time step auto-control mechanism combined with the self-recovery effect associated with the capillary term, guarantees the long time numerical stability and the accuracy of the explicit algorithm even after performing $10^3 - 10^6$ steps.

3.7. Adaptive Remeshing

In order to keep the experiment time and accuracy in an acceptable level, adaptive remeshing is required during the simulation. To express in more detail, in the case of exceeding the segment lengths from a critical value, the system loses the accuracy. On the other hand, increasing the number of nodes increases the computation time.

These two statements indicate that in order to obtain optimum results, the size of segment lengths should be kept in the range between the prescribed minimum and maximum segment lengths $[s_{min}, s_{max}]$. If the distance between two successive nodes becomes longer than s_{max}, the mid-point is converted into a node as illustrated in Figure 3.5.a.

Similarly, if the distance between two successive nodes becomes shorter than s_{min}, in order to control the nodes number, as shown in Figure 3.5.b the mid-point node replaces the two successive nodes. To increase the accuracy, the same procedure applies to the next segment length.

The further node is removed from the mesh and the new segment is formed after such a node removal process, the new segment lengths have to be controlled whether it is longer than s_{max} or not.

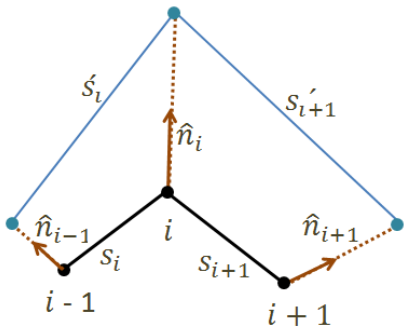


Figure 3.4: The profile evolution according to local line normal ($\hat{n}_i$), s_i is the segment length between two nodes and $\acute{s}_i$ is the segment length after the displacement.

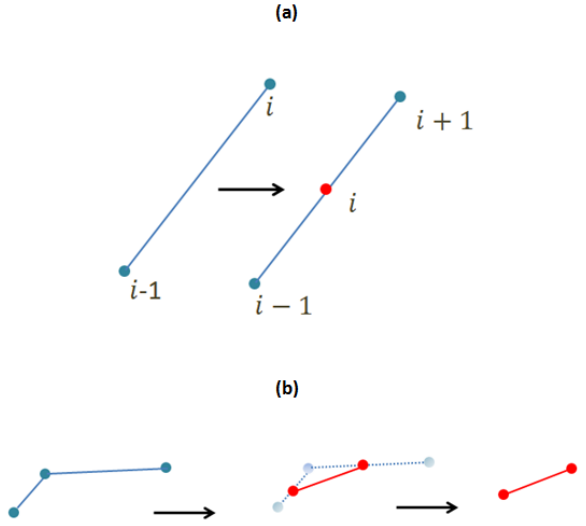


Figure 3.5: Remeshing for the cases of (a) the segment length is bigger than the maximum allowable segment length (b) the segment length is smaller than the minimum allowable segment length

Finally, the node velocities are calculated by solving the governing equations (2.1) and (2.2). Thus, the new configuration of the system can be estimated as a result of small displacement in time due to velocity of the nodes. All the numerical procedure steps are repeated for this new configuration to evolve the system further in time. Figure 3.6 depicts the numerical procedure of this study.

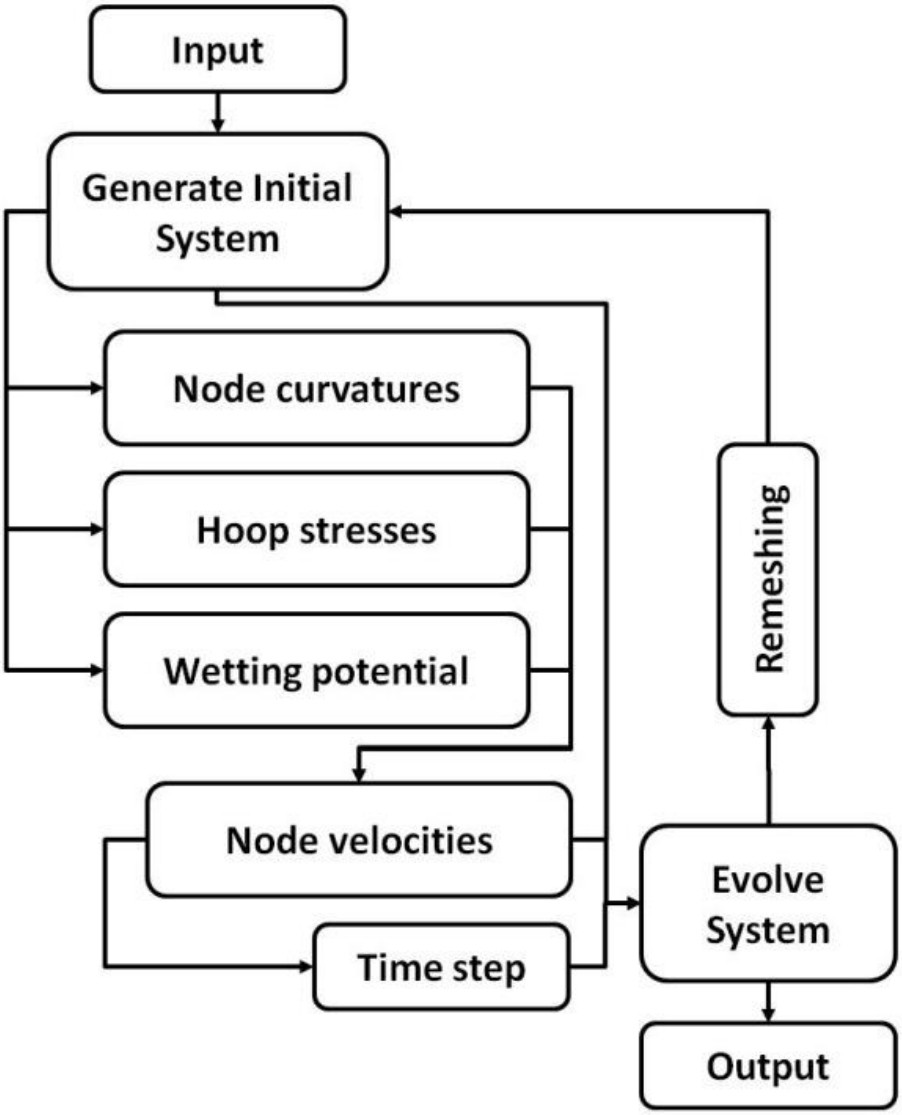


Figure 3.6: The numerical procedure of the program

CHAPTER 4

RESULTS AND DISCUSSION

In this chapter, the effects of strain relaxation on morphological evolution of QDs and the spontaneous evolution of an isotropic isolated thin solid droplet on a rigid substrate under various stress fields will be demonstrated. Those results, presented within this chapter, have been obtained by using the program explained in Chapter 3, in detail.

4.1. Determination of safe run parameters

Before starting to run various simulation experiments, it is important to consider some precautions in order to save computation time and to avoid performing redundant experiments. There are two independent parameters in the numerical procedure that can affect the computation speed and accuracy of the expected results. These two parameters are the initial node number (n) and the initial time step (δt) that describe in what detail the system is generated and the evolution speed, respectively.

Increasing the node numbers and decreasing the time step, lead to enhance the accuracy of the evolutionary path and thus the final stable configuration and vice versa. However, this accuracy may need prolonged calculation times to be able to reach the final stable state, if present. This situation generates a demand for some

precautions to predetermine these parameters to obtain the optimum time and accuracy for the numerical calculations. As we will see later this safe parameter sets also depend on the evolution route of the quantum dots in other words the path of the process.

Table 4.1. outlines the preliminary experiments that are carried out for various δt values. The other system parameters are chosen to cover various evolution paths. Here, by increasing the δt values, we continuously monitored the QD morphology. In those experiments, we observed that there is a critical value for the initial time step, below which the final system morphology converges. To provide a clear view, we summarized the results in Table 4.1., where the ✓ signs indicate the converged results (i.e., acceptable results) and × signs indicate the results deviates from the identical experiments conducted with smaller time steps (i.e., not acceptable).

Table 4.1. Effect of the initial time step on the convergence of experiments

Experiment \ δt	**0.0005**	**0.001**	**0.005**	**0.01**	**0.025**	**0.05**
$\boldsymbol{\beta = 28, \bar{M}_{long} = 1, \sum = 3.00, \lambda}$**=0.017**	✓	✓	×			
$\boldsymbol{\beta = 28, \bar{M}_{long} = 1, \sum = 1.50, \lambda}$**=0.990**	✓	✓	✓	✓	×	
$\boldsymbol{\beta = 28, \bar{M}_{long} = 1, \sum = 0.80, \lambda}$**=0.017**	✓	✓	×			
$\boldsymbol{\beta = 28, \bar{M}_{long} = 2, \sum = 0.80, \lambda}$**=0.500**	✓	✓	✓	✓	✓	×
$\boldsymbol{\beta = 10, \bar{M}_{long} = 2, \sum = 0.20, \lambda}$**=0.707**	✓	✓	✓	✓	✓	×
$\boldsymbol{\beta = 10, \bar{M}_{long} = 1, \sum = 1.00, \lambda}$**=0.500**	✓	✓	✓	✓	✓	✓
$\boldsymbol{\beta = 10, \bar{M}_{long} = 2, \sum = 0.75, \lambda}$**=0.017**	✓	✓	✓	✓	✓	×
$\boldsymbol{\beta = 10, \bar{M}_{long} = 2, \sum = 0.50, \lambda}$**=0.017**	✓	✓	✓	✓	✓	×

In Table 4.1, the results of a set of experiments carried out with different time steps are shown. The final configurations obtained with time steps $\delta t = 0.0005$ and $\delta t = 0.001$, are almost identical but when we increase the time step up to $\delta t = 0.005$, the final configuration deviates from the previous equilibrium configurations.

The larger time steps make the QD morphology lose its symmetry and/or experience a different route during the evolution, which happens due to large value of δt that causes numerical errors.

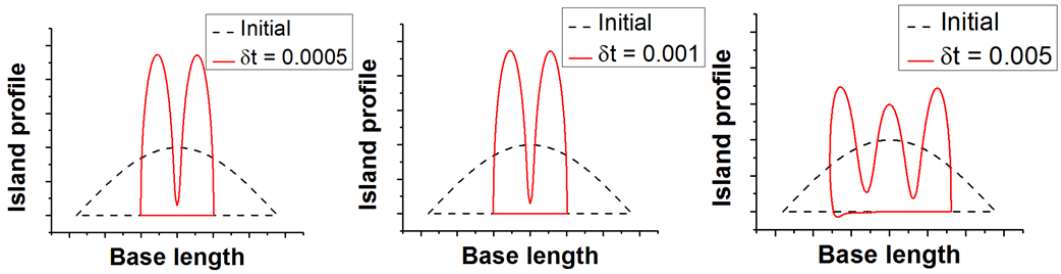


Figure 4.1: The effect of different δt on the accuracy of computation process for $\delta t = 0.0005$, $\delta t = 0.001$, $\delta t = 0.005$. In these simulations, the input parameters are $\beta = 28$, $\bar{M}_{long} = 1, \sum = 0.8, \lambda = 0.017, f_f = 1$, $f_s = 1.2$

According to the information given in Table 4.1, the approximate best δt for each individual experiment may be selected. However, to determine the acceptable δt for future experiments, in which we cannot foresee the evolution path prior to carrying the simulation experiment, it is safer to choose the smallest δt that fits for all the experiments, which is thus selected to be $\delta t \leq 0.001$. However, the computation time for the experiments should also be considered in order to choose the best result. For the smaller values of $\sum$ (for which the relatively simple evolution routes observed), the real computation time difference between various δt ´s may be negligible but at higher stress values, as the system becomes more and more

complex (more island number) the time saving parameter may become very important factor for the accuracy of the results. For example, for $\sum = 0.8$, in the case of $\delta t = 0.001$, the time needed for the system to achieve the final configuration is approximately 3 days. However, if we decrease the δt to 0.0005, the computation time increases to 4 days. According to this time difference, as we have the similar result, it is more convenient to select the lower CPU time consuming choice, which is $\delta t = 0.001$.

Similarly, to determine the optimum node number (n), some preliminary experiments are performed. The results are summarized in Table 4.2. As the node number increases thus the segment length decreases, the accuracy of the numerical procedure increases as a cost of increased computation time.

Table 4.2. Effect of the initial node number on the convergence of experiments

Experiment \ Node number	**40**	**60**	**80**	**100**
$\boldsymbol{\beta = 28, \bar{M}_{long} = 1, \sum = 3.00,\ \lambda = 0.017}$		×	✓	✓
$\boldsymbol{\beta = 28, \bar{M}_{long} = 1, \sum = 1.50,\ \lambda = 0.990}$		×	✓	✓
$\boldsymbol{\beta = 28, \bar{M}_{long} = 2, \sum = 0.80,\ \lambda = 0.500}$		×	✓	✓
$\boldsymbol{\beta = 10, \bar{M}_{long} = 2, \sum = 0.75,\ \lambda = 0.017}$	×	✓	✓	✓

In Figure 4.2, the results of a set of experiments carried out with different initial node numbers are shown. The final configurations obtained with node numbers of 80 and 100 are almost identical but when we decrease the node number to 60, the final configuration observed, turned out to be different from others.

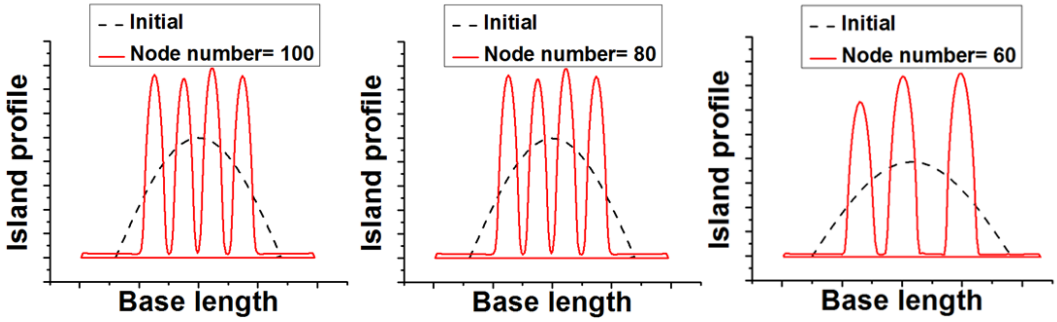


Figure 4.2: The effect of different node number on the accuracy of computation process for $n = 100$, $n = 800$, $n = 60$. In these simulations, the input parameters are $\beta = 28,\ \bar{M}_{long} = 1, \sum = 1.5, \lambda = 0.990, f_f = 1,\ f_s = 1.2$

Based on these results, the approximate best node number is chosen as 80. In this specific case, the approximate calculation time for no-stress experiments are less than 10 minutes and for the stress applied experiments, it varies between 1-7 days. The calculation period depends on the shape and the node number that change according to remeshing at each run step. Here, for $\sum = 0.8$, in the case of node number = 80, the time needed for the system to achieve the final configuration is approximately 1 day and in the case of node number = 100, the time needed for the system to achieve the final configuration is approximately 3 days. As seen from these numbers it is crucial to work with the optimal node numbers.

4.2. The effect of triple junction mobility on the morphological evolution

As discussed earlier, the morphological evolution of islands is governed by the velocity equations, where the velocity is calculated for each node using equations 2.1 and 2.2. Here, $\bar{M}_{long}$ is associated with the mobility of the triple junction points at the edges. It is expected that by increasing the TJ mobility, the process acceleration takes place. The results of preliminary experiments verify this hypothesis.

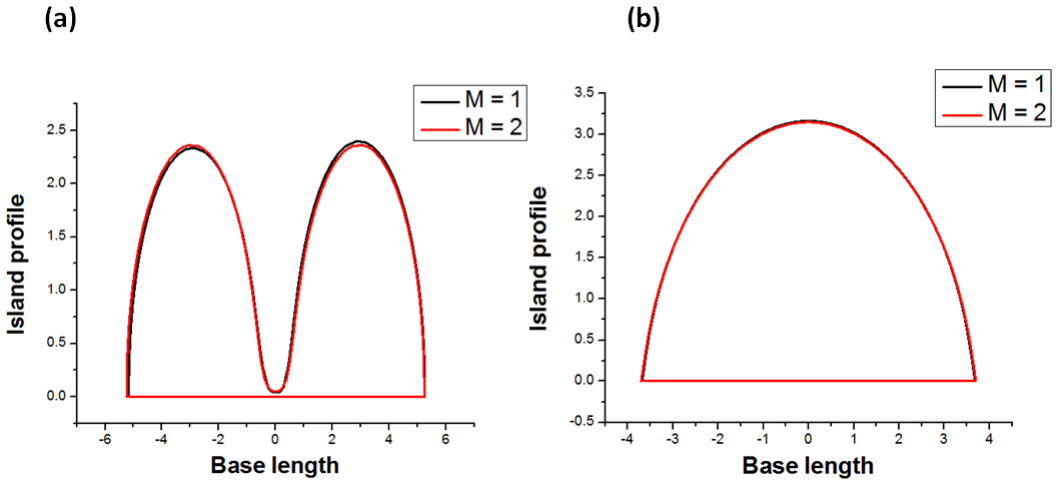


Figure 4.3: The effect of different $\bar{M}_{long}$ on the accuracy and normalized evolution time of computation process (a) the normalized evolution time for $\bar{M}_{long}$= 1, $\bar{M}_{long}$= 2 are 10.856876 and 8.822997 respectively, where the input parameters are $\beta = 28, \sum = 1, \lambda = 0.017, f_f = 1,\ f_s = 1.2$, (b) the normalized evolution time for $\bar{M}_{long}$= 1, $\bar{M}_{long}$= 2 are 29.792955 and 12.819254 respectively, where the input parameters are $\beta = 28, \sum = 0.5, \lambda = 0.259, f_f = 1,\ f_s = 1.2$.

Figure 4.3 indicates that the TJ mobility affects the kinetics of the evolution process rather than the final stable configuration. We carried out two sets of experiments that in both the TJ mobility values are changed. In the first set, where the stress level is chosen as $\sum = 1$, we observed both TJ mobility values converged into almost same morphologies. In the second set we increased the stress level to $\sum = 2$, and again observed a similar scenario. In both experiments, by increasing the mobility, the normalized time necessary to reach the stable state decreases 10% and 55 % respectively. However, this does not guarantee the reduction in the computation time. For example, some experiments show that the computation time increases in the case of larger mobility values. Thus, in order to choose a single mobility value to simplify the comparison between various experiments, it is decided to choose $\bar{M}_{long} = 1$, which may be a help to save the time or perhaps the accuracy of future experiments.

4.3. Droplet simulation

In this thesis, we focused on the QD formation via nucleation by producing shallow droplets, rather than surface roughening route in which the island forms through morphological instability. Thus, the aim of these simulations is to investigate the effect of various materials properties on the morphological change of droplets placed on rigid substrates. These parameters include the stress, film aspect ratio, wetting parameter, interface thickness and contact angles that also depend on the surface energy difference between the droplet and the substrate. In the following subsections, we will investigate those parameters in two cases: First we will assume that the stress level is negligible, which will allow us to demonstrate the effect of other material properties. Then, we took into account the misfit stresses formed between the droplet and the rigid substrate systematically.

In the following experiments, the final configurations associated with particular input parameters are demonstrated in a specific form. The final configuration is also represented by a Gaussian curve (i.e., second order) given by $G(x;\bar{h}_p,\bar{w}) = \exp(-\ln(2)\,x^2/\bar{w}^2)$, where $\bar{w}$ and $\bar{h}_p$ are halve-width and peak height in normalized space respectively.

Here, Figure 4.4.a shows the initial (dashed line) and the final configuration of the droplet (solid red line). The stability of the final image is determined according to stability of some parameters such as fractional height, base extension and wetting contact angle, which are shown in Figure 4.4.c and 4.4.d that become constant after 150 runs. Figure 4.4.b shows the hoop stress distribution in the morphology. Here, the final profile value of $\bar{h}_p$, and $\bar{w}$ are calculated to be 1.619 and 5.362, respectively.

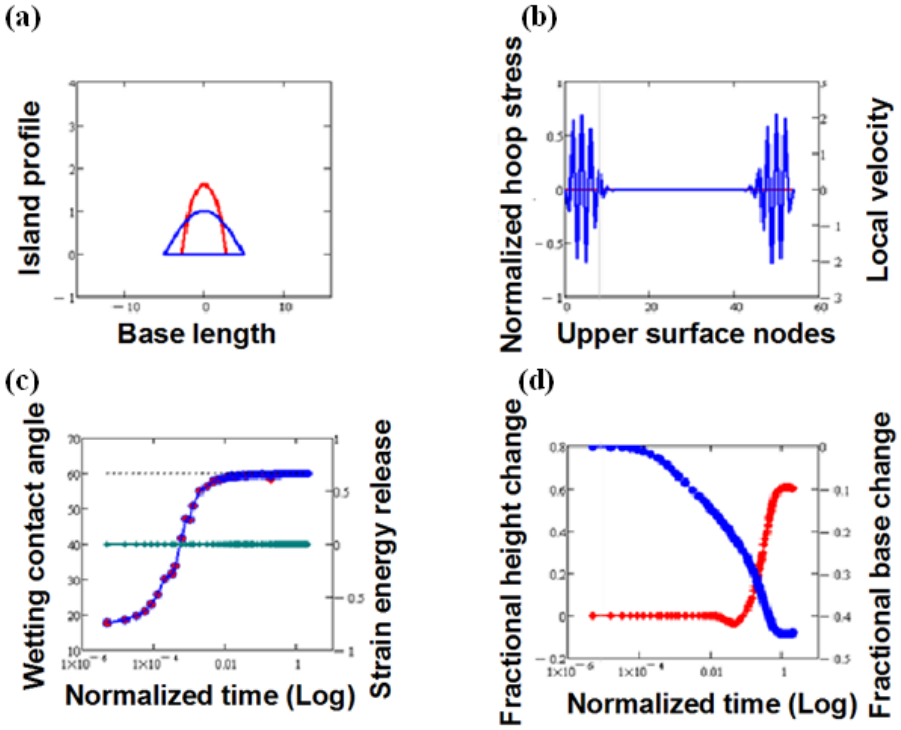


Figure 4.4: (a) Spontaneous formation of SK islands, (b) Normalized hoop stress of nodes in final configuration, (c) time evolution of wetting contact angle, (d) time evolution of fractional height and base length change. In these simulations, the input parameters are $\beta = 10,\ \bar{M}_{long} = 2,\ \sum = 0, \lambda = 0.5, f_f = 1,\ f_s = 1.2$

4.3.1. QD evolution without stress

In this section, the results are obtained from a set of experiments performed under the condition that intensity of Elastic Strain Energy Density (ESED) value is taken to be zero, which means that there is negligible misfit stress between the film and the substrate. The real material parameters are inserted as input data, which here, is assumed to be the Ge thin film epitaxial growth on the Si substrate. Namely: $E_{Ge} = 103\ GPa,\ \varepsilon_0 = -0.0042$ (misfit strain), $\nu_{Ge} = 0.3,\ f_{Ge} = 1, f_{Si} = 1.2.$ These numbers imply a characteristic length of $\ell^* = 12.11\ nm$, which is used to calculate the height and the base length of droplets corresponding to the strain energy intensity parameter for a given aspect ratio. If we take $\bar{E}_{Ge} \to 1$ and $\bar{\varepsilon}_0 \to 1$ as the initial scaling data, then only the actual value of the Poisson's ratio of the film is required for the computation of the normalized stress distribution. The results of

this program simulates the evolution process of the flat thin film layer under the effect of various parameters, where the evolution may terminates in the stable SK (Stranski-Krastanov) islands or may result in disappearing of the hills and formation of FM (Frank-Van der Merwe) layer.

To observe the effect of wetting contact angle on the final island morphology, the evolution of droplets were investigated by applying various λ values. Here, the following relationship exists between the wetting contact angle θ and the wetting parameter: $\theta = \mathrm{acos}(\lambda)$. Figure 4.5 represents the final morphology of non-stressed droplets considering various wetting contact angles.

It is obvious that as the contact angle increases (the slope becomes steeper), the height of the droplet increases due to constant volume of the droplet since there is no material deposition into or removal from the droplet system. The small angle leads to formation of shallower islands. There are two peculiar limits for TJ contact angle, which makes the system unstable, namely, 0 degree and 90 degree (Figure 4.5.b and 4.5.c).

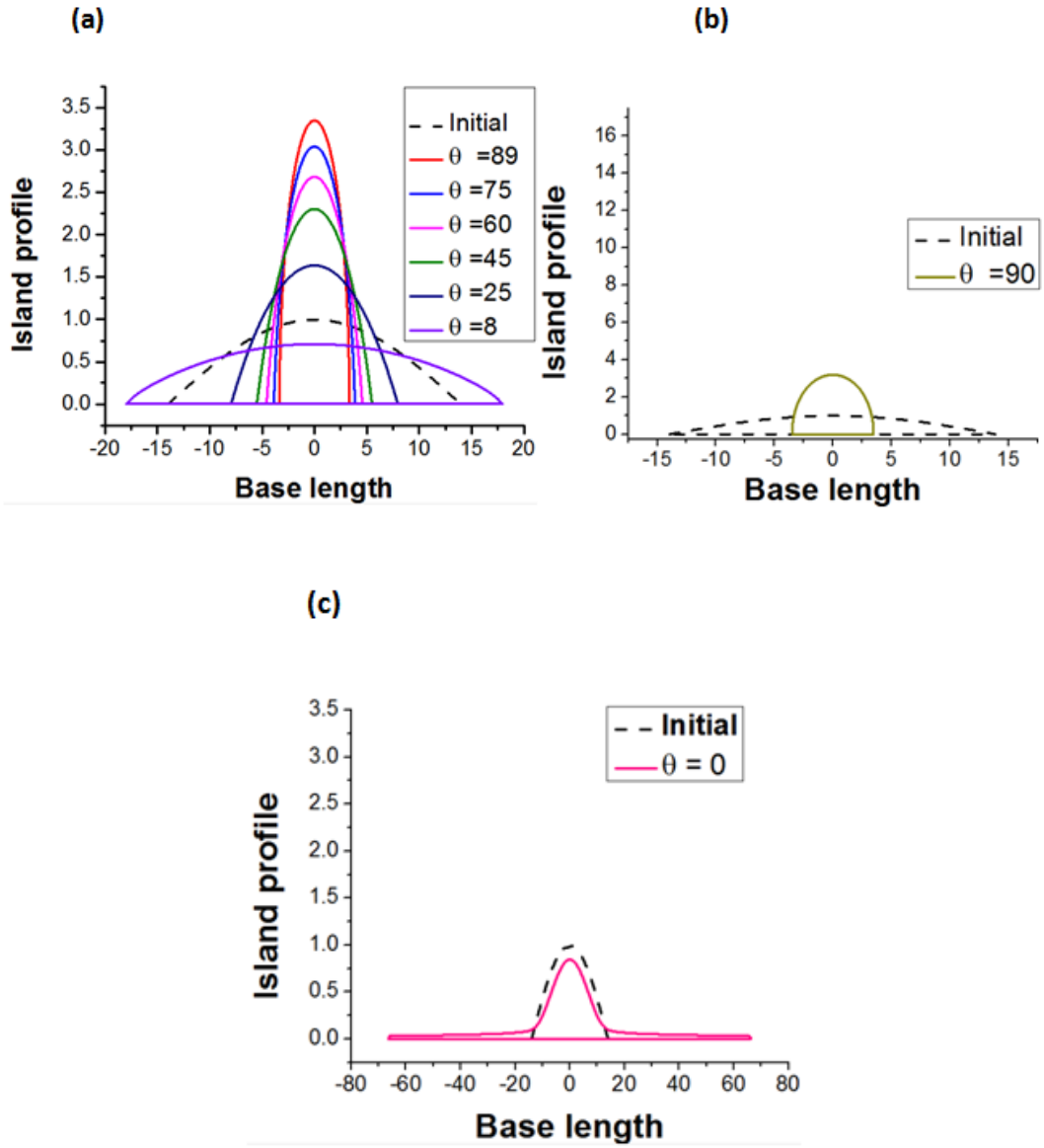


Figure 4.5: The effect of wetting contact angle on the final morphology. In these simulations, the input parameters are $\beta = 28,\ \bar{M}_{long} = 1, \sum = 0,\ f_f = 1,\ f_s = 1.2$

In the case of 90-degree angle, the system tries to evolve to a very perfect semicircle. At the initial stages, the system shows the expecting evolution behavior. However, at a critical point some factors cause interruption in the program and the system could not reach the final stability. This interruption can be due to some computation errors or maybe some unknown disregarded physical factors. In zero degree case, the island tries to become stretched on the substrate until forming an

extremely thin layer, which can continue into infinity (perfect wetting condition). In this case, the upper surface nodes become very close to the bottom interfacial part nodes of the system. This situation may cause the developed program to fail in prolonged run times. The last configurations before the breakdown of the experiments with limit values for wetting contact angles are also given in Figure 4.5.b. and 4.5.c. To avoid these problems, we consider angles very close to the limits, which can give stable results. 8 degree and 89 degree are found experimentally as suitable limit data that can be used without any problem during numerical procedure.

In order to have better visualization for θ detection, Figure 4.6 is prepared by enlarging a small section of Figure 4.5.a with identical scaling axes.

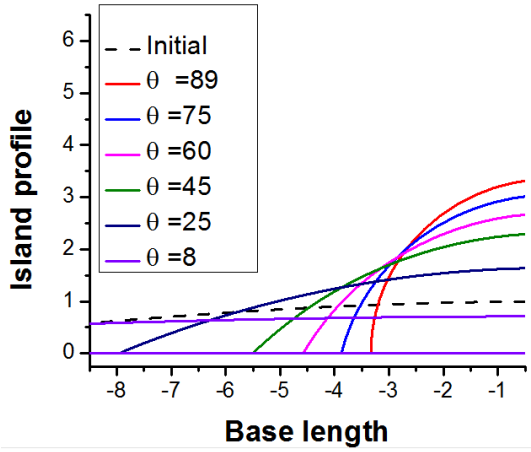


Figure 4.6: Zooming the Figure 4.4.a for better θ detection. The input parameters are $\beta = 28,\ \bar{M}_{long} = 1, \sum = 0,\ f_f = 1,\ f_s = 1.2$.

The evolution behavior of the contact angle of each experiment vs. normalized time is represented in Figure 4.7. Here, if we take the wetting contact angle of the initial shape as 18°, all of the experiments with larger θ s follow more or less a similar path. Their wetting contact angle increases until they reach their equilibrium wetting

angle θ, dictated by the material properties. For the rest, two cases (8° and 0°), as the equilibrium θ is smaller than the initial shape, they experience a decrement in wetting contact angle, which leads to formation of shallower islands.

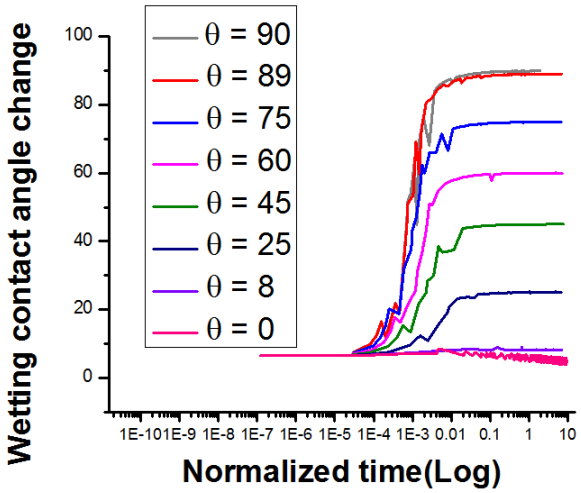


Figure 4.7: Time evolution of wetting contact angles for different θ values. In these simulations, the input parameters are $\beta = 28,\ \bar{M}_{long} = 1, \sum = 0,\ f_f = 1,\ f_s = 1.2$.

Figure 4.8 shows the 3D images of the three different experiments (low, medium and high wetting contact angles), which give better visualization about island evolution. Here, the final stages of the islands are projected onto the initial configurations. The final height of the QD's for $\theta = 89^\circ$, 45°, 8° are 3.3649, 2.3121 and 0.8144 respectively. Here, we need to state that there is no change in the volume of the islands. The only reason for the shape alteration is the variation in wetting contact angle, since other parameters are considered to be constant.

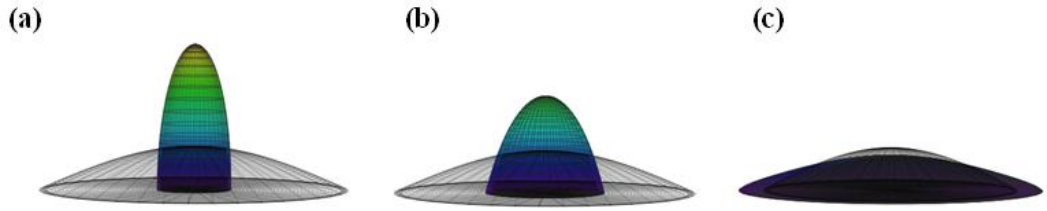


Figure 4.8: 3D images of spontaneous SK islands for three different λ values (a) $\theta = 89^{\circ}$, (b) $\theta = 45^{\circ}$, (c) $\theta = 8^{\circ}$, in these simulations, the input parameters are $\beta = 28,\ \bar{M}_{long} = 1, \sum = 0,\ f_f = 1,\ f_s = 1.2$.

Figure 4.9 demonstrates the island height change with time over the course of the morphological evolution process. As we discussed earlier, the stability of this curve is an effective factor in the determination of the stable (final) system configuration. For the two exceptional cases (90˚ and 0˚), the wetting contact angle curve shows no stability according to Figure 4.9.b, where we can see that the height of the islands alters continuously and make it difficult to obtain a stable system.

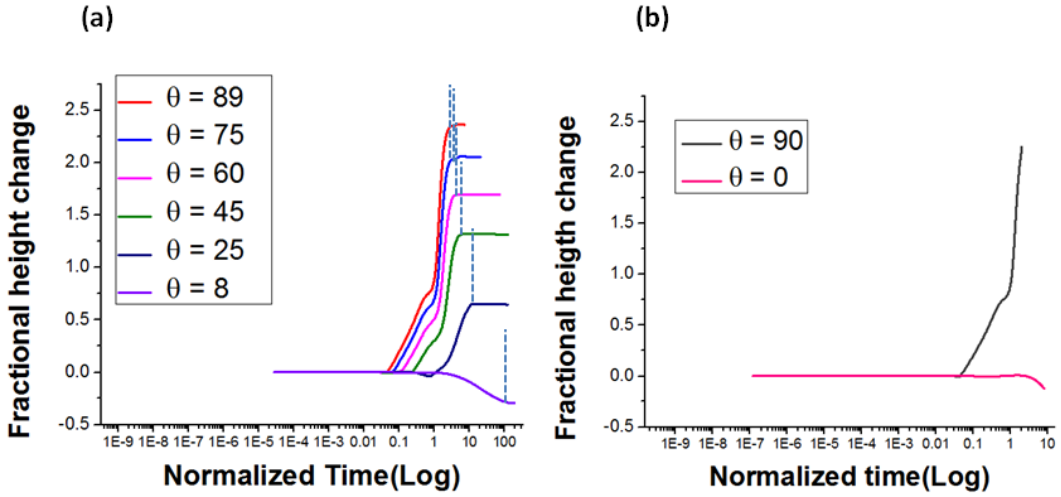


Figure 4.9: Time evolution of the change in fractional height of the islands for different θ values. In these simulations, the input parameters are $\beta = 28,\ \bar{M}_{long} = 1, \sum = 0,\ f_f = 1,\ f_s = 1.2$.

In Figure 4.9.a, the approximate points that stability begins are depicted with dashed lines. The interesting point that is noticeable in Figure 4.9.a is that the necessary time to reach the equilibrium contact angle is decreasing as the final and initial contact angle differences increases. This behavior may happen due to θ gradient between the initial and final configuration, where the initial system has a contact angle value of $\theta = 18^{\circ}$. This θ difference acts as a potential that affects the kinetic of the evolution process. However, on the other hand if the final equilibrium contact angle is less than the initial value; we experienced the longest time, required to reach the equilibrium QD configuration, among others.

We also investigated the effect of the aspect ratio (β) on the final morphology of the droplets and the results are indicated in Figure 4.10.

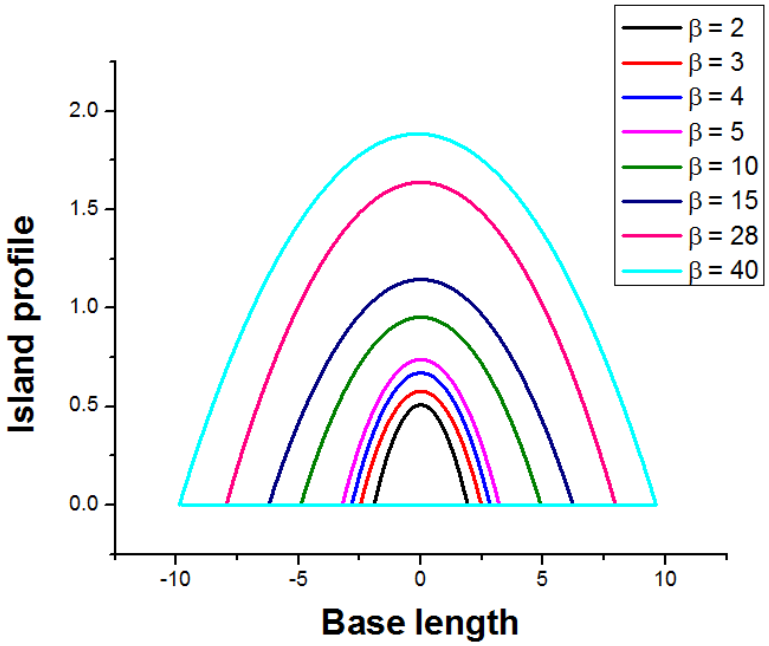


Figure 4.10: The effect of aspect ratio on the morphological evolution of quantum dots, where in these simulations, the input parameters are $\theta = 25^{\circ}$, $\bar{M}_{long} = 1, \sum = 0,\ f_f = 1,\ f_s = 1$

In Figure 4.10, 6 different β values are used while all the other system parameters are kept unchanged. According to our definition ($\beta = 2L/\bar{h}_p$), by altering the β, we can change the initial shape of the droplet, since the $\bar{h}_p$ value is invariant and it is equal to 1, by decreasing the β, we actually decrease the width of the droplet, which finally forms a smaller quantum dot. As a result, if the misfit stress is negligible, the β parameter only affects the size or the scale of the stable island formation.

4.3.2. QD evolution with stress

In this section, we applied various levels of stress to the system, and similar to the previous section, we have investigated the evolution behavior of the droplets under the effect of θ, β and $\sum$. Finally, we obtained an approximate 2-D phase diagram using different values of θ and $\sum$, which shows the stability regions of the various final morphology traits.

The morphological evolution stability of the system is very sensitive to the wetting contact angle, aspect ratio, applied stress and other material properties. Although, we know that the surface energy and surface diffusivity are function of crystallographic orientation (i.e. anisotropic), within the frame of this work, it is assumed that all the system parameters are isotropic.

As we have shown in the previous subsection, if the misfit stress level is negligible, the final islands morphologies are dictated mostly by the equilibrium dihedral angle and we always have single islands without any fragmentation over the course of the simulations. However, when we start including the effect of the misfit stresses into our simulations, the overall picture is changed: fragmentation and formation of multiple islands, sometimes separated with a wetting layer resembling the formation of Stranski-Krastanow thin film growth mode, are appeared. From now on, we will

discuss those materials properties, which will result in various final stable morphologies.

Figure 4.11 indicates the effect of the wetting contact angle on the final morphology of the QDs, where in this experiment the value of stress is chosen as$\sum = 1.5$. In the case of $\theta = 25^{\circ}$, similar to the no stress case we obtained a single isolated QD. However, when we compare this single QD case with the same one with no stress case (Figure 4.5.a), the effect of stress is obvious i.e. a sharper QD with almost twice peak height and half peak base length. Therefore, we state that with a proper control over the stress levels, one can control the aspect ratio of the stable QD formed and thus the energy spectrum, which in fact is technologically one of the most important outcomes.

However, this is not the case for all equilibrium wetting angle values. When we start increasing the wetting angle to $\theta = 45^{\circ}$, the final morphology stabilizes in doublet by fragmenting into two with similar size and shapes. By decreasing the wetting contact angle to $\theta = 89^{\circ}$, the island numbers increases to four (i.e. quadruplet). In all cases, the overall volume is constant; thus as the islands fragment into more and more daughter islands, the islands size decrease (Figure 4.11.a).

We also carried out simulation experiments for cases with higher wetting tendency (i.e. lower wetting angle). In Figure 4.11.b we present such two cases: for $\theta = 20^{\circ}$, the initial droplet evolves into a quadruplet with similar islands but compared to the case in $\theta = 89^{o}$, the edges are not very steep and bent outwards indicating a wetting layer formation. By decreasing the wetting contact angle further, we observed that the edges bent more. Finally, at a critical value, the wetting layer starts to appear.

The wetting layer formation is very obvious in $\theta = 8^{\circ}$, where we will have quadruplet similar islands with a very thin layer, which is stretched from both edges.

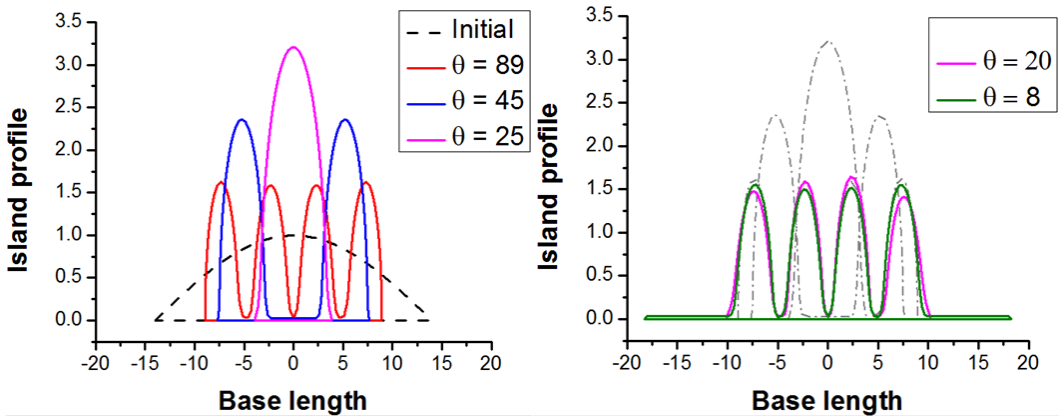


Figure 4.11: The effect of wetting contact angle on the final morphology. In these simulations, the input parameters are $\beta = 28,\ \bar{M}_{long} = 1, \sum = 1.5,\ f_f = 1,\ f_s = 1.2$

Figure 4.12 depicts the effect of $\sum$ on the morphological evolution of quantum dots. Considering that we assumed all the material properties are isotropic, the morphological change of the islands due to low stresses is limited. For low $\sum$ values ($\sum < 0.7$), the result is always a single quantum dot but with larger aspect ratio compared to the initial shape, which is due to the height increment of the islands (Figure 4.12.a). However, for larger $\sum$ values, the evolution behavior of the islands entirely changes and the single island divides into a dual island shape. As shown in Figure 4.12.a and 4.12.b, these two islands are distinct but identical. However, they are connected with a very thin wetting layer the length of which is stress dependent. By applying $\sum = 1$, the distance between the islands of this dual shape becomes larger. By increasing the $\sum$ value to $\sum = 2$, we observed the formation of quintuplet distinct islands (Figure 4.12.b).

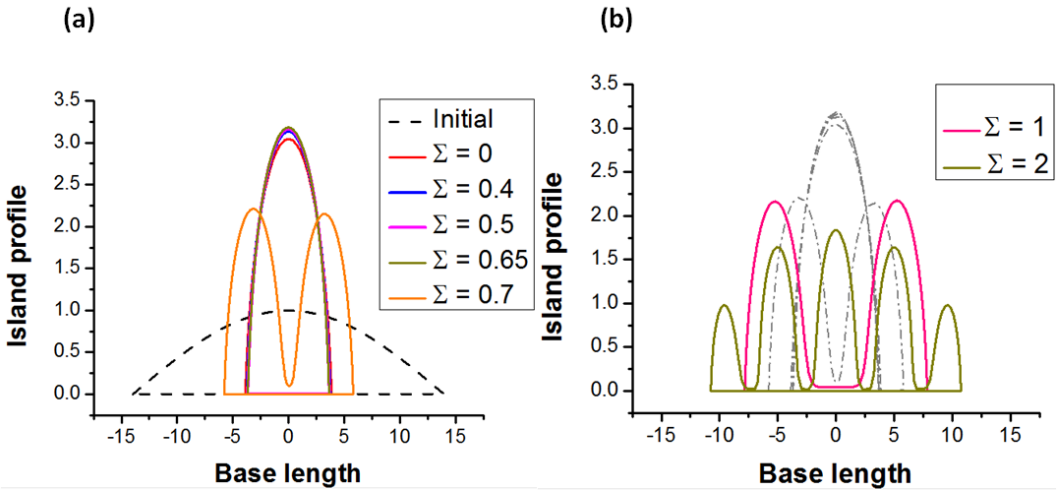


Figure 4.12: The effects of stress on the final morphology; in these simulations, the input parameters are $\beta = 28,\ \bar{M}_{long} = 1,\ \theta = 75^{\circ},\ f_f = 1,\ f_s = 1.2$

To understand the connection between the island height and the applied misfit stress level, we collected the data from various experiments and plotted them in Figure 4.13. The separation point of the island into distinct islands can be recognized with a sudden decline in the height of the island.

As a general tendency, as the stress level increases, the final height of the islands also increases but there are various threshold values above which we observe island fragmentation into two and more islands. Note that the threshold stress values for fragmentation are also a function of wetting angle thus the respective surface energies of the film and the substrate.

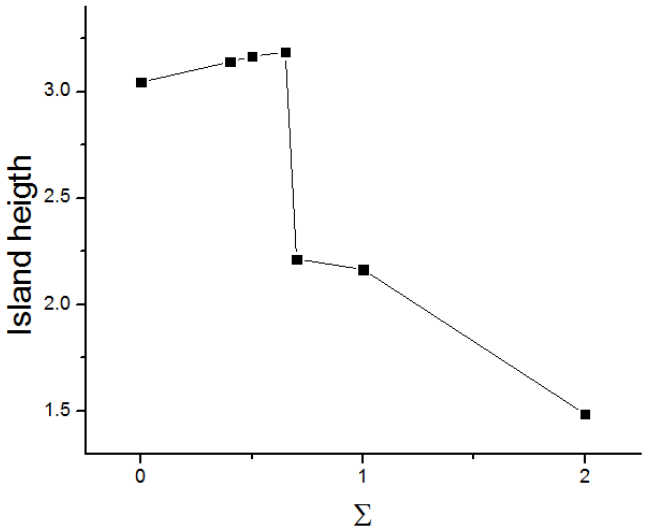


Figure 4.13: The effect of stress on the island height of final morphology; in these simulations, the input parameters are $\beta = 28,\ \bar{M}_{long} = 1,\ \theta = 75^{\circ}\ ,\ f_f = 1,\ f_s = 1.2$

Each experiment has its own path of evolution dictated by the initial parameters associated with both material characteristics and the environmental effects. Any shape transition path occurs in such a way to minimize the overall system energy. In some cases, we observe that two experiments with different initial parameters may come up with similar final stable configurations, however their path for shape transition are entirely different. Investigation about the transition paths will give us very interesting and beneficial results in controlling the QDs morphology. However, it is not within the scope of this project.

In Figure 4.14, we demonstrated the evolution process of four different experiments, for which the final configurations are also compared in Figure 4.12. Here, we took snapshots of island morphology at various times. The time used here is a normalized unit-less number as discussed earlier and given by Eq. (2.12).

In Figure 4.14.c, we demonstrated the path of the wetting layer formation. Initially we observed the formation of ripples on the surface of the droplet and then those ripples evolve into three separate islands the one in the middle is smaller compared the other two. Finally, bigger islands consume the middle one while separating from each other but are connected with a thin wetting layer. The effect of $\sum$ on the normalized time required for stability is also noticeable.

From figure 4.14 it can be seen that by increasing the value of $\sum$ the stability time decreases. The $\sum$ value acts as a potential that affects the kinetic of the evolution process. However, as mentioned before, this normalized time is different from the computation time needed for the program to find the stability point. Accordingly, in most of the cases by increasing the $\sum$ value, the computation time increases.

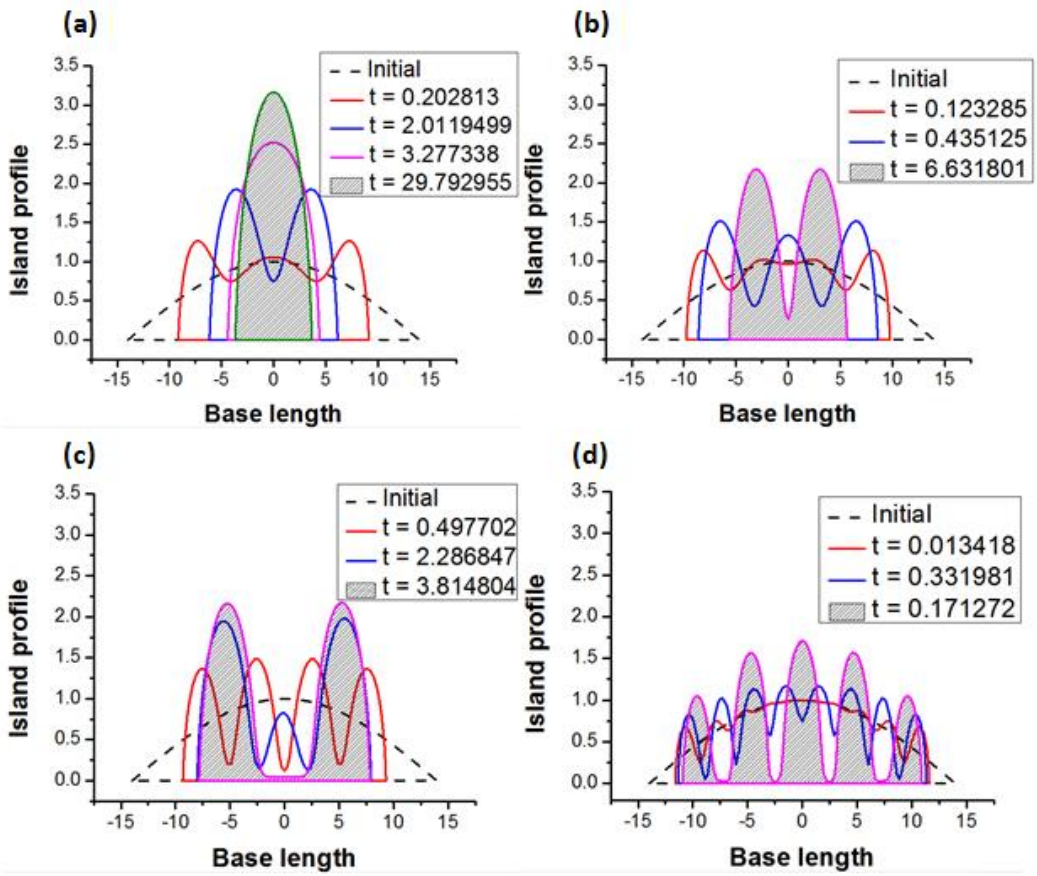


Figure 4.14: The effects of stress on the morphological evolution of quantum dots for (a) $\sum = 0.5$, (b) $\sum = 0.7$, (c) $\sum = 1$, (d) $\sum = 2$; in these simulations, the input parameters are$\beta = 28,\ \bar{M}_{long} = 1,\ \theta = 75^{\circ}\ ,\ f_f = 1,\ f_s = 1.2$

The effect of the aspect ratio of the initial droplet is another subject we concentrated within the scope of this thesis. To this end, we have carried out various experiments with different initial droplet sizes while keeping the applied stress as $\sum = 1$. The morphological evolution recorded for these experiments is summarized in Figure 4.15. By increasing the aspect ratio, island would have larger base length, which makes relatively shallower profiles. In longer base length islands, the island surface has the capacity to form more initial sinusoidal wave shapes as a result of the stress relaxation. Accordingly, the islands with larger aspect ratio have the chance to form more in number distinct islands.

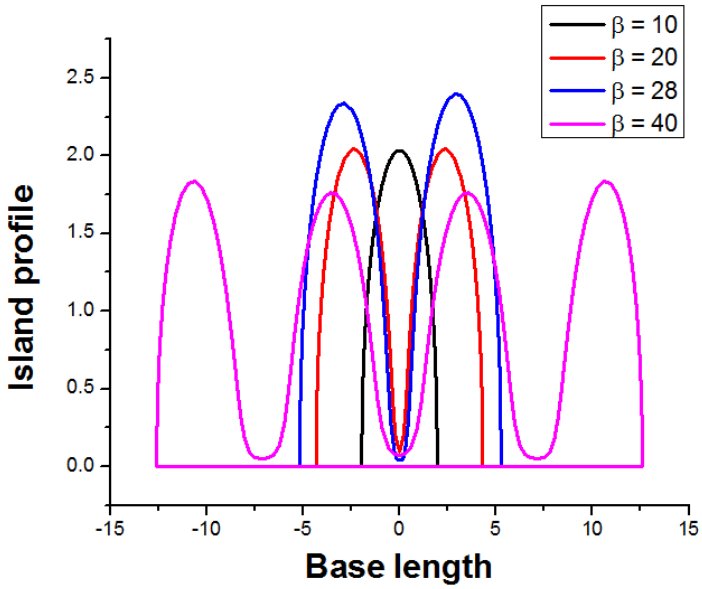


Figure 4.15: The effect of aspect ratio on the morphological evolution of quantum dots, where in these simulations, the input parameters are $\bar{M}_{long} = 1, \sum = 1,\ \theta = 89^{\circ}\ ,\ f_f = 1,\ f_s = 1.2$

According to Figure 4.15, for small aspect ratios such as ($\beta = 10$), one single island is the stable final configuration. The evolution process of this specific case is given in Figure 4.16.a. In this case, there are not much ripples formed on the droplet surface at early times. Thus, those ripples coalesce into one single island. On the other hand, we observed that by increasing the aspect ratio, the number of final stable islands increases.

At the initial evolution stages of island with large aspect ratio ($\beta = 40$), the surface instability generates several surface ripples where any of them may have the potential to form final quantum dots. This is the case we observed in Figure 4.16.b.

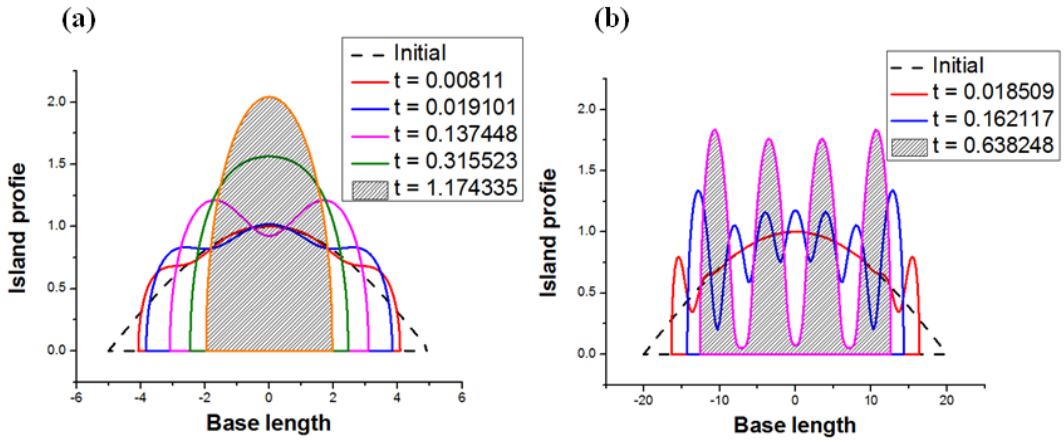


Figure 4.16: The effect of aspect ratio on the morphological evolution of quantum dots for (a) $\beta = 10$, (b) $\beta = 40$ where in these simulations, the input parameters are $\bar{M}_{long} = 1, \sum = 1,\ \theta = 89^{\circ}\ ,\ f_f = 1,\ f_s = 1.2$

As we have demonstrated above, both the island stability and the configuration of the stable islands is a function of various material properties especially the surface energies of the island and the substrate and the misfit stress levels in the system. To be able to provide a more clear view, we prepared a phase diagram (Figure 4.17) depicting the number of stable islands with respect to both the wetting parameter, λ and the stress parameter, $\sum$. In Figure 4.17, the different regions for island stability, which are determined according to island numbers of performed experiments in various values of $\sum$ and λ are shown. In this figure, we colored those different stability regions and also draw approximate boundaries between those regions for clarity.

When we carefully inspect Figure 4.17, it can be seen that for small values of $\sum$, independent of the λ values, we have single islands; this is in accord with our results for no or negligible stress levels presented in Section 4.3.1. However, as the stress level, $\sum$, increases, we start seeing island fragmentation and more islands as the stable final configurations.

The threshold value of the stress level necessary for fragmentation is a function of wetting parameter as clearly observable in Figure 4.17. For example for λ values closer to zero this threshold value is around $\Sigma = 0.6$, however for $\lambda = 0.7$, the threshold value increases up to $\Sigma = 1.2$.

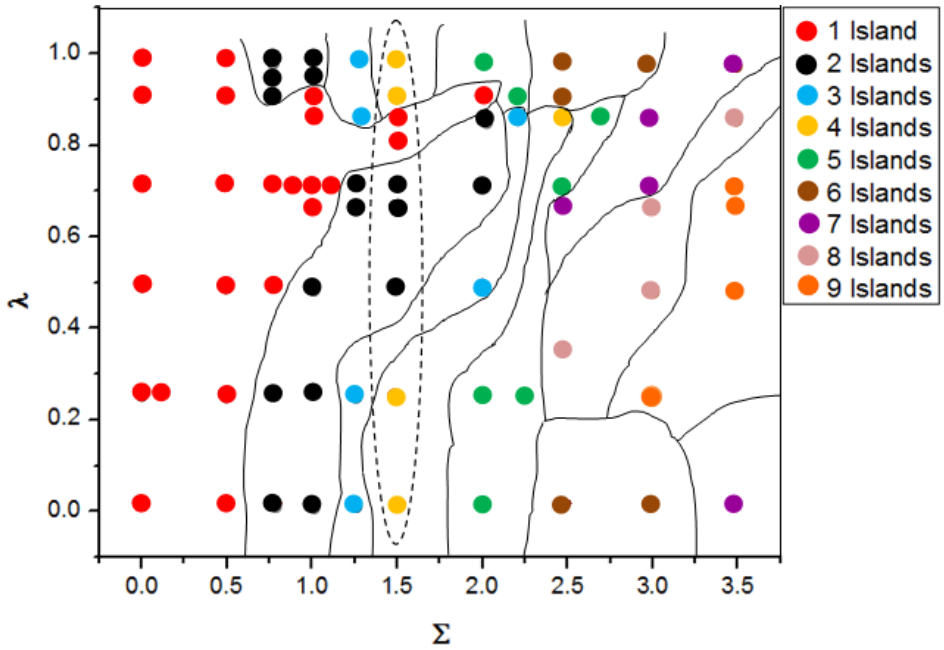


Figure 4.17: The phase diagram of various regions for different $\sum$ and λ values

There is also another order vertically for each $\sum$ value. For example, in $\sum = 1$ (limited area in Figure 4.17 with dashed line), this order is obvious. The profiles of this set of experiments were also given previously in Figure 4.11, where for small value of λ, quadruplet island formation was observed. By increasing the λ value, the number of islands increases. However, at a critical value, the island number starts to decrease and finally near $\lambda \approx 1$, the final morphology forms a quadruplet profile. The obvious difference between the small and large λ value quadruplet profiles is the wetting layer formation as shown in Figure 4.11.

The wetting layer formation is very important and it is the main difference between isolated island formations (Volmer-Weber) versus connected island formations (Stranski-Krastanow). When we more accurately inspect the phase diagram developed in this work, it seems to have three distinct regions: lower part ($\lambda<0.3$); middle part ($0.3<\lambda<0.9$) and top part ($0.9<\lambda$) that are separated with dashed lines as shown in Figure 4.18. Within each one of these regions, the number of islands at equilibrium increases as $\sum$ increases.

Besides the number of stable islands formed, the morphology of the islands is also important. To better understand the morphological differences within those regions and evolution process of each region; two different set of experiments have been chosen.

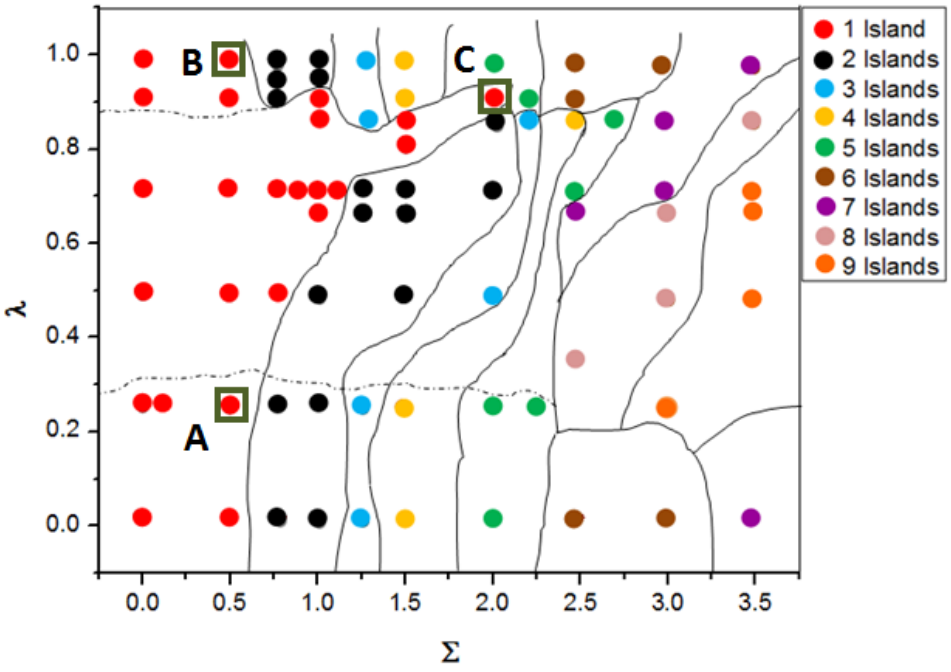


Figure 4.18: The phase diagram of various regions for different, $\sum$ and λ values

The first set is associated with the single island number (low stress region), where the selected experiments are indicated in Figure 4.18 with green squares. The evolution processes of these selected experiments, A, B and C, respectively for lower, middle and upper parts are shown in Figure 4.19.

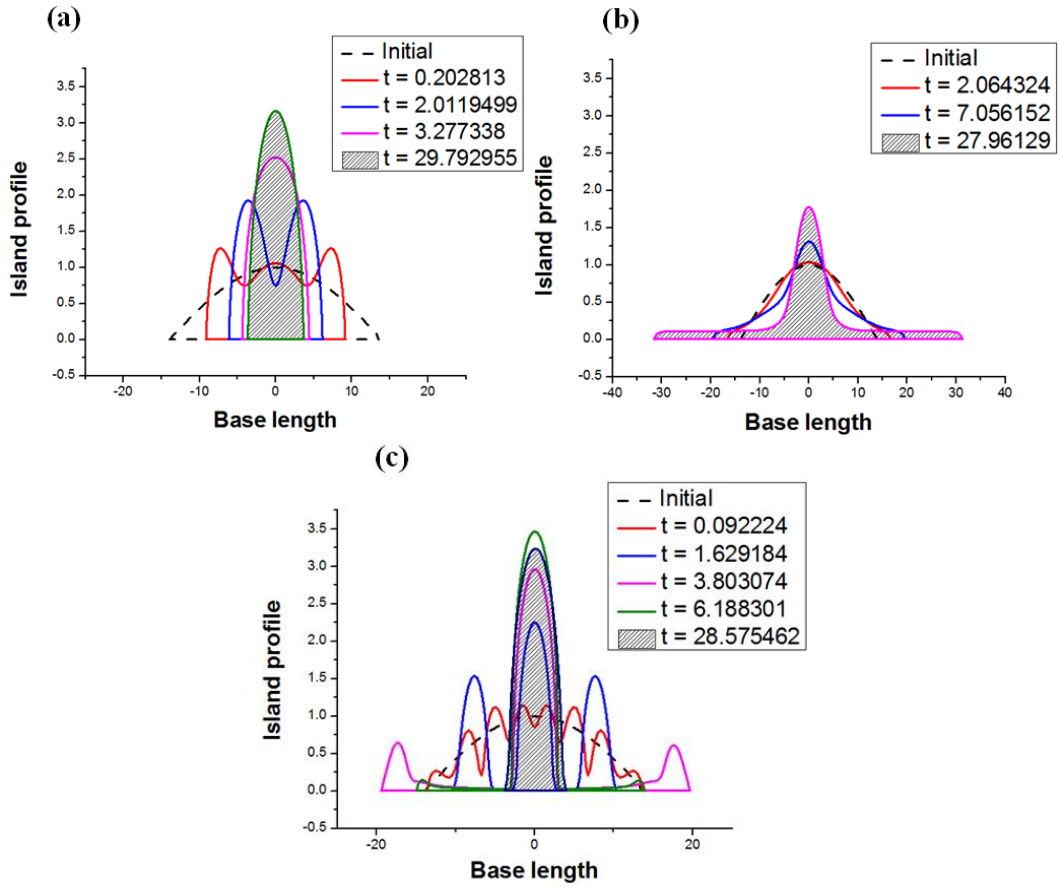


Figure 4.19: The evolution processes of single island region in phase diagram. In these simulations, the input parameters are (a) $\beta = 28,\ \lambda = 0.259,\ \sum = 0.5$, (b) $\beta = 28,\ \lambda = 0.990,\ \sum = 0.5$, (c) $\beta = 28,\ \lambda = 0.866,\ \sum = 1.5$.

As seen from Figure 4.19, all three cases A, B and C evolves into a single island but with different routes. First of all, case A from lower part directly evolves into a single isolated droplet (Figure 4.19.a). At high values of λ, B from upper part (Figure 4.19.b), we come up with a single island but with a well formed wetting layer. An interesting circumstance happens at middle region of λ that is for

experiment C (Figure 4.19.c): In this case, the droplet is separated into three distinct islands at the early times, and all are connected with wetting layers. The island in the middle is larger than the other two. The islands on each side are trying to move away further towards the sides. The energy balance between the surface free energies and interface energy are favorable for this phenomenon. However, it should be noticed that the size of moving islands decreases as the islands move further. The larger island in the middle gets larger by gaining the lost material from the side islands. At later times, at a critical time point, when the sizes of the satellite islands become smaller than a critical value, they no longer move away from the larger middle island but towards it. Later, the larger island dominates and completely consumes the material content of the smaller ones to form a single island with a well-defined equilibrium wetting angle.

The second set of experiments is chosen among the stable quadruplet islands from moderate to high stress regions. Similar to the previous set, A, B and C experiments, respectively for lower, middle and upper parts are shown in Figure 4.20.

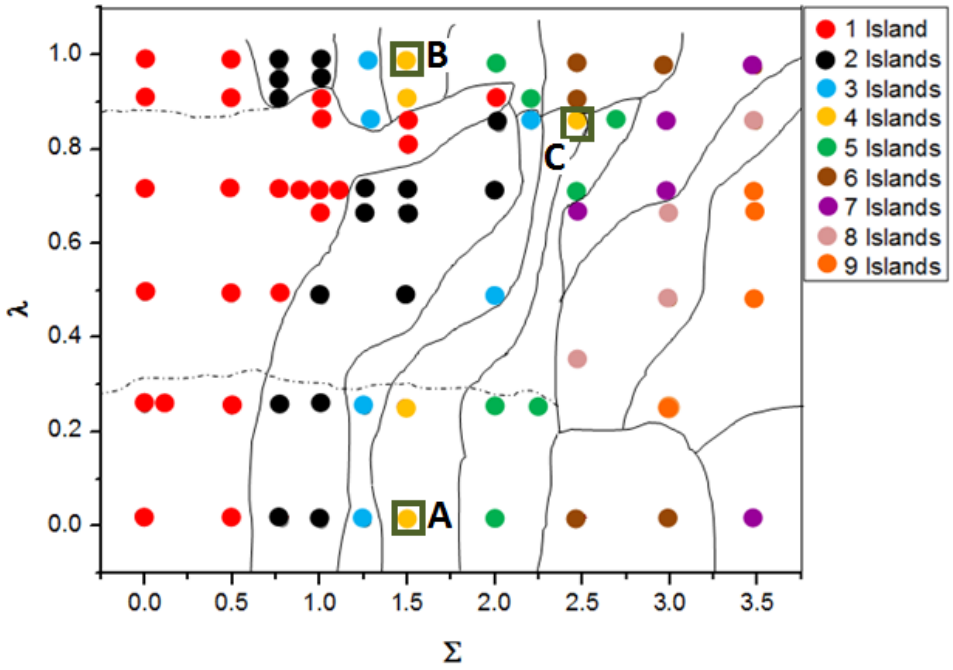


Figure 4.20: The phase diagram of various regions for different ∑ and λ values

The similar phenomena happens as seen from Figure 4.21, where all three cases A, B and C evolves into a quadruple island but with different routes.

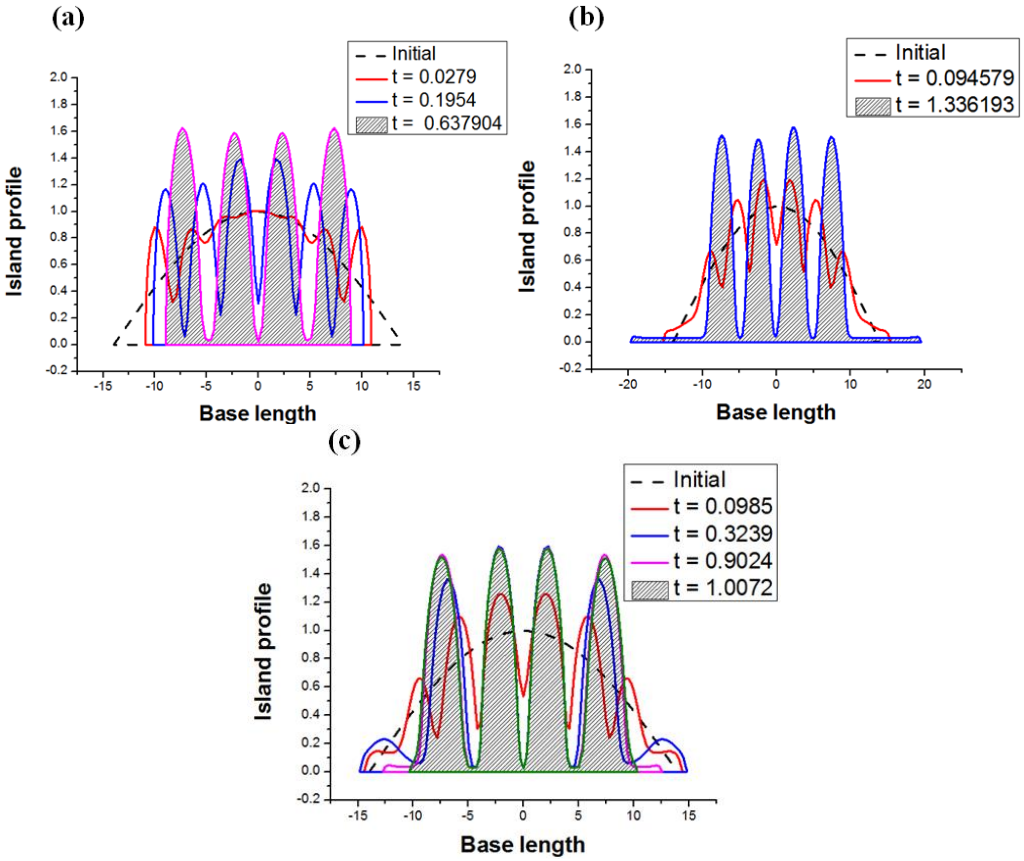


Figure 4.21: The evolution processes of quadruplet island region in phase diagram. In these simulations, the input parameters are (a) $\beta = 28, \lambda = 0.017, \sum = 1.5$, (b)$\beta = 28, \lambda = 0.990, \sum = 1.5$, (c) $\beta = 28, \lambda = 0.866, \sum = 2.5$

As can be seen from Figure 4.21, all three cases A, B and C evolve into a quadruplet islands but with different routes. First of all, case A from lower part directly evolves into a quadruplet islands without the wetting layer formation in both sides (Figure 4.21.a). At high values of λ, B from upper part (Figure 4.21.b), we come up with a very similar formation but with a well formed wetting layer in both sides. An interesting circumstance happens at middle region of λ that is for experiment C (Figure 4.21.c): In this case, the side islands move away to a certain point and again drift back to the middle and form a quadruplet island form with the equilibrium wetting angle.

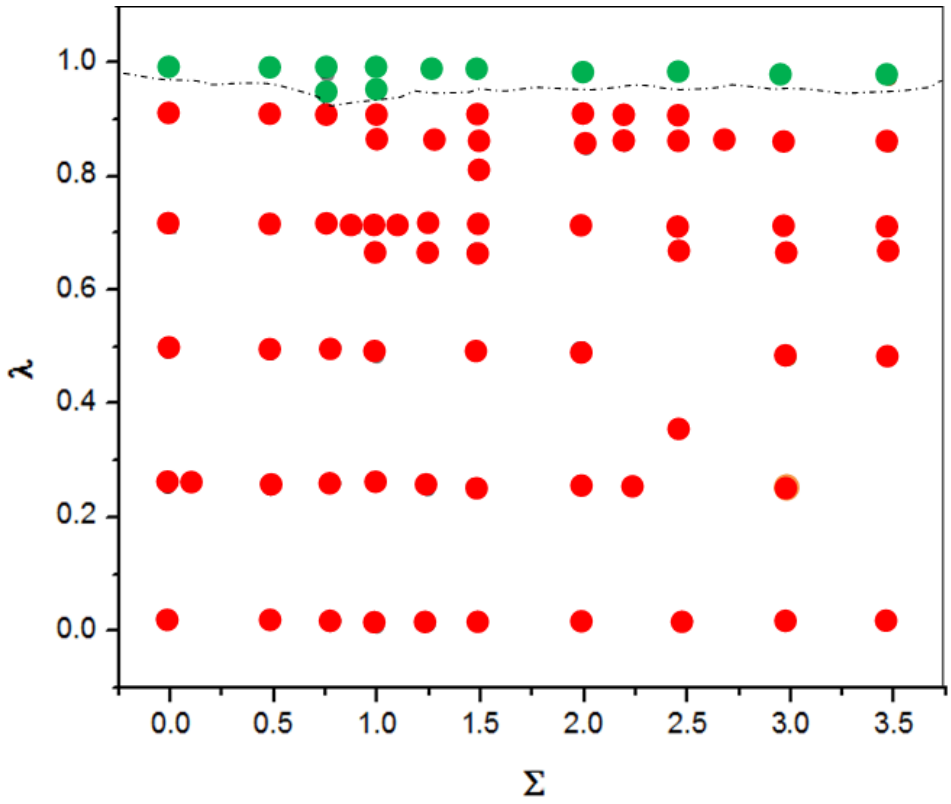


Figure 4.22: The phase diagram for different ∑ and λ values that separates the region with wetting layer and region without wetting layer

As mentioned previously, the formation of wetting layer at the edges of the quantum dots has a very significant importance for our studies, as they make quantum dots suitable for many applications. Accordingly, in figure 4.22 we separated the phase diagram with dashed lines into two regions to demonstrate the region that wetting layer formation is observed and the region without wetting layer. Here, green balls are associated to the experiments that wetting layer is formed and the rest are associated to the experiments without formation of wetting layers. It is obvious that at high values of λ independent of ∑ values we can gain wetting layers.

CHAPTER 5

CONCLUSIONS

Semi-conductor quantum dots show a significant potential in wide range of applications in technology. One of the most important aspects of quantum dots is the relationship between their size/morphology and electronic and photonic properties. Considering this relationship, understanding the mechanism of the formation of these nano particles and thus controlling the size, morphology and even organization is a very important technological and scientific problem.

Our study is based on the new model introduced by Ogurtani and Oren (2001; 2005), using continuum level dynamical simulations presented for the spontaneous evolution of an isolated thin solid droplet on a rigid substrate via computer simulation method.

Here, in droplet simulation experiments, the effects of film aspect ratio, equilibrium angle between the droplet and the substrate (surface energy difference dependent) and stress on the morphological evolution of QDs and occurrence of wetting layer are investigated in detail.

The morphological evolution kinetic of an isochoric surface in the direction of the surface normal, is governed by normalized and scaled velocity vector (equation 2.1) and the velocity associated with the edges between substrate and droplet (triple junction) governed by (Equation 2.2) are called V_{ord} and V_{edge} , respectively.

The simulation is established for a 2-D system. At the initial stage, the film surface is defined by a symmetrical halve-wave length Cosine-function using finite nodes with specific segment length (Figure 2.1). The evolution of the surface occurs by displacement of the nodes and changing the node coordinates in 2D. The governing equations for displacement (equation 2.1 and equation 2.2) are solved using numerical procedure. The differential equations are solved using Euler finite difference method.

To avoid the appearance of very short or very long segment lengths, the segment lengths should be controlled persistently and be able to be revised if required (remeshing). The normal vector and curvature of each node should be calculated after each run step.

Finally, the velocity of each node is calculated using the governing equations. By applying the velocities, the small displacement of the nodes makes the final configuration of the system.

By applying all the calculations on the final configuration, the morphological evolution of the system continues through time. C++ program is used to prepare the numerical procedure. The C++ code is given in the Appendix. The numerical procedure is also summarized schematically in Figure 3.6.

According to our definition ($\beta = 2L/\bar{h}_p$), by altering the β, we can change the initial shape of the droplet, since the $\bar{h}_p$ value is invariant and it is equal to one, so by decreasing the β, we actually decrease the width of the droplet, which finally forms a smaller quantum dot. At the initial evolution stages of island with large aspect ratio, the surface instability makes surface hills, which happen due to internal stress. Any of these hills may have the potential to form a quantum dot. However, for islands with small aspect ratio there are not much hills at the initial configuration.

To observe the effect of wetting parameter on the equilibrium morphology, the behavior of the system under the effect of different wetting parameters λ is investigated. As we mentioned before, there is a relationship between equilibrium contact angle θ and λ wetting parameter as follows: $\theta = \mathrm{acos}(\lambda)$. During the surface evolution, the wetting angle changes through time where at the final stages it reaches a specific value and stays constant. If the wetting contact angle declines to an aspect ratio dependent critical value, the system is observed to form a wetting layer. This value is also a function of the internal stress of the system.

The effect of stress on the system with isotropic properties, show that applying small values of stress cause a limited change on the configuration behavior. For low stress values, the final morphology shows a slight increment in the aspect ratio of the system. After reaching the stress to a critical level, the morphological behavior of the system changes entirely and the initial single droplet is divided in to two or more separated islands, which are connected with a thin wetting layer in between. This final configuration is called Stranski-Krastanow islands in the literature.

To discover the internal stress and wetting parameter effects on the morphology, various experiments are performed by changing both parameters separately. This

group of experiments is given in Figure 4.17 as a phase diagram. This diagram is beneficial in determination of the stability regions of the islands that show in which region the droplet is going to separate into smaller islands and the island number in the case of separation. This phase diagram denotes the complexity of the system. In this complex system, the curves that divide the regions are noticeable. There is also another categorizing manner where divides the phase diagram into three parts of low and high wetting contact angles and the middle region in between that has the intermediate value of wetting contact angle value. In each region, by increasing the stress, the number of islands increases. All three regions show separately but similar behavior. This diagram can provide beneficial information about essential conditions required for production of specific QD. After applying essential tests to verify the validity of the results, the following results is gained from the set of experiments done during this project.

The internal stress of the material and the wetting potential, have the effect of dividing the material into islands and forming the wetting layer respectively. In order for the wetting layer to be formed, there is a critical value for wetting potential where beyond, the wetting layer appears. In the domain of low wetting potential, the wetting layer only appears between islands. However, for high values of the wetting potential, the wetting layers form at the outer edges of the islands as well as islands interval.

Consequently, according to the results of various experiments, the surface evolution of the thin films depends on the internal stress values, the equilibrium wetting contact angle between the film/substrate, surface stiffness and initial conditions of the system. This information is essential for providing the scientific fundamental knowledge for novel fabrication techniques of quantum dots.

5.1. Future studies

The performed experiments in the scope of this project revealed essential information about the evolution behavior of thin films under stress and the formation of the quantum dots. It is also important here to mention the missing points of this project, which could be survey issues in the future.

This project indicates the effect of the internal stress on the morphological control of the quantum dots. In future, we can add the electromigration effects (the effect of the electrical field on the surface diffusion of the material) to our 2D program. The information in this issue can be helpful for island formation using electrical field as this technique can be easily applied.

All of the calculations in the scope of this project are based on the assumption that all the material properties are isotropic. However, in the real world the material properties related the surface diffusivity and also the surface energy are anisotropic, which could be considered in further studies.

Investigating the transition paths from initial droplet shape to the final configuration through time may give us beneficial knowledge about effective factors in minimizing the energy required for transition and also information about controlling the morphology of QDs.

Another valuable future study proposition is to develop a three dimensional model, which can indicate more precise information about the real world. Moreover, in 3D equilibrium systems, the information about shape, size, internal stress distribution etc. can be used to calculate the QD optical and electrical properties. In this issue,

after obtaining the final morphology, the calculations about energy levels, wave functions and optical dipole matrix elements can be performed to simulate the electrical transport properties and photo excited carriers of the Quantum dot arrays.

REFERENCES

[1] V. I. Klimov, et al. (2000). "Optical Gain and Stimulated Emission in Nanocrystal Quantum Dots." Science 290(5490): 314-317.

[2] P. Alivisatos (2004). "The use of nanocrystals in biological detection." Nature Biotechnology 22(1): 47-52.

[3] X. Gao, L. Yang, J. A. Petros, F. F. Marshall, J. W. Simons and S. Nie (2005). "In vivo molecular and cellular imaging with quantum dots." Current Opinion in Biotechnology 16(1): 63-72.

[4] H. Ahmed (2002). "Novel nanodevices for electronics: fabrication and characteristics." Microelectronic Engineering 61–62(0): 3-4.

[5] O. Stier, et al. (1999). "Electronic and optical properties of strained quantum dots modeled by 8-band k·p theory." Physical Review B 59(8): 5688-5701.

[6] M. S. Skolnick and D. J. Mowbray (2004). "Self-assembled semiconductor quantum dots: Fundamental Physics and Device Applications." Annual Review of Materials Research 34(1): 181-218.

[7] M. Kroutvar (2004). "Optically programmable electron spin memory using semiconductor quantum dots." Nature 432(7013): 81-84

[8] D. Wang, T. Xie and Y. Li (2009). "Nanocrystals: Solution-based synthesis and applications as nanocatalysts." Nano Research 2(1): 30-46.

[9] D. V. Talapin, J. S. Lee, M. V. Kovalenko and E. V. Shevchenko (2009). "Prospects of Colloidal Nanocrystals for Electronic and Optoelectronic Applications." Chemical Reviews 110(1): 389-458.

[10] P. Reiss, M. Protière and L. Li (2009). "Core/Shell Semiconductor Nanocrystals." Small 5(2): 154-168.

[11] W. W. Haixiong Ge, Zhiwei Li, Jizong Zhang, Yiming Shen, Changsheng Yuan and Yanfeng Chen (2013). "Nanopatterning highly curved surfaces using hybrid nanoimprint lithography." SPIE Newsroom: 1-3

[12] V. Nandwana, C. Subramani, Y.-C. Yeh, B. Yang, S. Dickert, M. D. Barnes, M. T. Tuominen and V. M. Rotello (2011). "Direct patterning of quantum dot nanostructuresviaelectron beam lithography." Journal of Materials Chemistry 21(42): 16859-16862

[13] M. Helfrich, P. Schroth, D. Grigoriev, S. Lazarev, R. Felici, T. Slobodskyy, T. Baumbach and D. M. Schaadt (2012). "Growth and characterization of site-selective quantum dots." physica status solidi (a) 209(12): 2387-2401.

[14] K. Shou-Yi, C. Wei-Chun, L. Fang-l, L. Woei-Tyng and H. Chien-Nan (2011). Effect of growth parameters on surface morphology evolution of MOMBE-grown InN. Nanoelectronics Conference (INEC), 2011 IEEE 4th International.1-2

[15] P.-Y. Hsiao, Z.-H. Tsai, J.-H. Huang and G.-P. Yu (2009). "Strong asymmetric effect of lattice mismatch on epilayer structure in thin-film deposition." Physical Review B 79(15): 155414.

[16] J. Gao, W. Jie, Y. Yuan, T. Wang, G. Zha and J. Tong (2011). "Dependence of film texture on substrate and growth conditions for CdTe films deposited by close-spaced sublimation." Journal of Vacuum Science & Technology A: Vacuum, Surfaces, and Films 29(5): 051507.

[17] M. H. Grabow and G. H. Gilmer (1988). "Thin film growth modes, wetting and cluster nucleation." Surface Science 194(3): 333-346.

[18] D. Leonard, et al. (1994). "Critical layer thickness for self-assembled InAs islands on GaAs." Physical Review B 50(16): 11687-11692.

[19] D. J. Eaglesham and M. Cerullo (1990). "Dislocation-free Stranski-Krastanow growth of Ge on Si(100)." Physical Review Letters 64(16): 1943-1946.

[20] A. Madhukar, K. C. Rajkumar, L. Chen, S. Guha, K. Kaviani and R. Kapre (1990). "Realization of low defect density, ultrathick, strained InGaAs/GaAs multiple quantum well structures via growth on patterned GaAs (100) substrates." Applied Physics Letters 57(19): 2007-2009.

[21] D. Leonard, M. Krishnamurthy, C. M. Reaves, S. P. Denbaars and P. M. Petroff (1993). "Direct formation of quantum-sized dots from uniform coherent islands of InGaAs on GaAs surfaces." Applied Physics Letters 63(23): 3203-3205.

[22] Y. Mo, D. E. Savage, B. S. Swartzentruber and M. G. Lagally (1990). "Kinetic pathway in Stranski-Krastanov growth of Ge on Si(001)." Phys Rev Lett 65(8): 1020-1023.

[23] Y. Zhang, et al. (2002). "Evolution of Ge/Si(100) island morphology at high temperature." Applied Physics Letters 80(19): 3623-3625.

[24] D. E. Jesson, K. M. Chen, S. J. Pennycook, T. Thundat and R. J. Warmack (1996). "Morphological evolution of strained films by cooperative nucleation." Physical Review Letters 77(7): 1330-1333.

[25] Y. Obayashi and K. Shintani (1998). "Directional dependence of surface morphological stability of heteroepitaxial layers." Journal of Applied Physics 84(6): 3141-3146.

[26] Y. W. Zhang and A. F. Bower (2001). "Three-dimensional analysis of shape transitions in strained-heteroepitaxial islands." Applied Physics Letters 78(18): 2706-2708

[27] A. A. Golovin, M. S. Levine, T. V. Savina and S. H. Davis (2004). "Faceting instability in the presence of wetting interactions: A mechanism for the formation of quantum dots." Physical Review B 70(23): 235342.

[28] T. Wiebach, M. Schmidbauer, M. Hanke, H. Raidt, R. Köhler and H. Wawra (2000). "Strain and composition in SiGe nanoscale islands studied by x-ray scattering." Physical Review B 61(8): 5571-5578.

[29] O. G. Schmidt, S. Kiravittaya, Y. Nakamura, H. Heidemeyer, R. Songmuang, C. Müller, N. Y. Jin-Phillipp, K. Eberl, H. Wawra, S. Christiansen, H. Gräbeldinger and H. Schweizer (2002). "Self-assembled semiconductor nanostructures: climbing up the ladder of order." Surface Science 514(1–3): 10-18.

[30] V. A. Shchukin and D. Bimberg (1999). "Spontaneous ordering of nanostructures on crystal surfaces." Reviews of Modern Physics 71(4): 1125-1171.

[31] J. Stangl, V. Holý and G. Bauer (2004). "Structural properties of self-organized semiconductor nanostructures." Reviews of Modern Physics 76(3): 725-783.

[32] S. Kiravittaya, R. Songmuang, A. Rastelli, H. Heidemeyer and O. G. Schmidt (2006). "Multi-scale ordering of self-assembled InAs/GaAs(001) quantum dots." Nanoscale Research Letters 1(1): 1-10.

[33] H. Lee, J. A. Johnson, J. S. Speck and P. M. Petroff (2000). "Controlled ordering and positioning of InAs self-assembled quantum dots." Journal of Vacuum Science & Technology B 18(4): 2193-2196.

[34] A. Pascale, I. Berbezier, A. Ronda and P. C. Kelires (2008). "Self-assembly and ordering mechanisms of Ge islands on prepatterned Si(001)." Physical Review B 77(7): 075311.

[35] D. J. Srolovitz (1989). "On the stability of surfaces of stressed solids." Acta Metallurgica 37(2): 621-625.

[36] H. Gao and W. D. Nix (1999). "Surface roughening of heteroepitaxial thin films." Annual Review of Materials Science 29(1): 173-209.

[37] H. T. Johnson and L. B. Freund (1997). "Mechanics of coherent and dislocated island morphologies in strained epitaxial material systems." Journal of Applied Physics 81(9): 6081-6090.

[38] A. Oral and R. Ellialtioglu (1995). "Initial stages of SiGe epitaxy on Si(001) studied by scanning tunneling microscopy." Surface Science 323(3): 295-303.

[39] H. Heidemeyer, S. Kiravittaya, C. Müller, N. Y. Jin-Phillipp and O. G. Schmidt (2002). "Closely stacked InAs/GaAs quantum dots grown at low growth rate." Applied Physics Letters 80(9): 1544-1546.

[40] R. Songmuang, S. Kiravittaya and O. G. Schmidt (2003). "Formation of lateral quantum dot molecules around self-assembled nanoholes." Applied Physics Letters 82(17): 2892-2894.

[41] Q. Sun, C. D. Yerino, B. Leung, J. Han and M. E. Coltrin (2011). "Understanding and controlling heteroepitaxy with the kinetic Wulff plot: A case study with GaN." Journal of Applied Physics 110(5): 053517.

[42] K. Hiramatsu, K. Nishiyama, M. Onishi, H. Mizutani, M. Narukawa, A. Motogaito, H. Miyake, Y. Iyechika and T. Maeda (2000). "Fabrication and characterization of low defect density GaN using facet-controlled epitaxial lateral overgrowth (FACELO)." Journal of Crystal Growth 221(1–4): 316-326.

[43] J. G. Belk, D. W. Pashley, C. F. McConville, B. A. Joyce and T. S. Jones (1998). "Surface morphology during strain relaxation in the growth of InAs on GaAs(110)." Surface Science 410(1): 82-98.

[44] X. L. Li (2010) "Thermodynamic analysis on the stability and evolution mechanism of self-assembled quantum dots." Applied Surface Science 256(12): 4023-4026.

[45] N. V. Medhekar, V. Hegadekatte and V. B. Shenoy (2008). "Composition Maps in Self-Assembled Alloy Quantum Dots." Physical Review Letters 100(10): 106104.

[46] X. B. Niu, et al. (2011). "Nonequilibrium Composition Profiles of Alloy Quantum Dots and their Correlation with the Growth Mode." Physical Review Letters 107(7): 076101.

[47] V. B. Shenoy (2011). "Evolution of morphology and composition in three-dimensional fully faceted strained alloy crystals." Journal of the Mechanics and Physics of Solids 59(5): 1121-1130.

[48] M. V. Shaleev, A. V. Novikov, D. V. Yurasov, J. M. Hartmann, O. A. Kuznetsov, D. N. Lobanov and Z. F. Krasilnik (2013). "Transition from the two- to three-dimensional growth of Ge films upon deposition onto relaxed SiGe/Si(001) buffer layers." Semiconductors 47(3): 427-432.

[49] I. Mukhametzhanov, Z. Wei, R. Heitz and A. Madhukar (1999). "Punctuated island growth: An approach to examination and control of quantum dot density, size, and shape evolution." Applied Physics Letters 75(1): 85-87.

[50] G. Medeiros-Ribeiro, A. M. Bratkovski, T. I. Kamins, D. A. A. Ohlberg and R. S. Williams (1998). "Shape transition of germanium nanocrystals on a silicon (001) surface from pyramids to domes." Science 279(5349): 353-355.

[51] J. Márquez, L. Geelhaar and K. Jacobi (2001). "Atomically resolved structure of InAs quantum dots." Applied Physics Letters 78(16): 2309-2311

[52] G. Costantini, A. Rastelli, C. Manzano, P. Acosta-Diaz, G. Katsaros, R. Songmuang, O. G. Schmidt, H. v. Känel and K. Kern (2005). "Pyramids and domes in the InAs/GaAs(0 0 1) and Ge/Si(0 0 1) systems." Journal of Crystal Growth 278(1–4): 38-45.

[53] Y. Suzuki, T. Kaizu and K. Yamaguchi (2004). "Controlled stacking growth of uniform InAs quantum dots by molecular beam epitaxy." Physica E: Low-dimensional Systems and Nanostructures 21(2–4): 555-559.

[54] V. M. Ustinov, E. R. Weber, S. Ruvimov, Z. Liliental-Weber, A. E. Zhukov, A. Y. Egorov, A. R. Kovsh, A. F. Tsatsul'nikov and P. S. Kop'ev (1998).

"Effect of matrix on InAs self-organized quantum dots on InP substrate." Applied Physics Letters 72(3): 362-364.

[55] A. Rastelli, H. Von Känel, B. J. Spencer and J. Tersoff (2003). "Prepyramid-to-pyramid transition of SiGe islands on Si(001)." Physical Review B 68(11): 115301.

[56] F. Montalenti, P. Raiteri, D. B. Migas, H. von Känel, A. Rastelli, C. Manzano, G. Costantini, U. Denker, O. G. Schmidt, K. Kern and L. Miglio (2004). "Atomic-Scale Pathway of the Pyramid-to-Dome Transition during Ge Growth on Si(001)." Physical Review Letters 93(21): 216102.

[57] J.-M. Baribeau, X. Wu, N. L. Rowell and D. J. Lockwood (2006). "Ge dots and nanostructures grown epitaxially on Si." Journal of Physics: Condensed Matter 18(8): R139.

[58] B. J. Spencer and J. Tersoff (2013). "Symmetry breaking in shape transitions of epitaxial quantum dots." Physical Review B 87(16): 161301.

[59] L. Geelhaar, Y. Temko, J. Márquez, P. Kratzer and K. Jacobi (2002). "Surface structure of GaAs(2 5 11)." Physical Review B 65(15): 155308.

[60] J. Márquez, L. Geelhaar and K. Jacobi (2000). "Atomic structure of the GaAs($1\bar{1}\bar{3}$)B(8×1) surface reconstruction." Physical Review B 62(15): 9969-9972.

[61] Y. Temko, et al. (2003). "InAs quantum dots grown on the GaAs ($1\bar{1}\bar{3}$) B surfaces: a comperative STM study." Physical Review B 68(16): 165310.

[62] H. Lee, R. Lowe-Webb, W. Yang and P. C. Sercel (1998). "Determination of the shape of self-organized InAs/GaAs quantum dots by reflection high energy electron diffraction." Applied Physics Letters 72(7): 812-814.

[63] F. Shahedipour-Sandvik, J. Grandusky, A. Alizadeh, C. Keimel, S. P. Ganti, S. T. Taylor, S. F. LeBoeuf and P. Sharma (2005). "Strain dependent facet

stabilization in selective-area heteroepitaxial growth of GaN nanostructures." Applied Physics Letters 87(23): 233108.

[64] S.C. Li, et al. (2006). "Determination of the Ehrlich-Schwoebel barrier in epitaxial growth of thin films." Physical Review B 74(19): 195428.

[65] S. Kiravittaya, Y. Nakamura and O. G. Schmidt (2002). "Photoluminescence linewidth narrowing of InAs/GaAs self-assembled quantum dots." Physica E: Low-dimensional Systems and Nanostructures 13(2–4): 224-228.

[66] Y. Nakata, K. Mukai, M. Sugawara, K. Ohtsubo, H. Ishikawa and N. Yokoyama (2000). "Molecular beam epitaxial growth of InAs self-assembled quantum dots with light-emission at 1.3 μm." Journal of Crystal Growth 208(1–4): 93-99.

[67] A. Rastelli, E. Muller and H. von Kanel (2002). "Shape preservation of Ge/Si(001) islands during Si capping." Applied Physics Letters 80(8): 1438-1440.

[68] V. Schmidt, P. C. McIntyre and U. Gösele (2008). "Morphological instability of misfit-strained core-shell nanowires." Physical Review B 77(23): 235302.

[69] H. Wang, et al. (2008). "Morphology of Epitaxial Core−Shell Nanowires." Nano Letters 8(12): 4305-4311.

[70] S. Kwon, et al. (2012). "Misfit-Guided Self-Organization of Anticorrelated Ge Quantum Dot Arrays on Si Nanowires." Nano Letters 12(9): 4757-4762.

[71] J. L. Taraci, et al. (2005). "Strain mapping in nanowires." Nanotechnology 16(10): 2365.

[72] Hsu P. S. Taraci, E. C. Young, A. E. Romanov, K. Fujito, S. P. DenBaars, S. Nakamura and J. S. Speck (2011). "Misfit dislocation formation via pre-existing threading dislocation glide in (112¯2) semipolar heteroepitaxy." Applied Physics Letters 99(8): 081912.

[73] A. Bourret, et al. (2001). "Strain relaxation in (0001) AlN/GaN heterostructures." Physical Review B 63(24): 245307.

[74] D. Zubia and S. D. Hersee (1999). "Nanoheteroepitaxy: The Application of nanostructuring and substrate compliance to the heteroepitaxy of mismatched semiconductor materials." Journal of Applied Physics 85(9): 6492-6496.

[75] I. A. Goldthorpe, A. F. Marshall and P. C. McIntyre (2008). "Synthesis and Strain Relaxation of Ge-Core/Si-Shell Nanowire Arrays." Nano Letters 8(11): 4081-4086.

[76] H. Katsuno, M. Uwaha and Y. Saito (2008). "Heteroepitaxial growth modes with dislocations in a two-dimensional elastic lattice model." Surface Science 602(22): 3461-3466.

[77] C. G. Gamage and Z.-F. Huang (2013). "Nonlinear dynamics of island coarsening and stabilization during strained film heteroepitaxy." Physical Review E 87(2): 022408.

[78] C. Ratsch and A. Zangwill (1993). "Equilibrium theory of the Stranski-Krastanov epitaxial morphology." Surface Science 293(1–2): 123-131.

[79] C. Priester and M. Lannoo (1995). "Origin of Self-Assembled Quantum Dots in Highly Mismatched Heteroepitaxy." Physical Review Letters 75(1): 93-96.

[80] L. G. Wang, et al. (2000). "Size, shape, and stability of InAs quantum dots on the GaAs(001) substrate." Physical Review B 62(3): 1897-1904.

[81] K. Kassner, C. Misbah, J. Müller, J. Kappey and P. Kohlert (2001). "Phase-field modeling of stress-induced instabilities." Physical Review E 63(3): 036117.

[82] J. Müller and M. Grant (1999). "Model of Surface Instabilities Induced by Stress." Physical Review Letters 82(8): 1736-1739.

[83] J. J. Eggleston and P. W. Voorhees (2002). "Ordered growth of nanocrystals via a morphological instability." Applied Physics Letters 80(2): 306-308.

[84] S. M. Wise, J. S. Lowengrub, J. S. Kim and W. C. Johnson (2004). "Efficient phase-field simulation of quantum dot formation in a strained heteroepitaxial film." Superlattices and Microstructures 36(1–3): 293-304.

[85] F. Long, S. P. A. Gill and A. C. F. Cocks (2001). "Effect of surface-energy anisotropy on the kinetics of quantum dot formation." Physical Review B 64(12): 121307.

[86] P. Liu, Y. W. Zhang and C. Lu (2003). "Formation of self-assembled heteroepitaxial islands in elastically anisotropic films." Physical Review B 67(16): 165414.

[87] C. h. Chiu (2004). "Stable and uniform arrays of self-assembled nanocrystalline islands." Physical Review B 69(16): 165413.

[88] V. V. Kuryliuk and O. A. Korotchenkov (2013). "Features of the stress-strain state of Si/SiO2/Ge heterostructures with germanium nanoislands of a limited density." Semiconductors 47(8): 1031-1036.

[89] R. J. Asaro and W. A. Tiller (1972). "Interface morphology development during stress corrosion cracking: Part I. Via surface diffusion." Metallurgical Transactions 3(7): 1789-1796.

[90] M. A. Grinfel'd (1987). "Instability of the equilibrium of a nonhydrostatically stressed body and a melt." Fluid Dynamics 22(2): 169-173.

[91] B. J. Spencer (1999). "Asymptotic derivation of the glued-wetting-layer model and contact-angle condition for Stranski-Krastanow islands." Physical Review B 59(3): 2011-2017.

[92] W. T. Tekalign and B. J. Spencer (2004). "Evolution equation for a thin epitaxial film on a deformable substrate." Journal of Applied Physics 96(10): 5505-5512.

[93] W. T. Tekalign and B. J. Spencer (2007). "Thin-film evolution equation for a strained solid film on a deformable substrate: Numerical steady states." Journal of Applied Physics 102(7): 073503.

[94] R. V. Kukta and L. B. Freund (1997). "Minimum energy configuration of epitaxial material clusters on a lattice-mismatched substrate." Journal of the Mechanics and Physics of Solids 45(11–12): 1835-1860.

[95] T. O. Ogurtani, A. Celik and E. E. Oren (2010). "Morphological evolution in a strained-heteroepitaxial solid droplet on a rigid substrate: Dynamical simulations." Journal of Applied Physics 108(6): 063527.

[96] T. O. Ogurtani (2006). "Mesoscopic nonequilibrium thermodynamics of solid surfaces and interfaces with triple junction singularities under the capillary and electromigration forces in anisotropic three-dimensional space." The Journal of Chemical Physics 124(14): 144706.

[97] T. O. Ogurtani (2006). "Unified theory of linear instability of anisotropic surfaces and interfaces under capillary, electrostatic, and elastostatic forces: The regrowth of epitaxial amorphous silicon." Physical Review B 74(15): 155422.

[98] T. O. Ogurtani and E. E. Oren (2005). "Irreversible thermodynamics of triple junctions during the intergranular void motion under the electromigration forces." International Journal of Solids and Structures 42(13): 3918-3952.

[99] T. Kundu (1993). "Introduction to finite and boundary element methods for engineers (G. Beer and J. O. Watson, Wiley, New York, 1992. ISBN 0471 928135 522 pp.)." International Journal for Numerical and Analytical Methods in Geomechanics 17(8): 600.

APPENDIX A

PROGRAM CODE

```
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <math.h>
#include <stdio.h>
#include <time.h>
#include <iomanip>
#include <string>
#include <ctime>
#include <cstdlib>

using namespace std;

ofstream out,kout;
ifstream in;

struct line
{string name;};

typedef double Number;
typedef Number arr1[1001];
typedef Number arr2[3][1001];
typedef Number arr3[1001][1001];
typedef Number arr4[3][3];
const long double pi = 3.14159265358979323846264338327795;
char line[256] = "";
FILE *fp;
 int
    nupdown, nd,numContData=0,lastOutNum,
    ms,nl,type,rem,contData,
        nrem,nrup,nrdown,nrud,ndelru,
        Modiv,Msin,nsw,mint,nu,t=0,mm,mpow,fmn;

 Number
        newdata,maxSeglenth,
        ho,ro=1,sl,sln,sw,slw,Amp,
        Cksi, ksi,
        ym,poisson,delGb,Mb,Mg,Cepsilon,Csigma,
        Aint,Bint,tphi,hfn,
        gammaf,gammas,delw,lamdag,gfm,
        epstime,Sigma,Eta,delta,
```

```
        rmax,rmin,
        lamda,lamdau,kv,omega,epsx,rtphi,
        dmean,vmax,
        lamG,lammu,lamlamda,cc,c1,c2,c3,c4,cc1,cc2,
        thetaR,thetaL,
        velL,velR,deltat,timex,delrend,delrlower,delrfirst;

 arr1
    sup,sdown,Wpot,Wcal,
        ulas,vect,
        s,su,sd,sud,
        teta,tetau,tetad,tetaw,
        kapkap,kapkapup,kapkapdown,kapkapud,
        xmu,ymu,mamu,
        tn,
        aqx,aqy,aqxy,aqz,Trq,hoop,Sighoop,
        dif,theta,
        omom,TauO,TauD,TauS,
        vel,Psiu,
        cffr,
        mu,fieldi,fieldii,fieldt,
        fieldin,fieldiin,fieldtn,
        trqa,fsigma,ub,tarik,
        strainenergy,tpot,tpotkap,
        tpotint,tpotvo,tpotkapint,tpotkapvo,
        velint,velvo,diffint,diffvo;

 arr2
        delrup,delrdown,
    rupa,rup,rdown,rud,rm,                                  //
        rintvo,rgbup,rgbdown,                               //
        delr,delru,delrd,delrud,                            //
        anti,                                               //
        lln,llnint,llnvo,                                   //
        noc,nocint,nocvo,nocintvo,                          //
        rcud,                                               //
        trac,ru,ruprc;                                          //

 arr3
        tt,delu,fttbig,fuubig,sgbig,duubig;                 //

 arr4
        ss,us,ttssx;                                        //

string  sy,textName,outName;
//

inline Number sqr(Number x);
inline int timer(int& m, int& e);
inline Number dotpro(Number& a0, Number& a1, Number& a2, Number& b0, Number
& b1, Number& b2);
inline Number magnitude(Number& a, Number& b, Number& c);
inline Number arcsin(Number& okst);
inline Number angle(Number& a0, Number& a1, Number& a2, Number& b0, Number&
 b1, Number& b2);
inline Number area(int& n, arr2& r);

void vectorpro(Number& a0, Number& a1, Number& a2, Number& b0, Number& b1,
Number& b2);
```

```
97. void antirotma(Number& w);
98. void trian(int colon,arr1 tek,arr3 cift);
99. void uppart();
100.        void lowpart();
101.        void stacksvi(arr2& ru, int& nu, arr2& rd, int&nd);
102.        void stackv(arr1& f, int& fn, arr1& s, int& sn, arr1& r, int& rn);

103.        void stackv(arr2& f, int& fn, arr2& s, int& sn, arr2& r, int& rn);

104.        void delr1(int& nn, arr2& r,arr2& delr,arr1& s);
105.        void deldelr1(int& nn, int& looptype, arr2& r);
106.        void psir(int deln, arr2 delr);   // degistirildi...
107.        void psipsir(int& deln, arr2& delr, arr1& s);
108.        void kappa(arr1& s, arr1& teta, arr2& delr, int& n);
109.        void stacksab();
110.        void nocRT(arr1& s, arr2& delr, int nu, int nd);
111.        void rcc(arr2& r, int& n);
112.        void Sss(Number& rk0, Number& rk1, Number& rk2);
113.        void uu(Number& rk0, Number& rk1, Number& rk2);
114.        void ttss(Number& rk0, Number& rk1, Number& rk2, Number& nk0, Numbe
    r& nk1, Number& nk2);
115.        void ftin(arr1& sup, arr2& delrup);
116.        void asym();
117.        void boundary();
118.        void pbfv();
119.        void fc();
120.        void dsglarge();
121.        void multa();
122.        void SigStress();
123.        void SigNodet();
124.        void durq();
125.        void multb();
126.        void ddif();
127.        void centerpoint();
128.        void remesh0();
129.        void remesh1();
130.        void remeshlower();
131.        void calnew();
132.        void calruv();
133.        void generate();
134.        void getparam();
135.        void getcontparam();
136.        void final();
137.        void ksitbir();
138.        void tbir();
139.        void clrscr(void);
140.        void recording();
141.        void recordtimestep();
142.        void needparam() ;
143.        void continues();
144.        void writeParam();
145.        void remeshEnd();
146.
147.        int main()
148.        {
149.          needparam();
150.
151.         writeParam();
152.          if(!contData) generate();
```

```
          else continues();

          final();
          return 0;
        }

        void continues(){

        int nulll;
        Number nullll;

        in.open("cont.dat");
        in >> rup[0][0] >> rup[0][1] >> nrup >> nrdown >> t >> lastOutNum >
    > timex >> dmean ;


            nrud = nrup+nrdown;

        for(int i=1; i<nrup;i++){
        in >> rup[0][i]>> rup[1][i] ;
        }

        for(int i=0; i<nrdown;i++){
        in >> rdown[0][i]>> rdown[1][i] ;
        }

        in.close();

            if (t < 256) mpow =pow(2,(double)lastOutNum);
            else if (t < 1000) mpow =300+200*(lastOutNum-9);
            else if (t < 10000) mpow =1000+500*(lastOutNum-13);
            else if (t < 20000) mpow =10000+1000*(lastOutNum-31);
            else if (t < 100000) mpow =20000+5000*(lastOutNum-41);
            else mpow =100000+10000*(lastOutNum-57);

        }

        void int2str(int i){
        char  index[10][2] = {"0","1","2","3","4","5","6","7","8","9"};
        //      mynum = (string)index[temp/1000];
        //      temp -= (temp/1000)*1000;
                textName = (string)index[i/100];
                i -= (i/100)*100;
                textName +=(string)index[i/10];
                i -= (i/10)*10;
                textName +=(string)index[i];
                textName += "csl.dat";
        }

        // AUXILIARY FUNCTIONS & PROCEDURES

        inline Number sqr(Number x){
            return x*x;
            }

        // this function determines the record time steps

        inline int timer(int& m,int& e)
        {
```

```
211.         int powa = 1;
212.         if (e != 0) for (int ki = 1; ki <= e; ki++)  powa *= m;
213.         return powa;
214.       }
215.
216.       //this function finds the dot product of two vectors
217.
218.       inline Number dotpro(Number& a0, Number& a1, Number& a2, Number& b0
    , Number& b1, Number& b2)
219.       {
220.         return a0*b0+a1*b1+a2*b2;
221.       }
222.
223.       //this function finds the magnitude of the vectors
224.
225.       inline Number magnitude(Number& a, Number& b, Number& c)
226.       {
227.         return sqrt(a * a + b * b + c * c);
228.       }
229.
230.       //this function finds the arcsin(teta)
231.
232.       inline Number arcsin(Number& okst)
233.       {
234.         Number arcs, sens =0.0000000000001;
235.
236.         if (okst > 1-sens) arcs = pi/2;
237.         if (okst < sens-1) arcs = -(pi/2) ;
238.         if (okst < sens)
239.           {
240.           if (okst > -sens) arcs = 0;
241.           }
242.         if (okst <= 1 - sens)
243.           {
244.           if (okst >= sens) arcs = atan(1/sqrt(1/(sqr(okst))-1));
245.           }
246.         if (okst <= -sens)
247.           {
248.           if (okst >= sens-1) arcs = atan(1/sqrt(1/(sqr(okst))-1));
249.           }
250.
251.         if (okst < 0)
252.           {
253.           if (okst > -1)  arcs = -arcs;
254.           }
255.        return arcs;
256.       }
257.
258.       // this function finds the angle between two vectors
259.
260.       inline Number angle(Number& a0, Number& a1, Number& a2, Number& b0,
    Number& b1, Number& b2)
261.       {
262.         Number angles,dd,asr,dotp;
263.         dotp = dotpro(a0,a1,a2,b0,b1,b2);
264.         dd = magnitude(a0,a1,a2)*magnitude(b0,b1,b2);
265.         asr = (a0*b1-a1*b0)/dd;
266.         angles = arcsin(asr);
267.         if (dotp<=0.0) angles = pi-angles;
```

```
268.          // if (angles > pi) angles  -= 2.0*pi;
269.          // if ( angles < 0.0) angles += 2.0*pi ;
270.          return angles;
271.        }
272.
273.        // this function finds the surface void/hillock area
274.
275.        inline Number area(int& n, arr2& r)
276.        {
277.          Number t;
278.
279.          for (int ki = 0; ki <= n-2; ki++)
280.            t += 0.5*(r[0][ki]*r[1][ki+1]-r[1][ki]*r[0][ki+1]);
281.          t += 0.5*(r[0][n-1]*r[1][0]-r[1][n-1]*r[0][0]);
282.          return t;
283.        }
284.
285.        // this function finds the vector product of two vectors
286.
287.        void vectorpro(Number& a0, Number& a1, Number& a2, Number& b0, Numb
    er& b1, Number& b2)
288.        {
289.          vect[0] = a1*b2-a2*b1;
290.          vect[1] = -a0*b2+a2*b0;
291.          vect[2] = a0*b1-a1*b0;
292.        }
293.
294.        // production of a anticlockwise rotation matrix
295.
296.        void antirotma(Number& w)
297.        {
298.          anti[0][0] = cos(w);
299.          anti[0][1] = -sin(w);
300.          anti[0][2] = 0.0;
301.          anti[1][0] = -anti[0][1];
302.          anti[1][1] = anti[0][0];
303.          anti[1][2] = 0.0;
304.          anti[2][0] = 0.0;
305.          anti[2][1] = 0.0;
306.          anti[2][2] = 1.0;
307.        }
308.
309.        // triangulation method in the solution of simulataneous set equati
    ons
310.
311.        void trian(int colon, arr1 tek, arr3 cift)
312.        {
313.
314.          Number tot,bol, trio[colon+2];
315.          //arr3   trio;
316.
317.          for (int ki = 0; ki <= colon; ki++)
318.            cift[ki][colon+1] = tek[ki];
319.
320.          for (int ki = 0; ki <= colon; ki++)
321.            {
322.                bol = cift[ki][ki];
323.                for (int kj = 0; kj <= colon + 1; kj++) cift[ki][kj] /= bol
    ;
```

```
                for (int kk = ki; kk <= colon; kk++)
                {
                    if (kk != ki)
                    {
                        for (int kj = 0; kj <= colon + 1; kj++)
                          trio[kj] = cift[ki][kj] * cift[kk][ki];
                        for (int kj = 0; kj <= colon + 1; kj++)
                          cift[kk][kj] -= trio[kj];
                    }
                }
            }

          ulas[colon] = cift[colon][colon+1];
          for (int ki = 1; ki <= colon; ki++)
            {
                tot = 0.0;
                for (int kj = 1; kj <= ki; kj++)
                  tot += ulas[colon - kj + 1] * cift[colon - ki][colon -
     kj + 1];
                ulas[colon - ki] = cift[colon - ki][colon+1] - tot;
            }
        }


        // MAIN FUNCTIONS & PROCEDURES

        // this procedure generates the upper part of the strip in clockwis
    e direction

        void uppart()
        {
          for(int ki = 0; ki<= 2*Msin ; ki++)
            {
            rup[0][ki] = (ki-Msin)*sl/Msin;
            rup[1][ki] = sw*cos(rup[0][ki]*pi/2/sl);
            rup[2][ki] = 0;
            }
          nrup = 2*Msin+1;

        }

        // this procedure generates the lower part of the strip

        void lowpart()
        {
            if (fmn>1){
                    for (int i=0; i<=2*fmn-1;i++){
                            rdown[0][i] = -
    ((i+1)*(sl/Modiv)/fmn)+Modiv*(sl/Modiv);
                            rdown[1][i] = 0;
                            rdown[2][i] = 0;
                    }
                    for (int j=3; j<=2*Modiv-2; j++){
                            rdown[0][2*fmn+j-3] = -(j-Modiv)*(sl/Modiv);
                            rdown[1][2*fmn+j-3] = 0;
                            rdown[2][2*fmn+j-3] = 0;
                    }
                    for (int k=0; k<=fmn-1; k++){
```

```
379.                            rdown[0][k+2*Modiv-4+2*fmn] = -
    ((k+1)*(sl/Modiv)/fmn)+rdown[0][2*Modiv-5+2*fmn];
380.                            rdown[1][k+2*Modiv-4+2*fmn] = 0;
381.                            rdown[2][k+2*Modiv-4+2*fmn] = 0;
382.                    }
383.                    for (int kj=0; kj<=fmn-2; kj++){
384.                            rdown[0][kj+2*Modiv-4+3*fmn] = -
    ((kj+1)*(sl/Modiv)/fmn)+rdown[0][2*Modiv-5+3*fmn];
385.                            rdown[1][kj+2*Modiv-4+3*fmn] = 0;
386.                            rdown[2][kj+2*Modiv-4+3*fmn] = 0;
387.                    }
388.                    nrdown=2*Modiv+4*fmn-5;
389.            }
390.            else {
391.          for(int ki = 1; ki<= 2*Modiv ; ki++)
392.            {
393.            rdown[0][ki-1] = -(ki-Modiv)*sl/Modiv;
394.            rdown[1][ki-1] = 0;
395.            rdown[2][ki-1] = 0;
396.            }
397.          nrdown = 2*Modiv-1;
398.
399.        }
400.        }
401.
402.        // this procedure combines the upper and lower parts
403.
404.        void stacksvi(arr2& rup, int& nrup, arr2& rdown, int&nrdown)
405.        {
406.          for (int kj = 0; kj <= nrup-1 ; kj++)
407.            {
408.            rud[0][kj] = rup[0][kj];
409.            rud[1][kj] = rup[1][kj];
410.            rud[2][kj] = rup[2][kj];
411.            }
412.          for (int kj = nrup; kj <= nrup+nrdown-1 ; kj++)
413.            {
414.            rud[0][kj] = rdown[0][kj-nrup];
415.            rud[1][kj] = rdown[1][kj-nrup];
416.            rud[2][kj] = rdown[2][kj-nrup];
417.            }
418.          nrud = nrup+nrdown;
419.        }
420.
421.        // These correspond to the total upper and lower segmen lengths and
      vectors set
422.        void delr1(int& nn, arr2& r,arr2& delr,arr1& s)
423.        {
424.          for (int ki = 0; ki <= nn-2; ki++)
425.            for (int kj = 0; kj <= 2; kj++)
426.              delr[kj][ki] = r[kj][ki+1]-r[kj][ki];
427.
428.          for (int ki = 0; ki <= nn-2; ki++)
429.            s[ki] = magnitude(delr[0][ki],delr[1][ki],delr[2][ki]);
430.        }
431.
432.        // this procedure calculates difference vectors between successive
    position
433.        // vectors and their magnitudes
```

```
434.
435.       void deldelr1(int& nn, int looptype, arr2& r)
436.       {
437.         if (looptype==1) ndelru = nn-1;
438.         if (looptype==0) ndelru = nn-2;
439.
440.         for (int ki = 0; ki <= ndelru; ki++)
441.           {
442.           if(looptype!=1)
443.             {
444.             for (int kj = 0; kj <= 2; kj++) delr[kj][ki] = r[kj][ki+1]-
    r[kj][ki];
445.             s[ki] = magnitude(delr[0][ki],delr[1][ki],delr[2][ki]);
446.             }
447.           else
448.             {
449.             if(ki==ndelru)
450.               {
451.           for (int kj = 0; kj <= 2; kj++) delr[kj][ki] = r[kj][0]-
    r[kj][ki];
452.           s[ki] = magnitude(delr[0][ki],delr[1][ki],delr[2][ki]);
453.               }
454.             else
455.               {
456.           for (int kj = 0; kj <= 2; kj++) delr[kj][ki] = r[kj][ki+1]-
    r[kj][ki];
457.           s[ki] = magnitude(delr[0][ki],delr[1][ki],delr[2][ki]);
458.               }
459.             }
460.           }
461.       }
462.
463.
464.       // this procedure calculates the angle between the two successive 3
    -d vectors
465.       // in a given set of vectors. the range -p and +p
466.
467.       void psir(int deln, arr2 delr, arr1& s)
468.       {
469.        for (int ki = 1; ki <= deln-1; ki++)
470.           {
471.           teta[ki] = -angle(delr[0][ki-1],delr[1][ki-1],delr[2][ki-
    1],delr[0][ki],delr[1][ki],delr[2][ki]);
472.           }
473.         teta[0] = teta[1]-(teta[2]-teta[1])/s[1]*s[0];
474.         teta[deln] = teta[deln-1]+(teta[deln-1]-teta[deln-2])/s[deln-
    2]*s[deln-1];
475.       }
476.
477.       // this procedure calculates the angle between the two successive 3
    -d vectors
478.       // in a given set of vectors. the range -p and +p
479.
480.       void psipsir(int deln, arr2& delr, arr1& s)
481.       {
482.         for (int ki = 0; ki <= deln-1; ki++)
483.           {
484.
485.           tetaw[ki] = asin(delr[1][ki]/s[ki]);
```

```
486.
487.
488.                 if( tetaw[ki] < 0.0) tetaw[ki] += 2*pi;
489.                 if( tetaw[ki] > pi) tetaw[ki] -= 2*pi;
490.
491.           }
492.         tetaw[deln] = -tetaw[0];
493.       }
494.
495.       // this procedure calculates the local curvature and the local line
     normal vector
496.       // at any given node knowing the successive segment vector set
497.
498.       // in this procedure:
499.       //                     kapkap : local curvature
500.       //                     lln    : local line normal
501.
502.       void kappa(arr1& s, arr1& teta, arr2& delr, int n)
503.       {
504.        arr1 alfa,beta;
505.         arr2 no;
506.         Number kapb;
507.
508.         for (int ki=1; ki<=n-1; ki++)
509.           {
510.            alfa[ki] = atan((sin(-teta[ki])*s[ki])/(s[ki-1]+cos(-
    teta[ki])*s[ki]));
511.            kapkap[ki] = 2*sin(alfa[ki])/s[ki];
512.           }
513.         //    alfa[0] = atan((sin(-teta[0])*s[0])/(s[n-1]+cos(-
    teta[0])*s[0]));
514.               kapkap[0] = kapkap[1]-(kapkap[2]-kapkap[1])/s[1]*s[0];
515.           kapkap[n] = kapkap[n-1]+(kapkap[n-1]-kapkap[n-2])/s[n-2]*s[n-
    1];
516.
517.         for (int ki=0; ki<=n-1; ki++)
518.           {
519.             beta[ki] = (pi-2*alfa[ki])*0.5;
520.             kapb = -beta[ki];
521.             antirotma(kapb);
522.
523.            if(ki == 0){
524.              beta[ki] = (pi-2*alfa[n-1])*0.5;
525.             kapb = -beta[ki];
526.             antirotma(kapb);
527.                 }
528.
529.           for(int kj=0; kj<=2; kj++)
530.             no[kj][ki] = dotpro(anti[kj][0],anti[kj][1],anti[kj][2],delr[
    0][ki-1],delr[1][ki-1],delr[2][ki-1]);
531.
532.           for(int kj=0; kj<=2; kj++)
533.             lln[kj][ki] = no[kj][ki]/magnitude(no[0][ki],no[1][ki],no[2][
    ki]);
534.           }
535.
536.           for(int kj=0; kj<=2; kj++)
537.           {
538.             lln[kj][0] = lln[kj][1];
```

```
539.             lln[kj][n] = lln[kj][n-1];
540.           }
541.       }
542.
543.       // This procedure combines the upper and lower parts
544.
545.       void stacksab()
546.       {
547.         for (int kj = 0; kj <= nrup-1 ; kj++)
548.           kapkapud[kj] = kapkapup[kj];
549.         for (int kj = nrup; kj <= nrup+nrdown-1 ; kj++)
550.           kapkapud[kj] = kapkapdown[kj-nrup];
551.         nrud = nrup+nrdown;
552.       }
553.
554.
555.       // this procedure calculates the normal unit vectors at the centroi
    ds
556.       // for the upper and lower cut interfaces plus the void.
557.       // Directions towards the interconnect material}
558.       //                          {noc : the centroid normal vector}
559.
560.       void nocRT(arr1& s, arr2& delr, int nu, int nd)
561.       {
562.         arr1 kz;
563.
564.         kz[0]=0;
565.         kz[1]=0;
566.         kz[2]=-1;
567.
568.         for (int kj = 0; kj <= nu-2; kj++)
569.           {
570.             vectorpro(kz[0],kz[1],kz[2],delr[0][kj],delr[1][kj],delr[2][k
    j]);
571.             noc[0][kj] = vect[0]/s[kj];
572.             noc[1][kj] = vect[1]/s[kj];
573.             noc[2][kj] = vect[2]/s[kj];
574.           }
575.         kz[0]=0;
576.         kz[1]=0;
577.         kz[2]=1;
578.         for (int kj = nu-1; kj <= nu+nd-1; kj++)
579.           {
580.             vectorpro(kz[0],kz[1],kz[2],delr[0][kj],delr[1][kj],delr[2][k
    j]);
581.             noc[0][kj] = -vect[0]/s[kj];
582.             noc[1][kj] = -vect[1]/s[kj];
583.             noc[2][kj] = -vect[2]/s[kj];
584.           }
585.       }
586.
587.       // this procedure calculates the centroid position vectors for the

588.       // upper and lower regions
589.
590.
591.       void rcc(arr2& r, int& n)
592.       {
593.         for(int ki=0; ki<=n-1; ki++)
```

```
            for(int kj=0; kj<=2; kj++)
              {
              if (ki==n-1)
                rcud[kj][ki] = (r[kj][0] + r[kj][ki])*0.5;
              else
                rcud[kj][ki] = (r[kj][ki+1] + r[kj][ki])*0.5;
              }
        }


        // STRESS CALCULATIONS

        // INDIRECT BOUNDARY ELEMENT METOD

        // UNIAXIAL TENSION or COMPRESSION along the X-
    axis (Plain Strain Assumption)

        // Most of the calculations done by pseudo vectors assuming that th
    e shear strain
        // is defined by e(i,j)=u(i,j)+u(j,i) where  i NQE j otherwise e(i,
    i)= u(i,i).
        // In the case of scientific notation using tensor or dyadics strai
    n components are
        // defined by e(i,j)= 1/2(u(i,j) + u(j,i) ).

        // Because of this disctinction between engineering and scientific
    definitions there
        // are many mistakes in the literature especially in Beer and Watso
    n formulas for
        // strain and stress (pages  488 and 489).

        // This program calculates the stress connection S matrix (3x2) at
    a point Q for unit
        // load situated at the point P. For a given connection vector rk (
    RK=QP) and poisson is
        // the Poisson's ratio

        void Sss(Number& rk0, Number& rk1, Number& rk2)
        {
          Number rx,ry,srk;

          srk = magnitude(rk0,rk1,rk2);
          rx = rk0/srk;
          ry = rk1/srk;
          ss[0][0] = c2*(c3*rx+2*rx*rx*rx)/srk;
          ss[0][1] = c2*(-c3*ry+2*rx*rx*ry)/srk;
          ss[1][0] = c2*(-c3*rx+2*rx*ry*ry)/srk;
          ss[1][1] = c2*(c3*ry+2*ry*ry*ry)/srk;
          ss[2][0] = c2*(c3*ry+2*rx*rx*ry)/srk;
          ss[2][1] = c2*(c3*rx+2*rx*ry*ry)/srk;
        }

        // This program calculates the displacement matrix due to a unit fo
    rce which is rk apart
        // from the point of interest. Here n  is the poisson's ratio. E is
     the Young Modulus should
        // be taken as equal to unity and the load should renormalized with
     respect to the Young Modulus.
```

```
641.       // The unit lenght should be so chosen that the void may lie  compl
   etely inside of a square
642.       // determined by the unit length.
643.
644.       void uu(Number& rk0, Number& rk1, Number& rk2)
645.       {
646.         us[0][0] = cc*(-(3-
   4*poisson)*log(magnitude(rk0,rk1,rk2))+sqr(rk0)/sqr(magnitude(rk0,rk1,rk2))
   );
647.         us[1][1] = cc*(-(3-
   4*poisson)*log(magnitude(rk0,rk1,rk2))+sqr(rk1)/sqr(magnitude(rk0,rk1,rk2))
   );
648.         us[0][1] = cc*(rk0*rk1/sqr(magnitude(rk0,rk1,rk2)));
649.         us[1][0] = us[0][1];
650.       }
651.
652.       // This program calculates the TRACTION function associated with an
     unit force situated at
653.       // P and acting at point Q. (rk=QP VECTOR). where  nk  is the unit
   ouward  normal at  Q,
654.       // which is given by minus NocRT. The surface of the interconnect i
   ncluding internal void
655.
656.       void ttss(Number& rk0, Number& rk1, Number& rk2, Number& nk0, Numbe
   r& nk1, Number& nk2)
657.       {
658.         Number rx;
659.         arr4 ts,ta;
660.
661.         rx = magnitude(rk0,rk1,rk2);
662.         ts[0][0] = ((1-
   2*poisson)+2*sqr(rk0/rx))*(rk0*nk0+rk1*nk1+rk2*nk2)/(sqr(rx));
663.         ts[1][1] = ((1-
   2*poisson)+2*sqr(rk1/rx))*(rk0*nk0+rk1*nk1+rk2*nk2)/(sqr(rx));
664.         ts[0][1] = 2*rk0*rk1*(rk0*nk0+rk1*nk1+rk2*nk2)/sqr(sqr(rx));
665.         ts[1][0] = ts[0][1];
666.
667.         ta[0][1] = (1-2*poisson)*(rk1*nk0-rk0*nk1)/sqr(rx);
668.         ta[1][0] = -ta[0][1];
669.         ta[0][0] = 0;
670.         ta[1][1] = 0;
671.
672.
673.         for(int ki=0; ki<=1; ki++)
674.           for(int kj=0; kj<=1; kj++)
675.             ttssx[kj][ki] = c2*(ts[ki][kj]+ta[ki][kj]);
676.       }
677.
678.       // This is an elastostatic connection matrix utilizing the element
   centroids,
679.       // M is the (odd) number of subsegment used in the integration proc
   edure
680.
681.       // In the following program Nor is the outward normal vector (from
   body to the
682.       // surroundings) at the segment centroids. This program has two par
   ts. boundary
683.       // displacement (Dirichlet ) and traction (Neuman) parts namely.
684.
```

```
685.       // For the present application we are using mixed everywhere.
686.
687.       void ftin(arr1& sup, arr2& delr)
688.       {
689.         arr2 rcc,nox;
690.         arr4 ttssa,ttssb,ttssk,ttsst,uua,uub,uusk,uuss;
691.
692.
693.         for(int ki=0; ki<=nrud-1; ki++)
694.           for(int kj=0; kj<=2; kj++)
695.             {
696.             nox[kj][ki] = -noc[kj][ki];
697.             }
698.
699.         for(int ki=0; ki<=nrup-2; ki++)
700.           for(int kj=0; kj<=nrud-1; kj++)
701.             {
702.             if (ki==kj)
703.               {
704.               fttbig[ki*2][kj*2] = 0.5;
705.               fttbig[ki*2][kj*2+1] = 0;
706.               fttbig[ki*2+1][kj*2] = 0;
707.               fttbig[ki*2+1][kj*2+1] = 0.5;
708.               }
709.             else
710.               {
711.                   for(int kk=0; kk<=mint; kk++)
712.                     for(int kl=0; kl<=2; kl++)
713.                   rcc[kl][kk] = rud[kl][kj]-
    rcud[kl][ki]+kk*delr[kl][kj]/mint;
714.
715.               ttss(rcc[0][0],rcc[1][0],rcc[2][0],nox[0][ki],nox[1][ki],no
    x[2][ki]);
716.
717.
718.                   for(int kik=0; kik<=1; kik++)
719.                     for(int kjk=0; kjk<=1; kjk++)
720.                       ttssa[kik][kjk] = ttssx[kik][kjk];
721.
722.
723.                   ttss(rcc[0][mint],rcc[1][mint],rcc[2][mint],nox[0][ki],
    nox[1][ki],nox[2][ki]);
724.
725.                   for(int kik=0; kik<=1; kik++)
726.                     for(int kjk=0; kjk<=1; kjk++)
727.                       ttssb[kik][kjk] = ttssx[kik][kjk];
728.
729.               ttsst[0][0] = 0.5*(ttssa[0][0]+ttssb[0][0]);
730.               ttsst[0][1] = 0.5*(ttssa[0][1]+ttssb[0][1]);
731.               ttsst[1][0] = 0.5*(ttssa[1][0]+ttssb[1][0]);
732.               ttsst[1][1] = 0.5*(ttssa[1][1]+ttssb[1][1]);
733.
734.                   for(int kk=1; kk<=mint-1; kk++)
735.                 {
736.                 ttss(rcc[0][kk],rcc[1][kk],rcc[2][kk],nox[0][ki],nox[1][k
    i],nox[2][ki]);
737.                     for(int kik=0; kik<=1; kik++)
738.                       for(int kjk=0; kjk<=1; kjk++)
739.                         ttssk[kik][kjk] = ttssx[kik][kjk];
```

```
                  ttsst[0][0] += ttssk[0][0];
                  ttsst[0][1] += ttssk[0][1];
                  ttsst[1][0] += ttssk[1][0];
                  ttsst[1][1] += ttssk[1][1];
                  }

                    fttbig[ki*2][kj*2] = sup[kj]*ttsst[0][0]/mint;
                    fttbig[ki*2][kj*2+1] = sup[kj]*ttsst[0][1]/mint;
                    fttbig[ki*2+1][kj*2] = sup[kj]*ttsst[1][0]/mint;
                    fttbig[ki*2+1][kj*2+1] =  sup[kj]*ttsst[1][1]/mint;
                }
              }
          for(int ki=nrup-1; ki<=nrud-1; ki++)
            for(int kj=0; kj<=nrud-1; kj++)
              {
                for(int kk=0; kk<=mint; kk++)
                  for(int kl=0; kl<=2; kl++)
                rcc[kl][kk] = rud[kl][kj]-
    rcud[kl][ki]+kk*delr[kl][kj]/mint;

                uu(rcc[0][0],rcc[1][0],rcc[2][0]);

                for(int kik=0; kik<=1; kik++)
                  for(int kjk=0; kjk<=1; kjk++)
                    uua[kik][kjk] = us[kik][kjk];

                uu(rcc[0][mint],rcc[1][mint],rcc[2][mint]);

                for(int kik=0; kik<=1; kik++)
                  for(int kjk=0; kjk<=1; kjk++)
                    uub[kik][kjk] = us[kik][kjk];

            uuss[0][0] = 0.5*(uua[0][0]+uub[0][0]);
            uuss[0][1] = 0.5*(uua[0][1]+uub[0][1]);
            uuss[1][0] = 0.5*(uua[1][0]+uub[1][0]);
            uuss[1][1] = 0.5*(uua[1][1]+uub[1][1]);

                for(int kk=1; kk<=mint-1; kk++)
              {
              uu(rcc[0][kk],rcc[1][kk],rcc[2][kk]);
                  for(int kik=0; kik<=1; kik++)
                    for(int kjk=0; kjk<=1; kjk++)
                      uusk[kik][kjk] = us[kik][kjk];
              uuss[0][0] += uusk[0][0];
              uuss[0][1] += uusk[0][1];
              uuss[1][0] += uusk[1][0];
              uuss[1][1] += uusk[1][1];
              }

            fttbig[ki*2][kj*2] = sup[kj]*uuss[0][0]/mint;
            fttbig[ki*2][kj*2+1] = sup[kj]*uuss[0][1]/mint;
            fttbig[ki*2+1][kj*2] = sup[kj]*uuss[1][0]/mint;
            fttbig[ki*2+1][kj*2+1] =  sup[kj]*uuss[1][1]/mint;
              }
        }

        // This program considers, the  droplet  surface is  free from the
    traction forces.
```

```
796.       // Lower interface surface is restrained by the substrade due to th
   e misfit strain.
797.       // Where the tractions and displacements are calculated at centroid
    positions.
798.       // The ends points of Droplet surface are placed at the interfaces,
    therefore the stress
799.       // calculated later at the nodes should include these points as wel
   l as!!!
800.
801.       // This is a very critical point because the same points are also b
   elong to the traction free sector.
802.       // Best thing is to avoid that by taking interface ends very close
   to the droplet ends!!!
803.
804.       void asym()
805.       {
806.         for(int ki=0; ki<=nrup-2; ki++)
807.           {
808.           trac[0][ki] = 0;
809.           trac[1][ki] = 0;
810.           trac[2][ki] = 1;
811.           }
812.         for(int ki=nrup-1; ki<=nrup+nrdown-1; ki++)
813.           {
814.           trac[0][ki] = epsx*rcud[0][ki];
815.           trac[1][ki] = 0;
816.           trac[2][ki] = epsx;
817.           }
818.         for(int ki=0; ki<=nrud-1; ki++)
819.             tn[ki] = -
   (trac[0][ki]*noc[0][ki]+trac[1][ki]*noc[1][ki]+trac[2][ki]*noc[2][ki]);
820.       }
821.
822.       // This is the boundary condition pseudo-vector
823.
824.       void boundary()
825.       {
826.         int ii;
827.         for(int ki=0; ki<=2*nrud-2; ki+=2)
828.           {
829.           ii = ki/2;
830.           trqa[ki] = trac[0][ii];
831.           }
832.         for(int ki=1; ki<=2*nrud-1; ki+=2)
833.           {
834.           ii = (ki-1)/2;
835.           trqa[ki] = trac[1][ii];
836.           }
837.
838.       }
839.
840.       // The following matrix equation calculates the pseudo-
   boundary force vector (pbfv)
841.       // which results zero net traction, in  the presence of thermal hyd
   rostatic
842.       // stress system, at the void-
   interconnect interface assuming that one has
843.       // plain strain situation.
844.
```

```
845.       // This corresponds to force intensities situated at the centroids
   of the segments
846.
847.       void pbfv()
848.       {
849.         trian(nrud*2-1,trqa,fttbig);
850.        for(int ki=0; ki<=2*nrud-1; ki++)
851.           fsigma[ki] = ulas[ki];
852.       }
853.
854.
855.       // This program separates the x- and y-
    components of the load vectors
856.
857.       void fc()
858.       {
859.         for(int ki=0; ki<=nrud-1; ki++)
860.           {
861.            xmu[ki]=fsigma[2*ki];
862.            ymu[ki]=fsigma[2*ki+1];
863.            mamu[ki]=sqrt(xmu[ki] * xmu[ki] + ymu[ki] * ymu[ki]);
864.           }
865.       }
866.
867.       // The following program calculates the displacement matrix
868.       // around the external and internal boundaries
869.
870.       void dsglarge()
871.       {
872.         arr2 rcc,nox;
873.         arr4 uussa,uussb,uussk,uusst,gssa,gssb,gssk,gsst;
874.
875.         for(int ki=0; ki<=nrud-1; ki++)
876.           for(int kj=0; kj<=nrud-1; kj++)
877.             {
878.             if (ki==kj)
879.           {
880.               gsst[0][0] = 0;
881.               gsst[0][1] = 0;
882.           gsst[1][0] = 0;
883.           gsst[1][1] = 0;
884.           gsst[2][0] = 0;
885.           gsst[2][1] = 0;
886.           }
887.             else
888.               {
889.               for(int kk=0; kk<=mint; kk++)
890.                 for(int kl=0; kl<=2; kl++)
891.               rcc[kl][kk] = rud[kl][kj]-
   rcud[kl][ki]+kk*delr[kl][kj]/mint;
892.
893.           Sss(rcc[0][0],rcc[1][0],rcc[2][0]);
894.               for(int kik=0; kik<=2; kik++)
895.                 for(int kjk=0; kjk<=1; kjk++)
896.                   gssa[kik][kjk] = ss[kik][kjk];
897.
898.           Sss(rcc[0][mint],rcc[1][mint],rcc[2][mint]);
899.               for(int kik=0; kik<=2; kik++)
900.                 for(int kjk=0; kjk<=1; kjk++)
```

```
                    gssb[kik][kjk] = ss[kik][kjk];

            gsst[0][0] = 0.5*(gssa[0][0]+gssb[0][0]);
                gsst[0][1] = 0.5*(gssa[0][1]+gssb[0][1]);
            gsst[1][0] = 0.5*(gssa[1][0]+gssb[1][0]);
            gsst[1][1] = 0.5*(gssa[1][1]+gssb[1][1]);
            gsst[2][0] = 0.5*(gssa[2][0]+gssb[2][0]);
            gsst[2][1] = 0.5*(gssa[2][1]+gssb[2][1]);

                for(int kk=1; kk<=mint-1; kk++)
              {
              Sss(rcc[0][kk],rcc[1][kk],rcc[2][kk]);
                  for(int kik=0; kik<=2; kik++)
                    for(int kjk=0; kjk<=1; kjk++)
                      gssk[kik][kjk] = ss[kik][kjk];
              gsst[0][0] += gssk[0][0];
              gsst[0][1] += gssk[0][1];
              gsst[1][0] += gssk[1][0];
              gsst[1][1] += gssk[1][1];
                  gsst[2][0] += gssk[2][0];
              gsst[2][1] += gssk[2][1];
                  }
                sgbig[ki*3][kj*2] = s[kj]*gsst[0][0]/mint;
                sgbig[ki*3][kj*2+1] = s[kj]*gsst[0][1]/mint;
            sgbig[ki*3+1][kj*2] = s[kj]*gsst[1][0]/mint;
            sgbig[ki*3+1][kj*2+1] = s[kj]*gsst[1][1]/mint;
                sgbig[ki*3+2][kj*2] = s[kj]*gsst[2][0]/mint;
            sgbig[ki*3+2][kj*2+1] = s[kj]*gsst[2][1]/mint;
                }
              }


          // for(int ki=0; ki<=3*nrud-1; ki++)
            // {
            // ub[ki]=0;
            // for(int kj=0; kj<=2*nrud-1; kj++)
            // ub[ki] += sgbig[ki][kj]*fsigma[kj];
            // }
        }

        // Matrix multiplication ssbig x fsigma

        void multa()
        {
          for(int ki=0; ki<=3*nrud-1; ki++)
            {
            ub[ki]=0;
            for(int kj=0; kj<=2*nrud-1; kj++)
            ub[ki] += sgbig[ki][kj]*fsigma[kj];
            }
        }
        // This program separates the components of the stress every where
     for the plane strain state.

        void SigStress()
        {
          int ii;

          for(int ki=0; ki<=3*nrud-2; ki+=3)
```

```
959.            {
960.            ii = ki/3;
961.            aqx[ii] = ub[ki];
962.            aqy[ii] = ub[ki+1];
963.            aqxy[ii] = ub[ki+2];
964.            aqz[ii] = poisson*(aqx[ii]+aqy[ii]);
965.            Trq[ii] = (1+poisson) * (aqx[ii]+aqy[ii]);
966.            hoop[ii] = delr[0][ii]*(aqx[ii]*delr[0][ii]+aqxy[ii]*delr[1][ii
    ]);
967.            hoop[ii] = hoop[ii]+delr[1][ii]*(aqxy[ii]*delr[0][ii]+aqy[ii]*d
    elr[1][ii]);
968.            hoop[ii] = hoop[ii]/(delr[0][ii]*delr[0][ii]+delr[1][ii]*delr[1
    ][ii]);
969.
970.            }
971.        }
972.
973.        // This program converts  the stress calculated at the centroid pos
    itions to the nodes
974.
975.        void SigNodet()
976.        {
977.          for(int ki=1; ki<=nrud-1; ki++)
978.            Sighoop[ki] = (hoop[ki-1]*s[ki]+hoop[ki]*s[ki-1])/(s[ki]+s[ki-
    1]);
979.
980.            Sighoop[0] = (hoop[nrud-1]*s[0]+hoop[0]*s[nrud-
    1])/(s[0]+s[nrud-1]);
981.
982.
983.        }
984.
985.        // The following program calculates the displacement matrix
986.        // around the external and internal boundaries
987.
988.        void durq()
989.        {
990.          arr2 rcc,nox;
991.          arr4 uussa,uussb,uussk,uusst;
992.
993.          for(int ki=0; ki<=nrud-1; ki++)
994.            for(int kj=0; kj<=nrud-1; kj++)
995.              {
996.                for(int kk=0; kk<=mint; kk++)
997.                  for(int kl=0; kl<=2; kl++)
998.                rcc[kl][kk] = rud[kl][kj]-
    rcud[kl][ki]+kk*delrud[kl][kj]/mint;
999.
1000.               uu(rcc[0][0],rcc[1][0],rcc[2][0]);
1001.
1002.               for(int kik=0; kik<=1; kik++)
1003.                 for(int kjk=0; kjk<=1; kjk++)
1004.                   uussa[kik][kjk] = us[kik][kjk];
1005.
1006.               uu(rcc[0][mint],rcc[1][mint],rcc[2][mint]);
1007.
1008.               for(int kik=0; kik<=1; kik++)
1009.                 for(int kjk=0; kjk<=1; kjk++)
1010.                   uussb[kik][kjk] = us[kik][kjk];
```

```
1011.
1012.               uusst[0][0] = 0.5*(uussa[0][0]+uussb[0][0]);
1013.               uusst[0][1] = 0.5*(uussa[0][1]+uussb[0][1]);
1014.           uusst[1][0] = 0.5*(uussa[1][0]+uussb[1][0]);
1015.           uusst[1][1] = 0.5*(uussa[1][1]+uussb[1][1]);
1016.
1017.               for(int kk=1; kk<=mint-1; kk++)
1018.             {
1019.             uu(rcc[0][kk],rcc[1][kk],rcc[2][kk]);
1020.                 for(int kik=0; kik<=1; kik++)
1021.                   for(int kjk=0; kjk<=1; kjk++)
1022.                     uussk[kik][kjk] = us[kik][kjk];
1023.             uusst[0][0] += uussk[0][0];
1024.             uusst[0][1] += uussk[0][1];
1025.             uusst[1][0] += uussk[1][0];
1026.             uusst[1][1] += uussk[1][1];
1027.             }
1028.               duubig[ki*2][kj*2] = s[kj]*uusst[0][0]/mint;
1029.               duubig[ki*2][kj*2+1] = s[kj]*uusst[0][1]/mint;
1030.               duubig[ki*2+1][kj*2] = s[kj]*uusst[1][0]/mint;
1031.               duubig[ki*2+1][kj*2+1] = s[kj]*uusst[1][1]/mint;
1032.             }
1033.
1034.         for(int ki=0; ki<=2*nrud-1; ki++)
1035.           {
1036.               ub[ki]=0;
1037.               for(int kj=0; kj<=2*nrud-1; kj++){
1038.                   cout<<"";
1039.                   ub[ki] += duubig[ki][kj]*fsigma[kj];
1040.               }
1041.           }
1042.       }
1043.
1044.       // Matrix multiplication duubig x fsigma
1045.
1046.       void multb()
1047.       {
1048.       //cout << "here";
1049.         for(int ki=0; ki<=2*nrud-1; ki++)
1050.           {
1051.           ub[ki]=0;
1052.           for(int kj=0; kj<=2*nrud-1; kj++)
1053.           ub[ki] += duubig[ki][kj]*fsigma[kj];
1054.           }
1055.       }
1056.
1057.       // Where B should be less then 1, 1/7 and 1/17 for 110, 100 and 111
     planes, otherwise one would have
1058.       // anomolous surface stiffness regime.
1059.
1060.       // This program calculates the anistropic diffusion coefficients at
     the centroid positions which is
1061.       // important for the flux evaluations there! here mm is the symmetr
    y fold number, fi is the tilt angle
1062.       // and A is the amplitude of the anistropic part of the diffusion c
    oefficient.
1063.
1064.       // mm=1 ->  two-fold symmetry  --> 110 plane
1065.       // mm=2 ->  four-fold symmetry --> 100 plane
```

```
1066.       // mm=3 ->  six-fold symmetry  --> 111 plane
1067.
1068.       void ddif()
1069.       {
1070.         int ii,ki;
1071.         arr1 ux,uy,delsc,epsc;
1072.         arr2 ubn,delrc;
1073.
1074.         for(int ki=0; ki<=nrup-2; ki++)
1075.             dif[ki] = 1 + Aint*cos(hfn*(tetaw[ki]-
    rtphi))*cos(hfn*(tetaw[ki]-rtphi));
1076.
1077.         for(int ki=0; ki<=nrup-1; ki++)
1078.           {
1079.           if (ki == 0)
1080.             theta[ki] = tetaw[0];
1081.           else if (ki == nrup-1)
1082.             theta[ki] = tetaw[nrup-2];
1083.           else
1084.             theta[ki] = (tetaw[ki]*s[ki-1]+tetaw[ki-1]*s[ki])/(s[ki-
    1]+s[ki]);
1085.           }
1086.         thetaR = theta[nrup-1];
1087.         thetaL = theta[0];
1088.         for(int kj=0; kj<=nrup-1; kj++)
1089.           {
1090.           if (type == 0)
1091.             {
1092.               TauO[kj] = 1 + Bint*sin(hfn*(theta[kj]-
    rtphi))*sin(hfn*(theta[kj]-rtphi));
1093.               TauD[kj] = Bint*hfn*sin(2*hfn*(theta[kj]-rtphi));
1094.               TauS[kj] = (1+Bint/2)*(1-(Bint*(1-
    4*hfn*hfn)/(Bint+2)*cos(2*hfn*(theta[kj]-rtphi))));
1095.             }
1096.           else if (type == 1)
1097.             {
1098.               TauO[kj] = 1-Bint+Bint*(fabs(sin(0.5*hfn*(rtphi-
    theta[kj])))+fabs(cos(0.5*hfn*(rtphi-theta[kj]))));
1099.               TauD[kj] = 0.5*hfn*Bint*sin(0.5*hfn*(rtphi-
    theta[kj]))*cos(0.5*hfn*(rtphi-theta[kj]));
1100.               TauD[kj] = TauD[kj]*((1/fabs(sin(0.5*hfn*(rtphi-
    theta[kj])))-(1/fabs(cos(0.5*hfn*(rtphi-theta[kj])))));
1101.               TauS[kj] = 1-Bint+Bint*(1-
    0.25*hfn*hfn)*(fabs(sin(0.5*hfn*(rtphi-
    theta[kj])))+fabs(cos(0.5*hfn*(rtphi-theta[kj]))));;
1102.             }
1103.           }
1104.       }
1105.
1106.       //CENTER POINT
1107.
1108.       void centerpoint()
1109.       {
1110.           int ki;
1111.           for (int ki=1; ki<=nrup-2; ki++)
1112.           {
1113.               ruprc[0][ki]=(((rup[1][ki]-rup[1][ki-1])/(rup[0][ki]-
    rup[0][ki-1]))*((rup[1][ki+1]-rup[1][ki])/(rup[0][ki+1]-
    rup[0][ki]))*(rup[1][ki+1]-rup[1][ki-1])+((rup[1][ki]-rup[1][ki-
```

```
    1])/(rup[0][ki]-rup[0][ki-1]))*(rup[0][ki]+rup[0][ki+1])-((rup[1][ki+1]-
    rup[1][ki])/(rup[0][ki+1]-rup[0][ki]))*(rup[0][ki-
    1]+rup[0][ki]))/(2*(((rup[1][ki]-rup[1][ki-1])/(rup[0][ki]-rup[0][ki-1]))-
    ((rup[1][ki+1]-rup[1][ki])/(rup[0][ki+1]-rup[0][ki]))));
1114.               ruprc[1][ki]=-((rup[0][ki]-rup[0][ki-1])/(rup[1][ki]-
    rup[1][ki-1]))*(ruprc[0][ki]-(rup[0][ki-1]+rup[0][ki])/2)+((rup[1][ki-
    1]+rup[1][ki])/2);
1115.           }
1116.           ruprc[0][nrup-1]=(((rup[1][nrup-1]-rup[1][nrup-
    2])/(rup[0][nrup-1]-rup[0][nrup-2]))*((rup[1][nrup]-rup[1][nrup-
    1])/(rup[0][nrup]-rup[0][nrup-1]))*(rup[1][nrup]-rup[1][nrup-
    2])+((rup[1][nrup-1]-rup[1][nrup-2])/(rup[0][nrup-1]-rup[0][nrup-
    2]))*(rup[0][nrup-1]+rup[0][nrup])-((rup[1][nrup]-rup[1][nrup-
    1])/(rup[0][nrup]-rup[0][nrup-1]))*(rup[0][nrup-2]+rup[0][nrup-
    1]))/(2*(((rup[1][nrup-1]-rup[1][nrup-2])/(rup[0][nrup-1]-rup[0][nrup-2]))-
    ((rup[1][nrup]-rup[1][nrup-1])/(rup[0][nrup]-rup[0][nrup-1]))));
1117.           ruprc[1][nrup-1]=-((rup[0][nrup-1]-rup[0][nrup-
    2])/(rup[1][nrup-1]-rup[1][nrup-2]))*(ruprc[0][nrup-1]-(rup[0][nrup-
    2]+rup[0][nrup-1])/2)+((rup[1][nrup-2]+rup[1][nrup-1])/2);
1118.       }
1119.
1120.
1121.
1122.       // REMESHING
1123.
1124.       // This procedure performs the remeshing by eliminating those
1125.       // segments smaller than rmin and dividing those which are
1126.       // greater than rmax into two parts and also keeps the grain
1127.       // boundary triple junction as a stable point.}
1128.
1129.       // Remeshing without grain boundary, (asimetric)
1130.
1131.       void remesh1() //MERT
1132.       {
1133.         int nrem,kz,ki,kl,kk,kt,f;
1134.         Number mag,mag1;
1135.         arr2 delrm;
1136.         f=0;
1137.         delr1(nrup, rup,delru, su);
1138.         psir(nrup-1,delru,su);
1139.         kappa(su,teta,delru,nrup-1);
1140.         nrem = nrup-1;
1141.
1142.         for(int kk=0;kk<=nrem;kk++)
1143.         {
1144.           rm[0][kk]=rup[0][kk];
1145.           rm[1][kk]=rup[1][kk];
1146.           rm[2][kk]=0;
1147.         }
1148.
1149.         for(int ki=0; ki<=nrem-2; ki++)
1150.           {
1151.
1152.           if (su[ki] >= rmax)
1153.             {
1154.             for(int kz=0; kz<=nrem-ki-1; kz++)
1155.             {
1156.               rm[0][nrem+1-kz+f]=rm[0][nrem-kz+f];
1157.               rm[1][nrem+1-kz+f]=rm[1][nrem-kz+f];
```

```
1158.            }
1159.
1160.                for (kl=0;kl<=2;kl++)
1161.                {
1162.                    delrm[kl][ki+f+1]=(rm[kl][ki+f+2]-
    ruprc[kl][ki+1])+(rm[kl][ki+f]-ruprc[kl][ki+1]);
1163.
1164.                }
1165.              for(int kj=0; kj<=2; kj++)
1166.            {
1167.                mag1=magnitude(delrm[0][ki+f+1],delrm[1][ki+f+1],delrm[2][k
    i+f+1]);
1168.                rm[kj][ki+f+1] = ruprc[kj][ki+1]+((1/kapkap[ki+1])*(kapkap[
    ki+1])/fabs(kapkap[ki+1]))*delrm[kj][ki+f+1]/mag1;
1169.             //   cout<< rm[kj][ki+f+1]<<"       "<<kapkap[ki+1]<<"
       "<<f<<endl;
1170.            }
1171.              f=f+1;
1172.              }
1173.            }
1174.          if (su[nrem-1] >= rmax)
1175.          {
1176.                rm[0][nrem+1+f]=rm[0][nrem+f];
1177.                rm[1][nrem+1+f]=rm[1][nrem+f];
1178.
1179.                for (kl=0;kl<=2;kl++)
1180.                  {
1181.                    delrm[kl][nrem+f]=(rm[kl][nrem+f+1]-ruprc[kl][nrem-
    1])+(rm[kl][nrem+f-1]-ruprc[kl][nrem-1]);
1182.
1183.                }
1184.              for(int kj=0; kj<=2; kj++)
1185.            {
1186.                mag1=magnitude(delrm[0][nrem+f],delrm[1][nrem+f],delrm[2][n
    rem+f]);
1187.                rm[kj][nrem+f] = ruprc[kj][nrem-1]+((1/kapkap[nrem-
    1])*(kapkap[nrem-1])/fabs(kapkap[nrem-1]))*delrm[kj][nrem+f]/mag1;
1188.            }
1189.              f=f+1;
1190.           }
1191.          for (kt=0;kt<=nrem+f;kt++)
1192.          {
1193.            rup[0][kt]=rm[0][kt];
1194.            rup[1][kt]=rm[1][kt];
1195.            rup[2][kt]=0;
1196.          }
1197.            nrup=nrem+f+1;
1198.        //  cout<<nrup<<endl;
1199.        }
1200.
1201.        void remesh0()   //AYTAC
1202.        {
1203.          int m, ka, crm,t=0;
1204.          Number mag, temp1, temp2, temp3;
1205.          arr2 rm;
1206.
1207.        nrem = nrup-1;
1208.        ka = 2;
1209.             for(int kj=0; kj<=2; kj++)
```

```
                rm[kj][0] = rup[kj][0] ;


       delr1(nrup,rup,delrup,sup);

       mag = magnitude(delrup[0][0],delrup[1][0],delrup[2][0]);
       if (mag >= rmax)
            {
            for(int kj=0; kj<=2; kj++)
              {
                  rm[kj][1] = (rup[kj][0] + rup[kj][1])*0.5;
                  rm[kj][2] = rup[kj][1];
              }
            ka = ka+1;
            }
       if (mag < rmax)
                  {
          if (mag > rmin) {
               for(int kj=0; kj<=2; kj++)
                 rm[kj][1] = rup[kj][1] ;
                 }
      }

      maxSeglenth = 0;


        for(int ki=1; ki<=nrem-2; ki++)
          {
              if (ki>nrem-2-t) break;
          mag = magnitude(delrup[0][ki+t],delrup[1][ki+t],delrup[2][ki+t]
    );
          if(maxSeglenth < mag) maxSeglenth = mag;

          if (mag >= rmax)
            {
            for(int kj=0; kj<=2; kj++)
              {
                  rm[kj][ka] = (rup[kj][ki+t] + rup[kj][ki+1+t])*0.5;
                  rm[kj][ka+1] = rup[kj][ki+1+t];
              }
            ka = ka+2;
            }
            else if (mag < rmax)
              {
                if (mag <= rmin){
              for(int kj=0; kj<=2; kj++)
      {

                  delrup[kj][ki-1] = delrup[kj][ki-
    1]+delrup[kj][ki+t]/2;
                  delrup[kj][ki] = delrup[kj][ki+1+t]+delrup[kj][ki+t]/2;

                  rm[kj][ka-1] = (rup[kj][ki+t] + rup[kj][ki+1+t])*0.5;
                  rm[kj][ka] = rup[kj][ki+2+t];
                  }
                  ka++;
                  t++;
                }
```

```
1266.
1267.             else if (mag > rmin)
1268.                   {
1269.                  for(int kj=0; kj<=2; kj++)
1270.                    rm[kj][ka] = rup[kj][ki+1+t] ;
1271.                  ka++ ;
1272.                    }
1273.
1274.
1275.           }
1276.           }
1277.        crm = ka-1;
1278.         mag = magnitude(delrup[0][nrem-1],delr[1][nrem-1],delr[2][nrem-
    1]);
1279.
1280.         if (mag >= rmax)
1281.           {
1282.           for(int kj=0; kj<=2; kj++)
1283.             {
1284.             rm[kj][crm+1] = (rm[kj][crm]+ rup[kj][nrem])*0.5 ;
1285.             rm[kj][crm+2] = rup[kj][nrem] ;
1286.             }
1287.           crm = crm+2;
1288.           }
1289.
1290.           if (mag < rmax)
1291.              {
1292.               for(int kj=0; kj<=2; kj++)
1293.               rm[kj][crm+1] = rup[kj][nrem];
1294.               crm = crm+1;
1295.              }
1296.
1297.         crm = crm+1;
1298.
1299.
1300.          for(int ki = 0; ki< crm ; ki++)
1301.           {
1302.                   rup[0][ki] = rm[0][ki];
1303.                   rup[1][ki] = rm[1][ki];
1304.                   rup[2][ki] = 0;
1305.           }
1306.
1307.         nrup = crm;
1308.
1309.          // delr1(nrup,rup,delrup,sup);
1310.
1311.           // int jj = 0;
1312.          // for(int ki = 0; ki< nrup ; ki++)
1313.           // {
1314.           // if(ki == 1){
1315.                // mag = magnitude(delrup[0][ki-1],delrup[1][ki-
    1],delrup[2][ki-1]);
1316.               // if(mag > maxSeglenth/4){
1317.                       // rup[0][jj] = rm[0][ki];
1318.                       // rup[1][jj] = rm[1][ki];
1319.                       // rup[2][jj] = 0;
1320.                       // jj++;
1321.                    // }
1322.           // }
```

```
1323.          // else if(ki == nrup-2){
1324.              // mag = magnitude(delrup[0][ki],delrup[1][ki],delrup[2][ki
    ]);
1325.              // if(mag > maxSeglenth/4){
1326.                      // rup[0][jj] = rm[0][ki];
1327.                      // rup[1][jj] = rm[1][ki];
1328.                      // rup[2][jj] = 0;
1329.                      // jj++;
1330.                  // }
1331.          // }
1332.          // else{
1333.              // rup[0][jj] = rm[0][ki];
1334.              // rup[1][jj] = rm[1][ki];
1335.              // rup[2][jj] = 0;
1336.              // jj++;
1337.          // }
1338.
1339.          // }
1340.
1341.        // nrup = jj;
1342.
1343.      }
1344.
1345.
1346.      void remeshEnd()   //AYTAC
1347.      {
1348.        int m, ka, crm;
1349.        Number mag, temp1, temp2, temp3;
1350.        arr2 rm;
1351.
1352.        for(int ki=0; ki<nrup; ki++)
1353.           for(int kj=0; kj<=2; kj++)
1354.                 rm[kj][ki] = rup[kj][ki] ;
1355.
1356.
1357.         delr1(nrup,rup,delrup,sup);
1358.      maxSeglenth = 0;
1359.
1360.           for(int ki=0; ki<nrup; ki++){
1361.             mag = magnitude(delrup[0][ki],delrup[1][ki],delrup[2][ki]);

1362.          if(maxSeglenth < mag) maxSeglenth = mag;
1363.          }
1364.
1365.
1366.      int jj = 0;
1367.         for(int ki = 0; ki< nrup ; ki++)
1368.          {
1369.              if(ki == 1){
1370.                  mag = magnitude(delrup[0][ki-1],delrup[1][ki-
    1],delrup[2][ki-1]);
1371.                  if(mag > maxSeglenth/4){
1372.                          rup[0][jj] = rm[0][ki];
1373.                          rup[1][jj] = rm[1][ki];
1374.                          rup[2][jj] = 0;
1375.                          jj++;
1376.                      }
1377.              }
1378.              else if(ki == nrup-2){
```

```
1379.                   mag = magnitude(delrup[0][ki],delrup[1][ki],delrup[2][k
    i]);
1380.                   if(mag > maxSeglenth/4){
1381.                           rup[0][jj] = rm[0][ki];
1382.                           rup[1][jj] = rm[1][ki];
1383.                           rup[2][jj] = 0;
1384.                           jj++;
1385.                       }
1386.               }
1387.               else{
1388.                   rup[0][jj] = rm[0][ki];
1389.                   rup[1][jj] = rm[1][ki];
1390.                   rup[2][jj] = 0;
1391.                   jj++;
1392.               }
1393.
1394.           }
1395.
1396.         nrup = jj;
1397.
1398.       }
1399.
1400.
1401.       void remeshlower()
1402.       {
1403.           delrlower=fabs(rdown[0][2*fmn+5]-rdown[0][2*fmn+4]);
1404.           int j,i,k;
1405.           delrend=fabs(rdown[0][2*fmn-1]-rdown[0][2*fmn]);
1406.           delrfirst=fabs(rdown[0][nrdown-2*fmn]-rdown[0][nrdown-2*fmn-
    1]);
1407.           if (delrend > delrlower*3/2 || delrfirst > delrlower*3/2)
1408.           {
1409.
1410.               if (delrend > delrlower*3/2){
1411.                   for (j=0; j<=2; j++){
1412.                   for (i=0;i<=nrdown-1-2*fmn;i++)
1413.                   {
1414.               rdown[j][nrdown-i]=rdown[j][nrdown-i-1];
1415.                   }
1416.               }
1417.               rdown[0][2*fmn]=rdown[0][2*fmn+1]+delrlower;
1418.               rdown[1][2*fmn]=0;
1419.               rdown[2][2*fmn]=0;
1420.               nrdown++;
1421.               }
1422.               if (delrfirst > delrlower*3/2){
1423.               //rdown[0][nrdown-2*fmn]=rdown[0][nrdown-1]-delrlower;
1424.               //rdown[1][nrdown]=0;
1425.               //rdown[2][nrdown]=0;
1426.               for (j=0; j<=2; j++){
1427.                   for (i=0;i<=2*fmn-1;i++)
1428.                   {
1429.               rdown[j][nrdown-i]=rdown[j][nrdown-i-1];
1430.                   }
1431.               }
1432.               rdown[0][nrdown-2*fmn]=rdown[0][nrdown-2*fmn-1]-
    delrlower;
1433.               rdown[1][nrdown-2*fmn]=0;
1434.               rdown[2][nrdown-2*fmn]=0;
```

```
1435.
1436.                 nrdown++;
1437.                 }
1438.           }
1439.           if (delrend < delrlower*1/4 || delrfirst < delrlower*1/4)
1440.           {
1441.               if (delrend < delrlower*1/4) {
1442.               for (j=0; j<=2; j++)
1443.                   for (i=2*fmn;i<=nrdown-2;i++)
1444.                   {
1445.               rdown[j][i]=rdown[j][i+1];
1446.                   }
1447.                   nrdown=nrdown-1;
1448.               }
1449.               if (delrfirst < delrlower*1/4){
1450.               for (j=0; j<=2; j++)
1451.                   for (i=nrdown-2*fmn;i<=nrdown-1;i++)
1452.                   {
1453.               rdown[j][i-1]=rdown[j][i];
1454.                   }
1455.                   nrdown=nrdown-1;
1456.               }
1457.               }
1458.               // cout<<fmn<<endl;
1459.           }
1460.
1461.
1462.
1463.
1464.       // OGURTANI MODEL
1465.       // Void- Grain Boundary Interactions
1466.       // Under the Effect of Electron Wind and Thermal Stresses
1467.       // by using Indirect BEM Calculations
1468.
1469.       // Finite Strip
1470.
1471.       // Interconnect node velocities
1472.
1473.       void calnew()
1474.       {
1475.       vmax =0;
1476.         // for(int kj=0; kj<=nrup-1; kj++)
1477.           // vel[kj]=0;
1478.
1479.         for(int kj=1; kj<=nrup-2; kj++)
1480.           {
1481.            vel[kj] = 2*(dif[kj]/s[kj]*(Psiu[kj+1]-Psiu[kj])-dif[kj-
    1]/s[kj-1]*(Psiu[kj]-Psiu[kj-1]))*ho*ho/(s[kj]+s[kj-1])-
    Mb*(delGb+(TauS[kj]*kapkap[kj]*Wcal[kj]-
    Wpot[kj])*ho);  // TauS[i]*kapkap[i]*Wcal[i]-Wpot[i]
1482.           }
1483.
1484.         vel[1] = 2*(dif[1]/s[1]*(Psiu[2]-Psiu[1]))*ho*ho/(s[1]+2*s[0])-
    Mb*(delGb+(TauS[1]*kapkap[1]*Wcal[1]-
    Wpot[1])*ho);  // TauS[i]*kapkap[i]*Wcal[i]-Wpot[i]
1485.         vel[nrup-2] = -2*(dif[nrup-3]/s[nrup-3]*(Psiu[nrup-2]-Psiu[nrup-
    3]))*ho*ho/(2*s[nrup-2]+s[nrup-3])-Mb*(delGb+(TauS[nrup-2]*kapkap[nrup-
    2]*Wcal[nrup-2]-Wpot[nrup-2])*ho);  // TauS[i]*kapkap[i]*Wcal[i]-Wpot[i]
1486.
```

```
1487.
1488.         vel[0] = Mg*delta/2/omega*(lamdag-(cos(thetaL)));
1489.         vel[nrup-1] = Mg*delta/2/omega*(lamdag-(cos(thetaR)));
1490.         velL = vel[0];
1491.         velR = vel[nrup-1];
1492.
1493.         for(int kj = 1; kj<nrup-
    1; kj++)   if(fabs(vel[kj]) > vmax) vmax = fabs(vel[kj]);
1494.       }
1495.
1496.       // Void node velocities
1497.
1498.       void calruv()
1499.       {
1500.         for(int kj=1; kj<=nrup-2; kj++)
1501.           {
1502.           rup[0][kj]=rup[0][kj]+deltat*vel[kj]*lln[0][kj];
1503.           rup[1][kj]=rup[1][kj]+deltat*vel[kj]*lln[1][kj];
1504.           rup[2][kj]=rup[2][kj]+deltat*vel[kj]*lln[2][kj];
1505.           }
1506.         rup[0][0]=rup[0][0]-deltat*vel[0];
1507.         rup[1][0]=rup[1][0];
1508.         rup[2][0]=rup[2][0];
1509.             for (int ki=0; ki<=2*fmn-1; ki++){
1510.               rdown[0][ki]=rdown[0][ki]+deltat*vel[nrup-1];
1511.               rdown[1][ki]=rdown[1][ki];
1512.               rdown[2][ki]=rdown[2][ki];
1513.             }
1514.         rup[0][nrup-1]=rup[0][nrup-1]+deltat*vel[nrup-1];
1515.         rup[1][nrup-1]=rup[1][nrup-1];
1516.         rup[2][nrup-1]=rup[2][nrup-1];
1517.               for (int j=nrdown-2*fmn; j<=nrdown-1; j++){
1518.               rdown[0][j]=rdown[0][j]-deltat*vel[0];
1519.               rdown[1][j]=rdown[1][j];
1520.               rdown[2][j]=rdown[2][j];
1521.             }
1522.       }
1523.
1524.       // this procedure generates the initial system
1525.
1526.       void generate()
1527.       {
1528.         uppart();
1529.         lowpart();
1530.       }
1531.
1532.       // this procedure gets the initial parameters from a file called An
    isoDrop_Stress.dat
1533.
1534.       void needparam(){
1535.        fp = fopen("input.dat", "r");
1536.           fgets(line, sizeof(line), fp);
1537.                                       sscanf(line, "%d", &newdata); //int

1538.           fgets(line, sizeof(line), fp);
1539.                                       sscanf(line, "%d", &type); //int
1540.       fgets(line, sizeof(line), fp);
1541.       fgets(line, sizeof(line), fp);
1542.           fgets(line, sizeof(line), fp);
```

```
1543.                                       sscanf(line, "%Lg", &ho); //double
1544.           fgets(line, sizeof(line), fp);
1545.                                       sscanf(line, "%Lg", &sl); //double
1546.           fgets(line, sizeof(line), fp);
1547.                                       sscanf(line, "%Lg", &sw); //double
1548.           fgets(line, sizeof(line), fp);
1549.                                       sscanf(line, "%Lg", &Amp); //double
1550.           fgets(line, sizeof(line), fp);
1551.                                       sscanf(line, "%d", &Modiv); //int
1552.           fgets(line, sizeof(line), fp);
1553.                                       sscanf(line, "%d", &Msin); //int
1554.           fgets(line, sizeof(line), fp);
1555.                                       sscanf(line, "%d", &nsw); //int
1556.       fgets(line, sizeof(line), fp);
1557.       fgets(line, sizeof(line), fp);
1558.           fgets(line, sizeof(line), fp);
1559.                                       sscanf(line, "%Lg", &ksi); //double
1560.       fgets(line, sizeof(line), fp);
1561.       fgets(line, sizeof(line), fp);
1562.           fgets(line, sizeof(line), fp);
1563.                                       sscanf(line, "%Lg", &ym); //double
1564.           fgets(line, sizeof(line), fp);
1565.                                       sscanf(line, "%Lg", &poisson); //do
    uble
1566.           fgets(line, sizeof(line), fp);
1567.                                       sscanf(line, "%Lg", &delGb); //int
1568.           fgets(line, sizeof(line), fp);
1569.                                       sscanf(line, "%Lg", &Mb); //double
1570.           fgets(line, sizeof(line), fp);
1571.                                       sscanf(line, "%Lg", &Mg); //double
1572.           fgets(line, sizeof(line), fp);
1573.                                       sscanf(line, "%d", &mint); //int
1574.           fgets(line, sizeof(line), fp);
1575.                                       sscanf(line, "%Lg", &Sigma); //doub
    le
1576.           fgets(line, sizeof(line), fp);
1577.                                       sscanf(line, "%Lg", &Eta); //double
1578.       fgets(line, sizeof(line), fp);
1579.       fgets(line, sizeof(line), fp);
1580.           fgets(line, sizeof(line), fp);
1581.                                       sscanf(line, "%Lg", &Aint); //doubl
    e
1582.           fgets(line, sizeof(line), fp);
1583.                                       sscanf(line, "%Lg", &Bint); //doubl
    e
1584.           fgets(line, sizeof(line), fp);
1585.                                       sscanf(line, "%Lg", &rtphi); //int
1586.           fgets(line, sizeof(line), fp);
```

```
1587.                                       sscanf(line, "%Lg", &hfn); //double

1588.       fgets(line, sizeof(line), fp);
1589.       fgets(line, sizeof(line), fp);
1590.           fgets(line, sizeof(line), fp);
1591.                                       sscanf(line, "%d", &nl); //double
1592.       fgets(line, sizeof(line), fp);
1593.           fgets(line, sizeof(line), fp);
1594.                                       sscanf(line, "%Lg", &gammaf); //dou
    ble
1595.       fgets(line, sizeof(line), fp);
1596.                                       sscanf(line, "%Lg", &gammas); //dou
    ble
1597.           fgets(line, sizeof(line), fp);
1598.                                       sscanf(line, "%Lg", &delw); //int
1599.           fgets(line, sizeof(line), fp);
1600.                                       sscanf(line, "%Lg", &lamdag); //dou
    ble
1601.           fgets(line, sizeof(line), fp);
1602.                                       sscanf(line, "%Lg", &epstime); //do
    uble
1603.           fgets(line, sizeof(line), fp);
1604.                                       sscanf(line, "%Lg", &delta); //doub
    le
1605.       fgets(line, sizeof(line), fp);
1606.       fgets(line, sizeof(line), fp);
1607.           fgets(line, sizeof(line), fp);
1608.                                       sscanf(line, "%Lg", &rmin); //int
1609.           fgets(line, sizeof(line), fp);
1610.                                       sscanf(line, "%Lg", &rmax); //int
1611.       fgets(line, sizeof(line), fp);
1612.           fgets(line, sizeof(line), fp);
1613.                                       sscanf(line, "%d", &rem); //int
1614.           fgets(line, sizeof(line), fp);
1615.                                       sscanf(line, "%d", &contData); //in
    t
1616.           fgets(line, sizeof(line), fp);
1617.                                       sscanf(line, "%d", &fmn); //int
1618.
1619.       if(ho == 0) ho = 0.95/sqrt(1+(4*sl*sl)) ;
1620.
1621.        Amp = Amp*ho;
1622.        gfm = (gammaf+gammas)/2/gammaf;  //
1623.        sl = sl*ho;                      //  scaled strip length
1624.        sw = sw*ho;                      //  scaled strip width
1625.
1626.        lamda = 2*sl/nsw;                //  normalized wave length
1627.        lamdau = 2*Msin/nsw;              //  node # in each wave length
1628.        kv = 2*pi/lamda;                  //  normalized wave vector
1629.
1630.        omega = delta*delta*delta;        // the atomic volume
1631.
1632.        epsx = 1-poisson;                 // the misfit strain energy
1633.
1634.        rtphi = rtphi*pi/180;             // texture tilt angle in rad
1635.
1636.        dmean = lamda/lamdau;             // mean segment length
1637.        rmin = rmin*ho;                  // minimum segment length
1638.        rmax = rmax*ho;                  // maximum segment length
```

```
1639.
1640.        // Lamm coefficients of elasticity
1641.        lamG = ym/(2*(1+poisson));
1642.        lammu = ym/(2*(1+poisson));      // lamg and lammu are identical
1643.
1644.        lamlamda = ym*poisson/((1+poisson)*(1-2*poisson));
1645.        cc = 1/(8*pi*lamG*(1-poisson));
1646.        c1 = 3-4*poisson;
1647.        c2 = 1/(4*pi*(1-poisson));
1648.        c3 = 1-2*poisson;
1649.        c4 = 2;
1650.
1651.        // Elastic Stiffness and Compliance Matrices given in various form
    ats
1652.        cc1 = ym*(1-poisson)/((1+poisson)*(1-2*poisson));
1653.        cc2 = poisson/(1-poisson);
1654.
1655.       deltat =epstime*dmean/1000;
1656.         lastOutNum = 0;
1657.         mpow =0;
1658.       }
1659.
1660.
1661.
1662.       void writeParam(){
1663.
1664.           out.open("inputt.dat");
1665.
1666.           out << newdata << endl;
1667.           out << type << endl;
1668.           out << ho << endl;
1669.           out << sl << endl;
1670.           out << sw << endl;
1671.           out << Amp << endl;
1672.           out << Modiv << endl;
1673.           out << Msin << endl;
1674.           out << nsw << endl;
1675.           out << ksi << endl;
1676.           out << ym << endl;
1677.           out << poisson << endl;
1678.           out << delGb << endl;
1679.           out << Mb << endl;
1680.           out << Mg << endl;
1681.           out << mint << endl;
1682.           out << Sigma << endl;
1683.           out << Eta << endl;
1684.           out << Aint << endl;
1685.           out << Bint << endl;
1686.           out << rtphi << endl;
1687.           out << hfn << endl;
1688.           out << nl << endl;
1689.           out << gammaf << endl;
1690.           out << gammas << endl;
1691.           out << delw << endl;
1692.           out << lamdag << endl;
1693.           out << epstime << endl;
1694.           out << delta << endl;
1695.           out << rmin << endl;
1696.           out << rmax << endl;
```

```
1697.           out << rem << endl;
1698.           out << contData << endl;
1699.           out << fmn << endl; //int
1700.
1701.           out.close();
1702.       }
1703.       // Program Restart Procedure
1704.
1705.       void getcontparam()
1706.       {
1707.       Number sil;
1708.
1709.       ifstream in;
1710.       in.open("cont.txt");
1711.
1712.       in >> ru[0][0] >> ru[1][0];
1713.       in >> sil >> sil >> sil >> sil >> sil >> sil >> sil >> sil;
1714.       in >> nu;
1715.       in >> t >> ms>> timex >> mm;
1716.
1717.       for(int i=1; i<=nu-1; i++){
1718.           in >>ru[0][i] >>ru[1][i];
1719.           in >> sil >> sil >> sil >> sil >> sil >> sil >> sil >> sil;
1720.               }
1721.
1722.       lowpart();
1723.
1724.       }
1725.
1726.
1727.       void recordtimestep(){
1728.
1729.
1730.           if (t==mpow || t==nl){
1731.
1732.               //  time ( &rawtime );
1733.
1734.
1735.               int2str(lastOutNum);
1736.
1737.               //textName = dirName+textName;//
1738.               out << setiosflags(ios::showpoint);
1739.               out.open(textName.c_str(),ios::trunc );
1740.               //outName   = ctime(&rawtime);
1741.                                           //      1           //      2
          //      3
1742.               out << setprecision(20) << rud[0][0]/1    <<" "<< rud[1][0]
     /1  <<" "<< s[0]/1         <<"  "
1743.                                           //      4           //      5
          //      6
1744.                                           << tetaw[0]         <<" "<< kapkap[
     0]*1  <<" "<< TauO[0]        <<" "
1745.                                           //      7           //      8
          //      9
1746.                                           << TauS[0]          <<" " << TauD[0
     ]     <<" "<< dif[0]         <<" "
1747.                                           //      10          //      11
          //      12
```

```
                                        << tetau[0]       <<" "<< hoop[0]
      <<" "<< Sighoop[0]     <<"   "
                                        //      13          //      14
       //      15
                                        << Psiu[0]        <<" "<< vel[0]
      <<" "<< nrup           <<" "
                                        //      16          //      17
       //      18
                                        << nrdown         <<" " << t
      <<" "<< lastOutNum     <<" "
                                        //      19          //      20
       //      21
                                        << timex          <<" "<< nrud
      <<" "<< dmean
                                        << endl;

            for(int i=1; i<nrud;i++){
            out << setprecision(20)
                                        << rud[0][i]/1 <<" "<< rud[1][i]/1
     <<" " << s[i]/1      <<" "
                                        << tetaw[i]      <<" "<< kapkap[i]*
   1    <<" "   << TauO[i]     <<" "
                                        << TauS[i]       <<" "<< TauD[i]
       <<" "   << dif[i]      <<" "
                                        << tetau[i]    <<" "<< hoop[i]
     <<" " << Sighoop[i]  <<"   "
                                        << Psiu[i]     <<" "<< vel[i]
                                        << endl;
            }
            out << endl ;


            out.close();

          lastOutNum++ ;

            if (t < 256) mpow =pow(2,(double)lastOutNum);
        else if (t < 1000) mpow =300+200*(lastOutNum-9);
        else if (t < 10000) mpow =1000+500*(lastOutNum-13);
        else if (t < 20000) mpow =10000+1000*(lastOutNum-31);
        else if (t < 100000) mpow =20000+5000*(lastOutNum-41);
        else /* if (t < 1000000) */ mpow =100000+10000*(lastOutNum-
   57);


        }

        if (numContData==10000  || t==nl){

            //time ( &rawtime );

            int2str(lastOutNum);

            textName = "cont.dat";//
            out << setiosflags(ios::showpoint);
            out.open(textName.c_str(),ios::trunc );
            //outName   = ctime(&rawtime);

```

```
1794.              out << setprecision(20) << rud[0][0] <<" "<< rud[1][0]<<"
   "
1795.              << nrup  <<" " << nrdown <<" "<< t <<" "<< lastOutNum <<" "
   << timex     <<" "
1796.              << dmean   << endl; //min max
1797.
1798.              for(int i=1; i<nrud;i++){
1799.              out << setprecision(20)
1800.                      << rud[0][i]<<" "<< rud[1][i]<< endl;
1801.              }
1802.              out << endl ;
1803.
1804.              out.close();
1805.          //  clrscr();
1806.
1807.            numContData = 0;
1808.          }
1809.
1810.          numContData++ ;
1811.
1812.      }
1813.
1814.
1815.      // MAIN PROGRAM Organization Procedure
1816.
1817.
1818.
1819.      void final()
1820.      {
1821.
1822.      stacksvi(rup,nrup,rdown,nrdown);
1823.
1824.      while(t < nl){
1825.
1826.      delr1(nrup,rup,delrup,sup);
1827.      psir(nrup-1,delrup,sup);
1828.
1829.                          for(int ki=0; ki<nrup; ki++) tetau[ki] = teta[ki];

1830.
1831.      kappa(sup,teta,delrup,nrup-1);
1832.
1833.      delr1(nrdown,rdown,delrdown,sdown);
1834.      deldelr1(nrud,1,rud);
1835.
1836.      psir(nrdown-1,delrdown,sdown);
1837.      psipsir(nrup-1, delrup, sup);
1838.      ddif() ;
1839.
1840.      nocRT(s,delr,nrup,nrdown);
1841.
1842.      rcc(rud,nrud);
1843.
1844.          if(Sigma != 0 || Eta !=0 ){
1845.              ftin(s,delr);
1846.              asym() ;
1847.              boundary() ;
1848.              pbfv() ;
1849.              fc();
```

```
1850.               dsglarge();
1851.               multa() ;
1852.               SigStress() ;
1853.               SigNodet() ;
1854.           }
1855.
1856.           // if (ksi==0)
1857.               // for(int i=0; i<=nrup-1; i++)
1858.                    // fieldtn[i] =0.0;
1859.
1860.
1861.            for(int i=0; i<nrup ;i++){
1862.                   Wpot[i] = (1/sqrt(1+sqr(tan(tetaw[ki]))))* (gammaf-
    gammas)/(pi*gammaf)*(delw*ho)/(sqr(delw*ho)+sqr(rup[1][i]));
1863.                   //  -lln[i][1] * (gammas-
    gammaf)/pi*(deltaW*ho)/(sqr(deltaW*ho)+4*sqr(rud[i][1]));
1864.                   }
1865.
1866.            for(int i=0; i<nrup ;i++){
1867.                   Wcal[i] = (gammas+gammaf)/(2*gammaf) - (gammas-
    gammaf)/(pi*gammaf) * atan( (rup[1][i]) / (delw*ho) )  ;
1868.               }
1869.
1870.           for(int i=0; i<= nrup-1 ; i++){
1871.           Psiu[i] = (kapkap[i]*Wcal[i]-Wpot[i])*ho;
1872.           if(ksi != 0)    Psiu[i] +=ksi*fieldtn[i];
1873.           if(Sigma != 0 || Eta !=0 )  Psiu[i] += -
    Sigma*Sighoop[i]*Sighoop[i]+Eta*Sighoop[i];
1874.               }
1875.
1876.           calnew();
1877.
1878.       //{calculation of record time step}
1879.
1880.           recordtimestep();
1881.           deltat =epstime*dmean/vmax;
1882.           calruv() ;
1883.
1884.           if(rem == 1 && t%20==0)
1885.           {
1886.           remesh0();
1887.           }
1888.           if(rem == 2&& t%20==0)
1889.           {
1890.           centerpoint();
1891.           remesh1();
1892.           }
1893.           remeshEnd();
1894.           remeshlower();
1895.           stacksvi(rup,nrup,rdown,nrdown);
1896.           timex = timex+deltat;
1897.           t++ ;
1898.           if(lastOutNum>999) t = nl;
1899.       }
1900.
1901.       }
```