**ร่างรายงานฉบับสมบูรณ์**

# โครงการพัฒนาซอฟต์แวร์เพื่อใช้ศึกษาพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศ

# Software development for the study of low-temperature atmospheric plasmas


**โดย**

**พงษ์แพทย์ เพ่งวาณิชย์**



**งบประมาณสนับสนุนจากสำนักงานคณะกรรมการการอุดมศึกษาและสำนักงานกองทุนสนับสนุนการวิจัย**



**คณะวิศวกรรมศาสตร์**

**จุฬาลงกรณ์มหาวิทยาลัย**

# กิตติกรรมประกาศ

คณะผู้วิจัยมีความทราบซึ้งและขอขอบคุณการสนับสนุนจากบุคคลและภาคส่วนต่างๆ ที่ช่วยให้การวิจัยสำเร็จลุล่วงด้วยดี ทั้งด้านงบประมาณจากสำนักงานคณะกรรมการการอุดมศึกษาและสำนักงานกองทุนสนับสนุนการวิจัย ด้านคำปรึกษาจาก รศ.ดร. ธวัชชัย อ่อนจันทร์ และ Professor Yue Ying Lau ด้านเครื่องมือและสถานที่จากภาควิชาวิศวกรรมนิวเคลียร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

## คำนำ

รายงานฉบับนี้ เป็นผลการดำเนินการของงานวิจัยเรื่อง “การพัฒนาซอฟต์แวร์เพื่อใช้ศึกษาพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศ” ซึ่งเป็นโครงการวิจัยที่ได้รับงบประมาณสนับสนุนจากสำนักงานคณะกรรมการการอุดมศึกษาและสำนักงานกองทุนสนับสนุนการวิจัยปี 2554 โดยแบ่งออกเป็น 5 บท คือ บทนำ ทฤษฎีและงานวิจัยที่เกี่ยวข้อง โปรแกรมการจำลอง ผลการทดสอบโปรแกรมและเครื่องกำเนิดพลาสมาแบบ DBD และสรุปงานวิจัย โดยมีรายละเอียดเพิ่มเติมของผลงานในภาคผนวก

## บทคัดย่อ

Low-temperature atmospheric plasmas (LTAP) have found their utilization in many of the modern technologies such as industrial material's surface coating and modification, medical sterilization, and chemical processing. The low-temperature property allows the LTAP to interact with various low-heat-resistant materials without damaging them, while the atmospheric property allows the plasma to be generated and controlled without a cumbersome vacuum system. Many of the current LTAP researches focus on identifying the discharge mechanisms and evaluating the LTAP's effects on various materials.

In this research, the LTAP generated in a DBD plasma source will be numerically simulated. The goal is to create a software that enables more understanding of the discharge mechanisms in a DBD, specially the microdischarge, which is one of the discharge characteristics of a DBD, and the interactions between the charged particles in the plasma and the neutral particles in air, which are non-negligible at atmospheric pressure. The simulation will utilize the particle-in-cell (PIC) and the Monte Carlo (MC) methods in order to follow various microscopic and localized interactions that can occur in the plasma. For instance, the PIC method is used to model the interactions between the charge particles and the local electric field, whereas the MC method is used to model the collisions between the plasma's charged particles and the air's neutral particles. Both methods must be solved simultaneously in order to obtain a "more" complete model of the discharge.

While the software will initially be created for only a specific case, i.e. LTAP in DBD, it will be designed with an ability to allow for further expansion to model more general cases in mind. Ultimately, the long-term goal is to upgrade the software so that it can be used for both educational purposes (e.g. for students studying plasmas) and other general purposes (e.g. for designing industrial plasma sources).

# สารบัญ

# สารบัญรูป

# บทที่ 1 บทนำ

## ความสำคัญและที่มาของปัญหาที่ทำการวิจัย

ปัจจุบันการประยุกต์ใช้พลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศ (Low-Temperature Atmospheric Plasma หรือ LTAP) ในงานด้านอุตสาหกรรม การแพทย์ และในงานวิจัยด้านวัสดุและเคมี กำลังเป็นที่นิยมเพิ่มขึ้นเรื่อย ๆ ทั้งนี้เนื่องจากการผลิตพลาสมาชนิดนี้ไม่จำเป็นต้องพึ่งระบบสุญญากาศ และพลาสมาที่ได้ยังมีอุณหภูมิต่ำ (อุณหภูมิของก๊าซอยู่ที่ระดับใกล้เคียงหรือเทียบเท่าอุณหภูมิห้อง) จึงสะดวกต่อการนำไปใช้ในระบบเปิดและในงานที่พลาสมาต้องสัมผัสกับวัสดุที่ไม่สามารถทนความร้อนสูงได้ เช่น พลาสติก โพลิเมอร์ สิ่งทอ หรือแม้แต่เซลล์ของสิ่งมีชีวิตต่าง ๆ ซึ่งรวมถึงเซลล์ผิวหนังด้วย

งานวิจัยนี้จะมุ่งเน้นไปที่การศึกษา LTAP ที่ผลิตจากไดอิเล็กทริกแบริเออร์ดิสชาร์จ (Dielectric Barrier Discharge หรือ DBD) ซึ่งเป็นแหล่งกำเนิดพลาสมาชนิดหนึ่งที่สามารถออกแบบเพื่อใช้ในการผลิต LTAP ได้ แม้ว่า DBD จะถูกค้นพบและมีการใช้งานมาเป็นเวลานานแล้วก็ตาม แต่การจำลองการเกิดพลาสมาใน DBD และปฏิกิริยาระหว่างพลาสมาใน DBD กับสสารชนิดต่าง ๆ ยังไม่สามารถทำได้อย่างสมบูรณ์ เนื่องจากลักษณะการดิสชาร์จ (Discharge) ของพลาสมาใน DBD ที่แตกต่างจากพลาสมาโดยทั่วไป นอกจากนี้ ที่ความดันบรรยากาศการชนกันระหว่างอนุภาคในพลาสมากับอนุภาคในอากาศยังไม่สามารถมองข้ามได้ ดังนั้นการจำลองพลาสมาด้วยวิธีพาร์ติเคิลอินเซลล์ (Particle-in-Cell หรือ PIC) ซึ่งเป็นวิธีที่ใช้โดยทั่วไปจึงไม่เพียงพออีกต่อไป

งานวิจัยเกี่ยวกับ LTAP ที่ผ่านมาส่วนใหญ่จะมุ่งเน้นไปที่การนำ LTAP ไปใช้ในการทดลอง ซึ่งมีข้อจำกัดที่สำคัญ คือ ผลการทดลองจะขึ้นอยู่กับแหล่งกำเนิดพลาสมาที่เลือกใช้ซึ่งผลิตพลาสมาได้ในช่วงพารามิเตอร์ที่จำกัด ดังนั้น จึงไม่สามารถบอกได้ว่าพลาสมาที่ผลิตจากแหล่งกำเนิดชนิดใดหรือพลาสมาที่มีคุณสมบัติอย่างไรจะให้ผลดีที่สุด งานวิจัยนี้มีเป้าหมายที่จะใช้การจำลองแบบทางคอมพิวเตอร์มาแก้ไขบางส่วนของข้อจำกัดดังกล่าว ซึ่งหากสามารถจำลอง LTAP ที่ผลิตจาก DBD ได้ จะสามารถคำนวณหาคุณสมบัติของ LTAP ที่เกิดขึ้นที่ช่วงพารามิเตอร์ต่าง ๆ และนำไปศึกษาปฏิกิริยาระหว่างพลาสมากับสสารอื่น ๆ ต่อไปได้ นอกจากนี้ยังสามารถนำแบบจำลองไปช่วยในการออกแบบ DBD ให้ได้ LTAP ที่มีคุณสมบัติตามต้องการได้ด้วย

ทั้งนี้ คาดว่าแบบจำลองจะสามารถใช้ได้กับพลาสมาชนิดอื่น ๆ ที่ผลิตโดย DBD ได้ด้วย ซึ่งจะเป็นการเอื้อประโยชน์ให้กับอุตสาหกรรม การแพทย์ และงานวิจัยที่จำเป็นต้องออกแบบ DBD ในอนาคต

**วัตถุประสงค์ของโครงการ**

- เพื่อจำลองพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศที่ผลิตจากแหล่งกำเนิดพลาสมาชนิดไดอิเล็กทริกแบริเออร์ดิสชาร์จ
- เพื่อศึกษาปฏิกิริยาที่เกิดขึ้นระหว่างอนุภาคในพลาสมาและอนุภาคในอากาศ
- เพื่อนำแบบจำลองที่ได้ไปใช้ประกอบการศึกษาด้านพลาสมา ทั้งในมหาวิทยาลัยและในการใช้งานทั่วไป

**ระเบียบวิธีวิจัย**

การวิจัยจะแบ่งเป็นสองส่วน คือ การสร้างแบบจำลองทางคอมพิวเตอร์ และ การสร้างเครื่องผลิต LTAP ชนิด DBD อย่างง่ายเพื่อใช้เปรียบเทียบกับแบบจำลอง ทั้งสองส่วน (ดังรายละเอียดในข้อ 8.1. และ 8.2.) สามารถดำเนินการไปพร้อมกันได้ โดยมีแผนการวิจัยดังต่อไปนี้

1. สร้างแบบจำลองทางคอมพิวเตอร์
    1.1. รวบรวมข้อมูลและเอกสารอ้างอิงที่เกี่ยวข้อง
    1.2. กำหนดรายละเอียดของระบบที่จะทำการจำลองเพื่อจำกัดขอบเขตของปัญหา ในเบื้องต้นผู้วิจัยจะจำกัดระบบที่จะจำลองไว้ไม่เกิน 2 มิติ (2D System) ที่เป็นสี่เหลี่ยมหรือวงกลม
    1.3. กำหนดพารามิเตอร์ที่ต้องการศึกษา (คำนวณ)
    1.4. เลือกวิธีการสร้างแบบจำลองทางคอมพิวเตอร์ที่เหมาะสม จากการค้นคว้าเบื้องต้นผู้วิจัยพบว่าวิธี PIC-MC เป็นวิธีที่ตรงกับวัตถุประสงค์ที่ต้องการ ซึ่งแบ่งเป็นสองวิธีย่อย คือ
        - PIC หรือ Particle-in-Cell Method เป็นวิธีทั่วไปที่ใช้ในการติดตามอนุภาคในพลาสมา เริ่มจากการสร้าง Computational Particles ขึ้นเป็นตัวแทนอนุภาคต่าง ๆ ในพลาสมา โดยในแต่ละช่วงเวลาจะติดตามการเคลื่อนที่ของ Computational Particles อันเป็นผลจากอิทธิพลของสนามไฟฟ้า และการเปลี่ยนแปลงของสนามไฟฟ้าอันเป็นผลจากการเคลื่อนที่ของ

Computational Particles ค่าต่าง ๆ ที่เกี่ยวกับ Computational Particles และสนามไฟฟ้าจะถูกเก็บไว้เพื่อใช้ประเมินคุณลักษณะของพลาสมาโดยรวม

- MC หรือ Monte-Carlo Method ใช้สำหรับจำลองปฏิกิริยาระหว่างอนุภาคในพลาสมาและอนุภาคในอากาศ วิธีนี้จะพิจารณาเหตุการณ์ชน (Collision Events) ระหว่างอนุภาคในพลาสมาและอนุภาคในอากาศ โดยจะประเมินว่าอนุภาคในพลาสมาตัวใดเกิดการชน มีปฏิกิริยาอะไรเกิดขึ้นจากการชน และมีการถ่ายทอดโมเมนตัมอย่างไร

1.5. วางแผนการสร้างแบบจำลองโดยเลือก Numerical Algorithms ที่เหมาะสม

1.6. สร้างแบบจำลอง

2. การสร้างเครื่องผลิต LTAP ชนิด DBD

2.1. รวบรวมข้อมูลและเอกสารอ้างอิงที่เกี่ยวข้อง

2.2. ออกแบบระบบที่จะสร้าง ในกรณีนี้จะต้องเป็นระบบที่อยู่ในขอบเขตที่แบบจำลองจะสามารถประเมินได้ ในเบื้องต้นผู้วิจัยคาดว่าจะใช้รูปทรงแบบง่าย คือ แบบ Planar หรือแบบ Cylindrical นอกจากนี้ การออกแบบเครื่องกำเนิดพลาสมา จะเน้นการออกแบบให้สามารถนำไปประยุกต์ใช้ในงานทางการแพทย์ด้าน Plasma Sterilization ต่อไปได้

2.3. รวบรวมอุปกรณ์สำหรับสร้างระบบ

2.4. สร้างระบบและทดสอบการผลิตพลาสมา

2.5. ศึกษาลักษณะของพลาสมาที่ผลิตได้และทำการวัดอุณหภูมิของอิเล็กตรอน ไอออน และก๊าซ

3. เมื่อดำเนินการในข้อ 1. และ 2. เรียบร้อยแล้ว ในขั้นตอนนี้จะนำระบบที่สร้างใน 2. ไปประเมินด้วยแบบจำลองใน 1. และเปรียบเทียบกับผลที่ได้จากการวัดและการคำนวณ

4. วิเคราะห์ข้อมูล จัดทำรายงาน และนำเสนอผลงาน

**ขอบเขตของการวิจัย**

1.1. ศึกษาและพัฒนาการจำลองแบบทางคอมพิวเตอร์ของพลาสมาด้วยวิธี Particle-in-Cell

1.2. ศึกษาและพัฒนาการจำลองแบบทางคอมพิวเตอร์ของปฏิกิริยาระหว่างอนุภาคในพลาสมาและอนุภาคในอากาศที่ความดันบรรยากาศด้วยวิธี Monte-Carlo

1.3. ศึกษาและพัฒนาการจำลองแบบทางคอมพิวเตอร์ของพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศในแหล่งกำเนิดพลาสมาชนิด DBD

1.4. ศึกษาและพัฒนาเครื่องกำเนิดพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศชนิด DBD

1.5. วิเคราะห์คุณสมบัติของพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศซึ่งรวมถึงอุณหภูมิของอิเล็กตรอน ไอออน และก๊าซ

2. **แผนการดำเนินงานตลอดโครงการ**

2.1. กิจกรรมและแผนการดำเนินงานเป็นไปตามตารางดังต่อไปนี้

| กิจกรรม | ปีที่ 1 | | | | ปีที่ 2 | | | |
|---|---|---|---|---|---|---|---|---|
| | ไตรมาส1 | ไตรมาส2 | ไตรมาส3 | ไตรมาส4 | ไตรมาส5 | ไตรมาส6 | ไตรมาส7 | ไตรมาส8 |
| รวบรวมข้อมูลวิจัยที่เกี่ยวข้อง | ■ | ■ | | ■ | | ■ | | ■ |
| ออกแบบแบบจำลองและคัดเลือก Algorithms ที่จะใช้ | | ■ | | | | | | |
| สร้างแบบจำลองในส่วนของ PIC | | ■ | ■ | ■ | | | | |
| สร้างแบบจำลองในส่วนของ MC | | | | ■ | ■ | ■ | | |
| ออกแบบเครื่องกำเนิด LTAP ชนิด DBD และรวบรวมอุปกรณ์ | | ■ | | | | | | |
| สร้างเครื่องกำเนิด LTAP | | | ■ | ■ | | | | |
| สร้าง Langmuir Probe | | | | ■ | | | | |
| เดินเครื่องกำเนิด LTAP และวัดพารามิเตอร์ต่าง ๆ | | | | | ■ | ■ | | |
| ทดสอบแบบจำลองและเปรียบเทียบกับผลที่ได้จากเครื่องกำเนิด LTAP | | | | | | | ■ | |
| สรุปผลและจัดทำรายงาน | | | | ■ | | | | ■ |

2.2. ผล (Output) ที่จะได้ ได้แก่

2.2.1. แบบจำลองทางคอมพิวเตอร์ของพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศที่ผลิตจากเครื่องกำเนิดพลาสมาชนิด DBD

2.2.2. เครื่องกำเนิดพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศชนิด DBD ที่สามารถผลิต LTAP สำหรับใช้ในการทำลองต่อไปได้

2.3. ผลงานที่คาดว่าจะตีพิมพ์ในวารสารวิชาการนานาชาติ (อย่างน้อยหนึ่งผลงาน) ได้แก่

2.3.1. Simulation of a low-temperature atmospheric dielectric-barrier-discharge plasma

2.3.2. Atmospheric plasma simulation using PIC-MC method

3. **ประโยชน์ที่คาดว่าจะได้รับ**

3.1. ประโยชน์ที่จะได้รับจากโครงการวิจัยนี้ ได้แก่

3.1.1. พัฒนาองค์ความรู้ด้านพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศซึ่งมีศักยภาพในงานอุตสาหกรรม การแพทย์ และงานวิจัยด้านต่าง ๆ

3.1.2. พัฒนาแบบจำลองทางคอมพิวเตอร์ของพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศซึ่งสามารถช่วยในการออกแบบและ Optimization เครื่องมือผลิตพลาสมา

3.1.3. สามารถนำแบบจำลองทางคอมพิวเตอร์ที่ได้จากงานวิจัยนี้ไปเป็นเครื่องมือประกอบการศึกษาด้านพลาสมาเพื่อพัฒนาศักยภาพของนิสิตได้

3.1.4. สามารถนำแบบจำลองทางคอมพิวเตอร์ที่ได้จากงานวิจัยนี้ไปประยุกต์ใช้ในการศึกษาปฏิกิริยาระหว่างพลาสมาและวัสดุต่าง ๆ ต่อไปได้

3.1.5. สามารถนำแบบจำลองทางคอมพิวเตอร์ที่ได้จากงานวิจัยนี้ไปดัดแปลงให้สามารถจำลองแบบพลาสมาโดยทั่วไปในช่วงอุณหภูมิและความดันอื่นต่อไปได้

3.2. ชื่อผลงานที่คาดว่าจะตีพิมพ์ในวารสารวิชาการนานาชาติ เป็นไปตามที่ระบุไว้ในข้อ 11.2. ข้างต้น

3.3. ชื่อวารสารวิชาการนานาชาติที่คาดว่าจะตีพิมพ์ (อย่างน้อยหนึ่งที่) ได้แก่

3.3.1. IEEE Transactions of Plasma Sciences

3.3.2. Journal of Applied Physics

3.3.3. Applied Physics Letters

3.3.4. Physics of Plasmas

# บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

## ผลงานวิจัยที่เกี่ยวข้อง

ไดอิเล็กทริกแบริเออร์ดิสชาร์จ (Dielectric Barrier Discharge) หรือ DBD เป็นแหล่งกำเนิดพลาสมาชนิดหนึ่ง ซึ่งสามารถผลิตพลาสมาภายใต้ความดันตั้งแต่ 0.1 ถึง 10 เท่าของความดันบรรยากาศ DBD ประกอบด้วย ขั้วไฟฟ้า (อิเล็กโทรด) สองขั้วซึ่งถูกกั้นด้วยวัสดุที่มีคุณสมบัติเป็นไดอิเล็กทริก เช่น แก้ว พลาสติก อากาศ ก๊าซไนโตรเจน และก๊าซซัลเฟอร์เฮกซะฟลูออไรด์ เป็นต้น โดยทั่วไปแล้วในการผลิตพลาสมาด้วย DBD จะจ่ายความต่างศักย์ให้กับขั้วไฟฟ้าทั้งสองที่มีขนาดตั้งแต่ 1 ถึง 100 kV และมีความถี่ตั้งแต่ 50 Hz ถึง 1 MHz ซึ่งทั้งนี้ขึ้นอยู่กับระยะห่างระหว่างขั้วไฟฟ้าและความดันที่ใช้[1-8] ข้อได้เปรียบของ DBD เมื่อเทียบกับแหล่งกำเนิดพลาสมาชนิดอื่น ๆ คือ สามารถผลิตพลาสมาชนิด non-thermal ซึ่งให้อิเล็กตรอนที่มีอุณหภูมิสูง (เนื่องจากมีไดอิเล็กทริกกั้นระหว่างขั้วไฟฟ้าทั้งสอง จึงสามารถจ่ายความต่างศักย์สูงมาก ๆ ได้โดยไม่ต้องกังวลเกี่ยวกับการ arcing) และตัวอุปกรณ์ยังสามารถออกแบบให้มีขนาดใหญ่หรือเล็กได้ไม่ยากนัก จึงสามารถผลิตพลาสมาที่มีขนาดตามต้องการได้ นอกจากนี้ DBD ยังสามารถผลิตพลาสมาที่ความดันบรรยากาศได้ ซึ่งช่วยลดปัญญาที่เกี่ยวข้องกับการใช้งานระบบสุญญากาศ และยังทำให้การนำไปประยุกต์ใช้ในด้านต่าง ๆ ทำได้ง่ายขึ้นอีกด้วย[9] ปัจจุบัน DBD ถูกประยุกต์ใช้ในงานด้านการบำบัด การทำความสะอาด การปลูกฟิล์มบาง และการปรับเปลี่ยนคุณสมบัติของพื้นผิว[10-12] การผลิตสารเคมี[13] การบำบัดก๊าซ[14] การผลิตโอโซนสำหรับบำบัดน้ำ[15] การฆ่าเชื้อโรคทางการแพทย์[16] และการแสดงภาพบนจอพลาสมา เป็นต้น

พลาสมาใน DBD มีรูปแบบ (Mode) ของการเกิดที่เรียกว่าไมโครดิสชาร์จ (microdischarge) หรือสตรีมเมอร์เบรคดาวน์ (streamer breakdown) ซึ่งหากมองด้วยตาจะเห็นเป็นเป็นสายฟิลาเมนต์ (filaments) เล็ก ๆ หลายสายที่มีเส้นผ่านศูนย์กลางประมาณ 0.1 mm ฟิลาเมนต์แต่ละสายเกิดจากการวิ่งของประจุที่ได้จากการแตกตัวของก๊าซไปยังพื้นผิวของไดอิเล็กทริกเพื่อชดเชยสนามไฟฟ้าจากภายนอก ซึ่งกระบวนการจะเกิดขึ้นภายใน 1-10 ns และหลังจากนั้นฟิลาเมนต์ดังกล่าวก็จะหายไป ความหนาแน่นของกระแส (current density) ในฟิลาเมนต์แต่ละสายจะอยู่ที่ประมาณ 100 – 1000 A $cm^{-2}$ ส่วนความหนาแน่นของอิเล็กตรอน (electron density) จะอยู่ที่ประมาณ $10^{14}$ – $10^{15}$ $cm^{-3}$ และพลังงานของอิเล็กตรอนจะอยู่ในช่วง 1 – 10 eV

การศึกษาเกี่ยวกับกระบวนการเกิด microdischarge ได้รับความสนใจเป็นอย่างมาก ทั้งในลักษณะของการสร้างแบบจำลองการคำนวณทางคอมพิวเตอร์ (numerical simulation)[17-19] และในลักษณะของการทดลอง (experiment)[2-6] อย่างไรก็ตาม เนื่องจากระยะเวลาของการเกิด microdischarge แต่ละครั้งจะอยู่ในระดับ ns ทำให้การศึกษาด้วยการทดลองเป็นไปได้ยากและต้องอาศัยอุปกรณ์ที่มีราคาแพง เช่น กล้อง charge-coupled device (CCD) จึงทำให้การศึกษาในลักษณะของการสร้างแบบจำลองการคำนวณทางคอมพิวเตอร์เป็นทางเลือกที่น่าสนใจ

ได้มีผู้เสนอแบบจำลองการคำนวณทางคอมพิวเตอร์ของการเกิดพลาสมาใน DBD ไว้หลายแบบ หนึ่งในนั้นเป็นแบบจำลองชนิดหนึ่งมิติ (1D) ซึ่งอาศัยแบบจำลองการไหล (fluid model) ชนิด combined fluid model และวิธี finite element ในการคำนวณ[20-22] แบบจำลองนี้สามารถอธิบายการเกิด Townsend discharge และกระบวนการที่นำไปสู่การเกิด Townsend discharge ในระบบผลิตพลาสมาซึ่งประกอบด้วยขั้วไฟฟ้าแบบแบนราบ (planar) และสามารถวิเคราะห์ Paschen curve ของระบบได้ แบบจำลองนี้สามารถเพิ่มการคำนวณในส่วนที่เป็นเคมีของพลาสมา (plasma chemistry) เพื่อศึกษาอนุภาคต่าง ๆ ที่เกิดขึ้นในกระบานการดิสชาร์จได้[23] แต่อย่างไรก็ตาม การจำลองนี้มิได้พิจารณาปฏิกิริยาระหว่างอนุภาคในพลาสมาและอนุภาคในก๊าซซึ่งเกิดขึ้นที่ความดันสูง และยังเป็นการจำลองในหนึ่งมิติซึ่งไม่สามารถใช้อธิบายการกระจายตัวของฟิลาเมนต์ซึ่งเป็นคุณสมบัติเฉพาะของพลาสมาใน DBD ได้

แบบจำลองการคำนวณทางคอมพิวเตอร์อีกแบบหนึ่งอาศัยวิธี Particle-in-Cell[24] ร่วมกับวิธี Monte Carlo[25-26] ในส่วนของ Particle-in-Cell หรือ PIC จะทำการติดตามการเคลื่อนที่ของอนุภาคที่มีประจุต่าง ๆ ในพลาสมาภายใต้อิทธิพลของสนามไฟฟ้า และทำการติดตามการเปลี่ยนแปลงของสนามไฟฟ้าอันเนื่องมาจากการเคลื่อนที่ของอนุภาคข้างต้น ในแต่ละขั้นเวลา (time step) ข้อมูลเกี่ยวกับตำแหน่ง ความเร็ว โมเมนตัม และการกระจายตัวของอนุภาค และข้อมูลเกี่ยวกับความต่างศักย์และสนามไฟฟ้าจะถูกบันทึกไว้เพื่อใช้ในการคำนวณค่าในขั้นเวลาถัดไป ในส่วนของ Monte Carlo จะใช้ประเมินการชนระหว่างอนุภาคในพลาสมาและอนุภาคในอากาศ (ซึ่งจำเป็นในการจำลองพลาสมาภายใต้ความดันบรรยากาศ) การถ่ายทอดโมเมนตัม ความเร็ว และทิศทางของอนุภาค รวมถึงปฏิกิริยาทางเคมีที่เกิดขึ้นในการชนแต่ละครั้งจะถูกบันทึกไว้ และนำไปรวมกับผลที่ได้จากการคำนวณในส่วนของ PIC

**พลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศ**

ปัจจุบันต้นกำเนิดพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศ (Low-Temperature Atmospheric-Pressure Plasma Source) ถูกนำมาใช้ประโยชน์ในด้านต่างๆ อย่างแพร่หลาย เช่น การบำบัดพื้นผิว (Surface Treatment) การสร้างฟิล์มบาง (Thin-Film Deposition) และการฆ่าเชื้อโรค (Sterilization) ซึ่งสาเหตุหลักที่เทคโนโลยีชนิดนี้ได้รับความนิยมเป็นอย่างมากสืบเนื่องมาจากความสามารถในการสร้างพลาสมาที่ระดับความดันบรรยากาศปกติ ทำให้ไม่มีความจำเป็นต้องใช้ระบบสุญญากาศและระบบอื่นๆ ที่ต้องใช้ในสุญญากาศอีกต่อไป จึงมีข้อได้เปรียบหลายอย่างเมื่อเทียบกับต้นกำเนิดพลาสมาที่ความดันต่ำ เช่น สามารถปฏิบัติงานได้ง่ายขึ้น สามารถเพิ่มขนาดพื้นที่สัมผัสพลาสมาได้กว้างขึ้น และลดค่าใช้จ่ายของอุปกรณ์ที่เกี่ยวข้อง อย่างไรก็ดี การประยุกต์ใช้พลาสมาในงานด้านต่างๆ จะต้องเลือกพลาสมาที่มีคุณลักษณะที่เหมาะสมเพื่อให้สามารถเกิดปฏิกิริยาตามที่ต้องการได้ ซึ่งพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศจะมีคุณลักษณะเฉพาะตัวที่แตกต่างจากพลาสมาชนิดอื่นๆ

**คุณลักษณะของพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศ**

ต้นกำเนิดพลาสมาโดยทั่วไปอาศัยสนามแม่เหล็กไฟฟ้าเป็นตัวกลางในการถ่ายเทพลังงานให้กับอิเล็กตรอน เพื่อให้มีพลังงานเพียงพอที่จะก่อให้เกิดอันตรกิริยาไอออไนเซชัน (Ionization) กับอนุภาคก๊าซและแตกตัวเป็นไอออนบวกขึ้น ซึ่งไอออนที่เกิดขึ้นก็จะได้รับพลังงานจากสนามแม่เหล็กไฟฟ้าด้วยเช่นกัน สำหรับพลาสมาอุณหภูมิต่ำ (Low-Temperature หรือ Cold Plasma) เป็นพลาสมาที่อุณหภูมิหรือพลังงานของอิเล็กตรอนและไอออนไม่อยู่ในสภาวะที่สมดุล (Non-Equilibrium) สืบเนื่องจากอิเล็กตรอนซึ่งมีมวลน้อยกว่าไอออนได้รับพลังงานจากสนามแม่เหล็กไฟฟ้ามากกว่า จึงทำให้มีอุณหภูมิหรือพลังงานสูงกว่าไอออน

การชน (collision) ระหว่างอิเล็กตรอนกับอนุภาคอื่นๆ จะช่วยลดพลังงานของอิเล็กตรอนลงได้ ซึ่งสำหรับต้นกำเนิดพลาสมาที่อยู่ภายใต้ความดันก๊าซสูง อนุภาคก๊าซก็จะมีความหนาแน่นสูงด้วย ส่งผลให้ความน่าจะเป็นที่จะเกิดการชนกันระหว่างอนุภาคต่างๆ เพิ่มขึ้น และทำให้พลาสมาเข้าสู่สภาวะที่สมดุล (Equilibrium) ได้ แต่สำหรับต้นกำเนิดพลาสมาที่อยู่ในสภาวะความดันก๊าซต่ำ จะมีอัตราการชนที่น้อยมาก จึงทำให้พลังงานของอิเล็กตรอนอยู่ในระดับที่สูงกว่าพลังงานของไอออนและอนุภาคอื่นๆ ในพลาสมา ส่วนการชนก็ไม่ส่งผลให้พฤติกรรมของพลาสมาเปลี่ยนแปลงมากนัก

พลาสมาความดันต่ำที่ความดันบรรยากาศ เป็นพลาสมาที่อยู่ในสภาวะไม่สมดุล แต่มีความหนาแน่นของก๊าซสูงเพียงพอที่อาจส่งผลต่อพฤติกรรมของพลาสมาอันเนื่องมาจากการชนได้ โดยทั่วไปปริมาณการไอออไนเซชันของก๊าซจะเป็นแบบบางส่วน (Partial Ionization) ทำให้ภายในพลาสมาประกอบด้วยอนุภาคอิเล็กตรอนที่มีประจุลบ อนุภาคไอออนที่มีประจุบวก และอนุภาคก๊าซที่ไม่มีประจุ โดยความหนาแน่นของอนุภาคก๊าซที่ไม่มีประจุอาจสูงกว่าอนุภาคมีประจุประมาณ 1000 เท่าหรือมากกว่านั้น[29] ดังนั้นอนุภาคที่

**การเคลื่อนที่ของอนุภาคมีประจุภายใต้สนามไฟฟ้าและสนามแม่เหล็ก**

พลาสมาประกอบด้วยอนุภาคหลายชนิดเคลื่อนที่ด้วยความเร็วที่แตกต่างกันอยู่ภายใน เช่น อิเล็กตรอน ไอออนที่เกิดจากการแตกตัวของก๊าซ และอนุภาคของก๊าซเอง โดยอนุภาคอาจมีประจุลบ ประจุบวก หรือไม่มีประจุ ซึ่งสำหรับอนุภาคที่มีประจุเมื่ออยู่ภายใต้สนามไฟฟ้า ($\vec{E}$, V/m) และสนามแม่เหล็ก ($\vec{B}$, T) จะได้รับอิทธิพลจากแรงที่เกิดจากทั้งสองสนาม ทำให้เกิดการเคลื่อนที่ที่สามารถอธิบายได้ด้วย Lorentz Force Equation

$$\vec{F} = m\frac{d\vec{v}}{dt} = q\left(\vec{E} + \vec{v}\times\vec{B}\right) \quad (1)$$

โดย $m$ และ $q$ คือ มวลและประจุของอนุภาค มีหน่วย kg และ C ตามลำดับ

$\vec{v}$ คือ ความเร็วของอนุภาค มีหน่วย m/s

ตำแหน่งของอนุภาค ($\vec{r}$, m) สามารถคำนวณได้จากสมการ

$$\vec{r} = \int \vec{v}dt \quad (2)$$

**การเปลี่ยนแปลงของสนามไฟฟ้าและสนามแม่เหล็กในพลาสมา**

ความหนาแน่นของอนุภาคมีประจุในพลาสมา มีผลต่อสนามไฟฟ้าและสนามแม่เหล็ก โดยสามารถอธิบายได้ด้วย Maxwell's Equations ในสุญญากาศ (3a-3d)

$$\vec{\nabla}\cdot\vec{E} = \frac{\rho}{\varepsilon_0} \quad (3a)$$

$$\vec{\nabla}\cdot\vec{B} = 0 \quad (3b)$$

$$\vec{\nabla}\times\vec{E} = -\frac{d\vec{B}}{dt} \quad (3c)$$

$$\vec{\nabla}\times\vec{B} = \mu_0\varepsilon_0\frac{d\vec{E}}{dt} + \mu_0\vec{J} \quad (3d)$$

โดย $\varepsilon_0$ คือ ค่าสภาพยอมผ่านสนามไฟฟ้าในสุญญากาศ มีค่า 8.854 x $10^{-12}$ F/m

$\mu_0$ คือ ค่าสภาพยอมผ่านสนามแม่เหล็กในสุญญากาศ มีค่า 1.257 x $10^{-6}$ H/m

และ $\rho$ คือ ความหนาแน่นของประจุไฟฟ้า มีหน่วย C/$m^3$

$\vec{J}$ คือ ความหนาแน่นของกระแสไฟฟ้า มีหน่วยเป็น A/$m^2$

ซึ่งสามารถคำนวณได้จากสมการ

$$\rho = \sum_i(q_i n_i) \quad (4)$$

$$\vec{J} = \sum_i(q_i\langle n_i\vec{v}_i\rangle) \quad (5)$$

โดย $n$ คือ ความหนาแน่นของอนุภาคชนิด $i$ มีหน่วย $m^{-3}$

$\vec{v}_i$ คือ ความเร็วของอนุภาคชนิด $i$

สำหรับพลาสมาที่มีฟังค์ชันการกระจายตัวของอนุภาคตามความเร็ว $f(\vec{v})$ จะมี

$$n = \int f(\vec{v})d\vec{v} \quad (6)$$

$$\langle n\vec{v}\rangle = \int \vec{v}f(\vec{v})d\vec{v} \quad (7)$$

เมื่อ $d\vec{v} \equiv dv_x dv_y dv_z$

ศักย์ไฟฟ้า ($\Phi$, V) สามารถคำนวณได้จากสมการ

$$\vec{\mathrm{E}} = -\vec{\nabla}\Phi \quad (8)$$

ซึ่งเมื่อแทนค่าในสมการ (3a) จะได้ Poisson's Equation

$$\nabla^2\Phi = -\frac{\rho}{\varepsilon_0} \quad (9)$$

**การชนภายในพลาสมา**

อนุภาคภายในพลาสมาเคลื่อนที่อยู่ตลอดเวลาด้วยความเร็ว $\vec{v}$ จึงทำให้มีโอกาสที่จะชนกันและเกิดผลสืบเนื่องดังต่อไปนี้ได้

- ถ่ายเทโมเมนตัม ทำให้พลังงานจลน์ของอนุภาคที่เกี่ยวข้องเปลี่ยนไป
- เปลี่ยนทิศทางการเคลื่อนที่
- สร้างอนุภาคตัวใหม่ขึ้น
- รวมตัวกัน

โดยการชนสามารถแบ่งได้เป็นหลายประเภท เช่น Elastic Collision, Excitation, De-excitation, Ionization, Attachment และ Recombination ซึ่งให้ผลสืบเนื่องที่แตกต่างกัน

ประเภทของการชนในพลาสมาจะขึ้นอยู่กับปัจจัยหลายชนิด รวมถึง

- อนุภาคที่เกี่ยวข้องในการชน เช่น ระหว่างอิเล็กตรอนกับอิเล็กตรอน (e-e) ระหว่างไอออนกับไอออน (i-i) ระหว่างอิเล็กตรอนกับไอออน (e-i) ระหว่างอิเล็กตรอนกับอนุภาคไม่มีประจุ (e-n) และการชนระหว่างไอออนกับอนุภาคไม่มีประจุ (i-n)
- ความหนาแน่นของอนุภาคเป้าหมาย (อนุภาคที่จะโดนชน)
- ความเร็วสัมพัทธ์ระหว่างอนุภาคที่เกี่ยวข้อง
- พลังงานจลน์ของอนุภาคเข้าชน (อนุภาคที่จะวิ่งเข้าชน)
- ภาคตัดขวางของการชน (cross section)

ปริมาณของการชนแต่ละประเภทจะขึ้นอยู่กับความถี่ของการชน ($\nu$) สามารถคำนวณได้จาก

$$\nu = n\langle\sigma v\rangle \tag{10}$$

โดย $n$ คือ ความหนาแน่นของอนุภาคเป้าหมาย

$\sigma$ คือ ภาคตัดขวางของการชน มีหน่วย $m^{-2}$

$v$ คือ ขนาดความเร็วสัมพัทธ์ระหว่างอนุภาคเป้าหมายและอนุภาคเข้าชน

สำหรับพลาสมาที่มีฟังค์ชันการกระจายตัวของอนุภาคตามความเร็ว $f(v)$ จะมี

$$\langle\sigma v\rangle = \frac{\int \sigma v f(v) dv}{\int f(v) dv} \tag{11}$$

**ภาคตัดขวางของการชน**

ภาคตัดขวางของการชนแต่ละประเภทจะมีความแตกต่างกัน ซึ่งในการศึกษานี้จะให้ความสนใจกับการชนระหว่างอิเล็กตรอนและอนุภาคก๊าซที่ไม่มีประจุ (e-n) เนื่องจากเป็นการชนที่มีบทบาทสำคัญเพิ่มขึ้นเมื่อความดันก๊าซ (ความหนาแน่น) เพิ่มขึ้น โดยเฉพาะการชนแบบยืดหยุ่นหรือ electron elastic collision

การชนแบบยืดหยุ่นระหว่างอิเล็กตรอนและอนุภาคก๊าซเป็นการชนที่พลังงานจลน์รวมก่อนการชนเท่ากับพลังงานจลน์รวมหลังการชน โดยการชนจะไม่ทำให้เกิดการเปลี่ยนแปลงสภาพภายในของอนุภาคทั้งสองชนิดและไม่มีอนุภาคชนิดใหม่เกิดขึ้น แต่อาจทำให้ความเร็วและทิศทางของอนุภาคเปลี่ยนไปหลังการชน ภาคตัดขวางของการชนจะขึ้นอยู่กับ

- พลังงานของอิเล็กตรอน เช่น การชนระหว่างอิเล็กตรอนกับอนุภาคก๊าซอาร์กอน (Ar) จะมีภาคตัดขวางรวม (Total Cross Section) เท่ากับ 4.9246 x 10-20 และ 1.3783 x 10-20 m-2 สำหรับอิเล็กตรอนที่มีพลังงาน 100 และ 1000 eV ตามลำดับ[30]
- ชนิดของก๊าซ เช่น การชนระหว่างอิเล็กตรอนที่มีพลังงาน 1000 eV กับอนุภาคก๊าซอาร์กอน และระหว่างอิเล็กตรอนที่พลังงานเดียวกันกับอนุภาคก๊าซฮีเลียม (He) มีภาคตัดขวางรวมเท่ากับ 1.3783 x 10-20 และ 4.4375 x $10^{-22}$ $m^{-2}$ ตามลำดับ[30]

การเปลี่ยนแปลงความเร็วของอนุภาคระหว่างก่อนและหลังการชน สามารถคำนวณได้จากสมการอนุรักษ์โมเมนตัมและพลังงาน

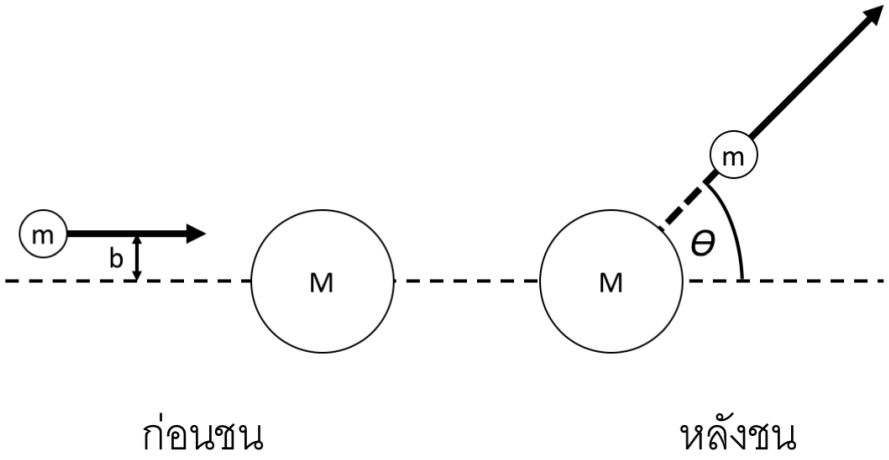
$$m\vec{v}_{ei} + M\vec{v}_{ni} = m\vec{v}_{ef} + M\vec{v}_{nf} \tag{12}$$

$$\frac{1}{2}mv_{ei}^2 + \frac{1}{2}Mv_{ni}^2 = \frac{1}{2}mv_{ef}^2 + \frac{1}{2}Mv_{nf}^2 \tag{13}$$

เมื่อ $m$, $M$, $v_{ei}$, $v_{ni}$, $v_{ef}$ และ $v_{nf}$ คือ มวลของอิเล็กตรอน มวลของอนุภาคก๊าซ ความเร็วก่อนชนของอิเล็กตรอน ความเร็วก่อนชนของอนุภาคก๊าซ ความเร็วหลังชนของอิเล็กตรอน และความเร็วหลังชนของอนุภาคก๊าซ ตามลำดับ ซึ่งเมื่อแก้สมการ (12) และ (13) จะให้ความเร็วหลังชนของอิเล็กตรอนและอนุภาคก๊าซใน Laboratory Frame ซึ่งกำหนดให้ความเร็วก่อนชนของอนุภาคก๊าซเป็นศูนย์ และก๊าซมีมวลสูงกว่าอิเล็กตรอนมาก จนพลังงานของอนุภาคก๊าซก่อนและหลังชนไม่เปลี่ยนแปลง (=0) ดังนั้น

$$v_{ei} = v_{ef} \tag{14}$$

และหากกำหนดให้ $\theta$ เป็นมุมของการกระเจิงหลังการชนในระนาบของการชน (ดังแสดงในรูปที่ 1) ซึ่งมีค่าระหว่าง 0 และ $\pi$ จะพบว่าค่าของ $\theta$ ขึ้นอยู่กับระยะ impact parameter (b) ซึ่งเมื่อ b = 0 อิเล็กตรอนจะวิ่งกลับในทิศทางตรงข้ามด้วยความเร็วเท่าเดิม และเมื่อ b มีค่าสูงขึ้น $\theta$ จะมีค่าลดลง




**รูปที่ 1** การชนแบบยืดหยุ่นซึ่ง M >> m

National Institute of Standards and Technology (NIST) ได้ทำการวัดและรวบรวมค่าภาคตัดขวางเชิงอนุพันธ์ ($d\sigma/d\Omega$) และภาคตัดขวางรวม ($\sigma_T$) ของการชนแบบยืดหยุ่นระหว่างอิเล็กตรอนกับอนุภาคก๊าซที่พลังงานต่างๆ ไว้[30] ซึ่งสำหรับการชนระหว่างอิเล็กตรอนกับก๊าซอาร์กอนที่พลังงาน 100 และ 1000 eV จะมีภาคตัดขวางเชิงอนุพันธ์ที่ $\theta$ ต่างๆ ดังรูปที่ 2 และ 3 ตามลำดับ เมื่อ $a_0^2$ = 2.8002852 x $10^{-21}$ $m^2$

สเตอเรเดียน ($\Omega$) มุมกระเจิง ($\theta$) และมุมรอบแกน ($\varphi$) มีความสัมพันธ์ดังสมการ

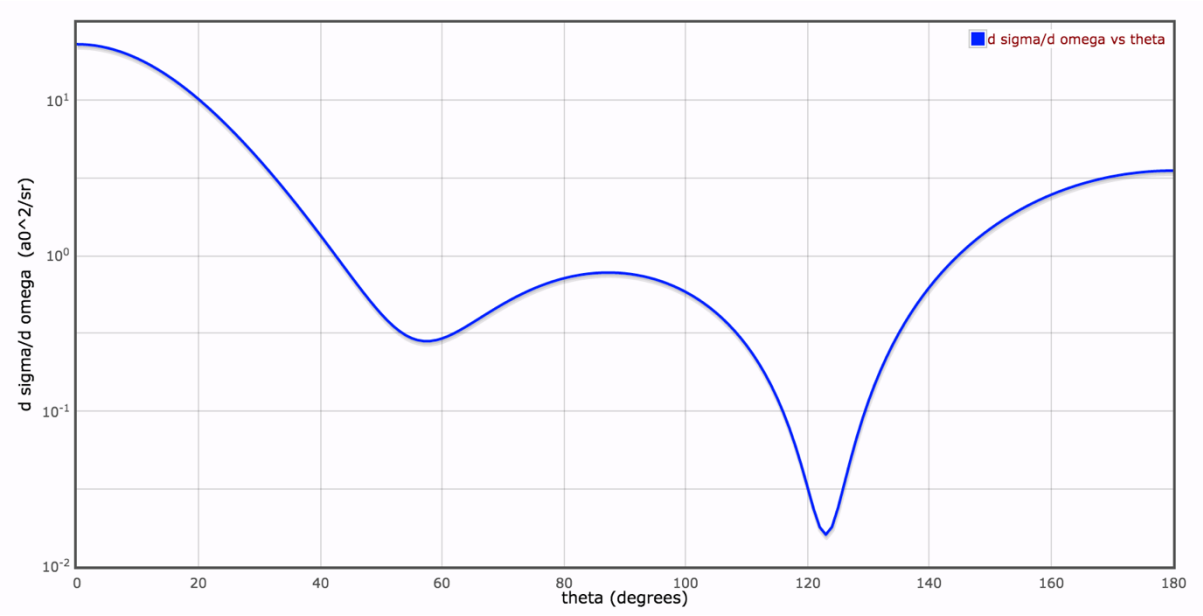
$$d\Omega = sin\theta d\theta d\varphi \tag{15}$$

และสามารถคำนวณภาคตัดขวางรวมได้จาก

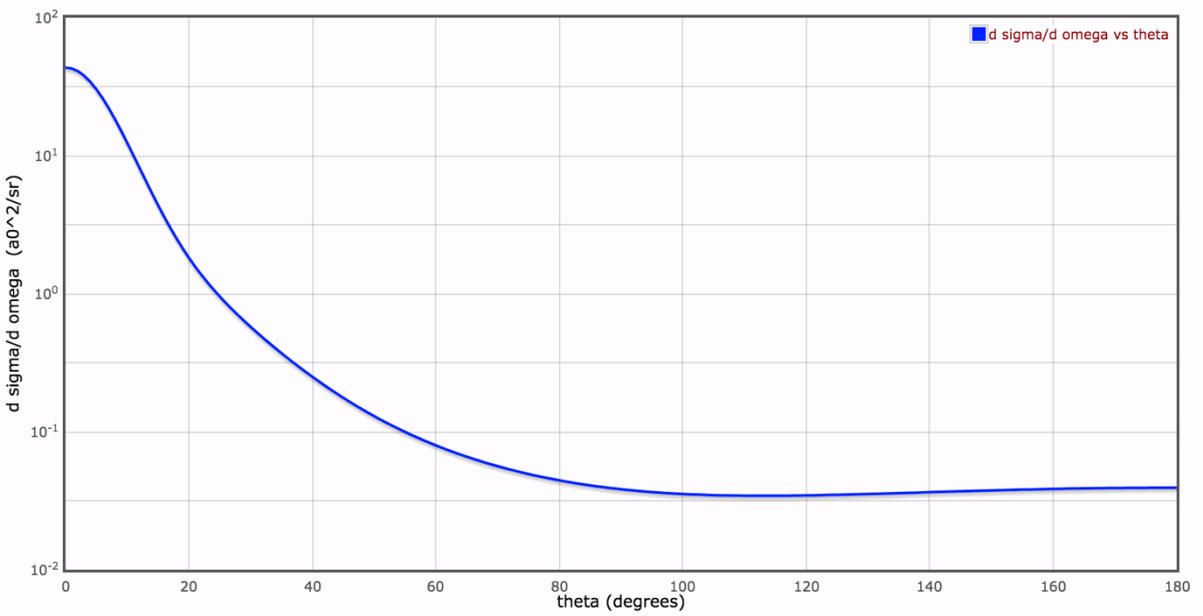
$$\sigma_T = \int \frac{d\sigma}{d\Omega} d\Omega = \int_0^{2\pi} \int_0^{\pi} \frac{d\sigma}{d\Omega} sin\theta d\theta \, d\varphi \tag{16}$$

ค่างของ $\sigma_T$ จะขึ้นอยู่กับพลังงานของอิเล็กตรอนที่วิ่งเข้าชน และชนิดของอนุภาคก๊าซเป้าหมาย รูปที่ *4* แสดง $\sigma_T$ สำหรับก๊าซอาร์กอนและฮีเลียม

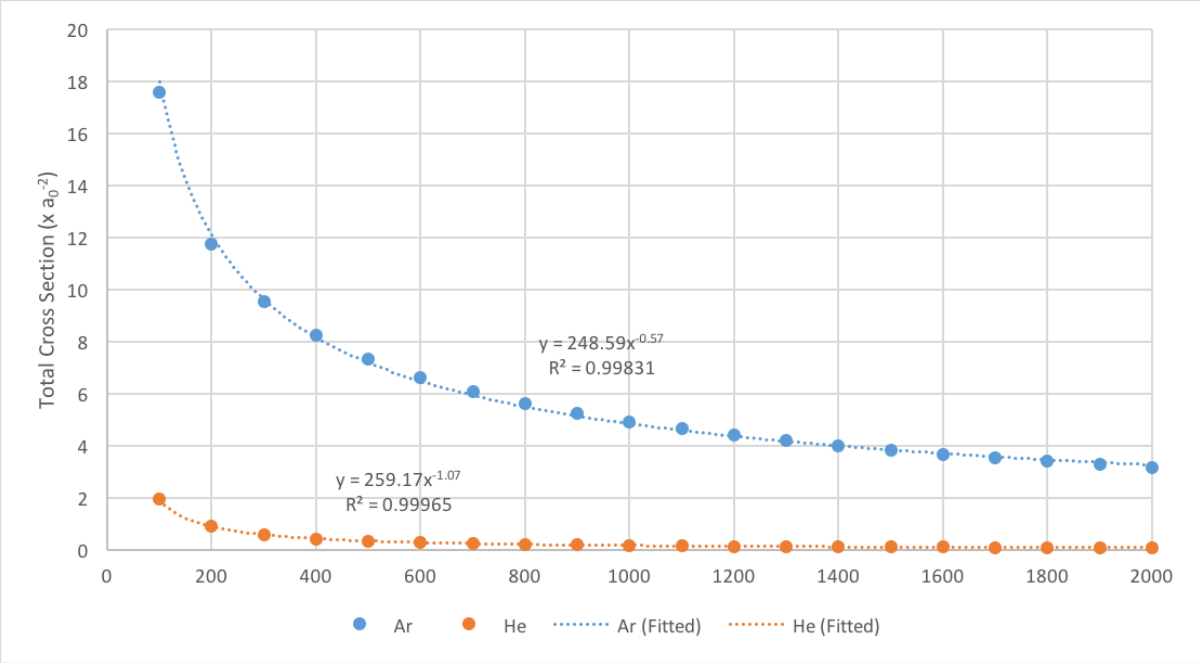
เนื่องจากค่าของ $d\sigma/d\Omega$ จะสูงสุดในช่วงแรก และลดลงอย่างรวดเร็วเมื่อ $\theta$ มากขึ้น ชี้ให้เห็นว่า $\sigma$ ไม่มีการเปลี่ยนแปลงมากนัก




**รูปที่ 2** ภาคตัดขวางเชิงอนุพันธ์ของการชนระหว่างอิเล็กตรอนพลังงาน 100 eV และ Ar (ภาคตัดขวางรวม = $4.9246 \times 10^{-20}$ $m^{-2}$) [30]




**รูปที่ 3** ภาคตัดขวางเชิงอนุพันธ์ของการชนระหว่างอิเล็กตรอนพลังงาน 1000 eV และ Ar (ภาคตัดขวางรวม = $1.3783 \times 10^{-20}$ $m^{-2}$) [30]




**รูปที่ 4** ค่าภาคตัดขวางรวมของการชนแบบยืดหยุ่นระหว่างอิเล็กตรอนกับอนุภาคของก๊าซอาร์กอนและฮีเลียม ที่พลังงานต่างๆ ($a_0^2$ = 2.8002852 x $10^{-21}$ $m^2$)

**Debye Length**

พลาสมาประกอบด้วยอนุภาคมีประจุซึ่งมีคุณสมบัติในการกำบังสนามไฟฟ้า ส่งผลให้ศักย์ไฟฟ้า ณ จุดใดๆ ได้รับอิทธิพลจากประจุที่ซึ่งอยู่ห่างจากจุดนั้นๆ ไม่เกินระยะที่เรียกว่า Debye length ($\lambda_D$) เท่านั้น ซึ่งสำหรับ Non-Equilibrium Plasma ที่มี $T_e >> T_i$

$$\lambda_D = \left(\frac{\varepsilon_0 k_B T_e}{e^2 n_e}\right)^{1/2} \tag{17}$$

เมื่อ $k_B$ = Boltzmann Constant = 8.6173303(50)×$10^{-5}$ eV/K

**การอนุรักษ์พลังงานภายในระบบ**

พลังงานโดยรวมของระบบปิดจะต้องคงที่

$$\frac{d}{dt}(\text{Total Energy}) = 0 \tag{18}$$

ซึ่งสำหรับระบบที่ประกอบด้วยสนามแม่เหล็ก สนามไฟฟ้า และอนุภาคมีประจุ ในสุญญากาศ พลังงานงานที่เกี่ยวข้องประกอบด้วย

- พลังงานศักย์ในสนามไฟฟ้า สามารถคำนวณได้จาก

$$U_E = \frac{1}{2}\varepsilon_0 \int_V \left|\vec{E}\right|^2 dV \tag{19}$$

- พลังงานศักย์ในสนามแม่เหล็ก สามารถคำนวณได้จาก

$$U_B = \frac{1}{2\mu_0} \int_V \left|\vec{B}\right|^2 dV \tag{20}$$

- พลังงานจลน์ของอนุภาค สามารถคำนวณได้จาก

$$KE = \sum_i \frac{1}{2} m_i |\vec{v}_i|^2 \tag{21}$$

- พลังงานอื่นๆ เช่น พลังงานความร้อน พลังงานการหมุน หรือ พลังงานที่เกิดขึ้นหรือสูญเสียไปอันเนื่องมาจากอันตรกิริยาที่เกิดขึ้นระหว่างอนุภาค เป็นต้น

**การอนุรักษ์โมเมนตัมในระบบ**

เช่นเดียวกับพลังงาน โมเมนตัมของระบบปิดจะต้องคงที่ การเปลี่ยนแปลงของโมเมนตัมเป็นศูนย์

$$\frac{d\vec{P}}{dt} = 0 \tag{22}$$

ซึ่งจากสมการ *Lorentz Force* สามารถคำนวณโมเมนตัมรวมของระบบได้จาก

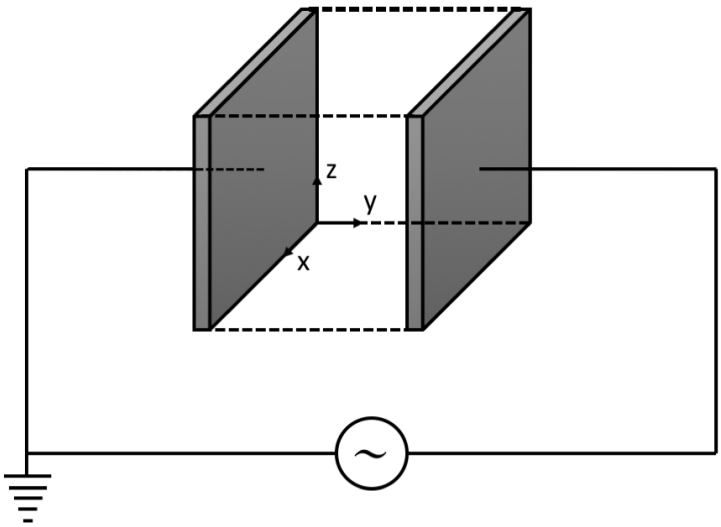
$$\frac{d\vec{P}}{dt} = \sum_i m_i \frac{d\vec{v}_i}{dt} = \int_V \rho\vec{E}\, dV \tag{23}$$

# บทที่ 3 โปรแกรมการจำลอง

โปรแกรมการจำลองพัฒนาขึ้นด้วยภาษา Java ภายใต้ Java Environment Version 8

**แบบจำลอง**

วงจรที่ใช้ในการจำลองประกอบด้วยขั้วไฟฟ้า 2 ขั้วซึ่งด้านหนึ่งต่อเข้ากับ Ground และอีกด้านหนึ่งต่อเข้ากับแหล่งจ่ายแรงดันไฟฟ้าแบบ AC ดังรูปที่ 5 การคำนวณต่างๆ จะจำกัดอยู่ในบริเวณที่สนใจ (Region of Interest, ROI) ซึ่งเป็นส่วนที่อยู่ระหว่างขั้วไฟฟ้าทั้งสอง โดยในบทนี้จะอธิบายโดยใช้สมการการคำนวณแบบ 3 มิติใน Cartesian Coordinate เป็นหลัก




**รูปที่ 5** วงจรของแบบจำลอง

อย่างไรก็ดีผู้วิจัยได้ทำการพัฒนาโปรแกรมทั้งแบบ 1 และ 2 มิติเพื่อใช้ศึกษาปรากฏการณ์ที่ไม่จำเป็นต้องใช้การคำนวณแบบ 3 มิติไว้ด้วย และชุดสมการต่างๆ ยังสามารถปรับเปลี่ยนสำหรับการคำนวณใน Cylindrical Coordinate และ Spherical Coordinate ได้เช่นกัน

ภายใน ROI สามารถกำหนดค่าเริ่มต้นต่างๆ ได้ดังนี้

- ศักย์ไฟฟ้า ($\Phi$) โดยใช้
    - Dirichlet Boundary Condition เช่น บนระนาบ $y=y_{min}$ และ $y=y_{max}$ กำหนดให้

$$\Phi(x, y_{min}, z) = a \text{ และ } \Phi(x, y_{max}, z) = b$$

- Neumann Boundary Condition เช่น บนระนาบ $x=x_{min}$, $x=x_{max}$, $z=z_{min}$ และ $z=z_{max}$ กำหนดให้ $\Phi_x(x_{min},y,z)=a$, $\Phi_x(x_{max},y,z)=b$, $\Phi_z(x,y,z_{min})=c$ และ $\Phi_z(x,y,z_{max})=d$

หรือ

- Periodic Boundary Condition เช่น บนระนาบ $y=y_{min}$, $y=y_{max}$, $x=x_{min}$, $x=x_{max}$, $z=z_{min}$ และ $z=z_{max}$ กำหนดให้ $\Phi(x_{min},y,z)=\Phi(x_{max},y,z)$, $\Phi(x,y_{min},z)=\Phi(x,y_{max},z)$, $\Phi(x,y,z_{min})=\Phi(x,y,z_{max})$, $\Phi_x(x_{min},y,z)=\Phi_x(x_{max},y,z)$, $\Phi_y(x,y_{min},z)=\Phi_y(x,y_{max},z)$ และ $\Phi_z(x,y,z_{min})=\Phi_z(x,y,z_{max})$

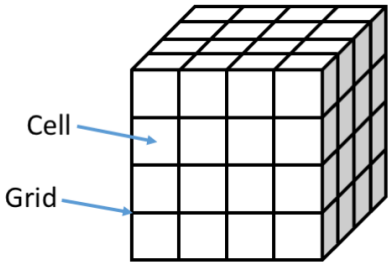
- สนามไฟฟ้า ($\vec{B}$) โดยกำหนดให้ $\vec{B}(x,y,z)=B_x\hat{x}+B_y\hat{y}+B_z\hat{z}$ หรือสนามไฟฟ้าคงที่ใน ROI
- ชนิดและปริมาณของอนุภาคต่างๆ เช่น อิเล็กตรอน ไอออนของก๊าซ และอนุภาคของก๊าซ ความหนาแน่นของอนุภาคแต่ละชนิด ความเฉลี่ยความเร็วเริ่มต้นและค่าความเบี่ยงเบนของอนุภาคแต่ละชนิด (สำหรับ Normal Distribution)
- อันตรกิริยาระหว่างอนุภาคต่างๆ เช่น ชนิดของอันตรกิริยา อนุภาคที่เกี่ยวข้อง (อนุภาคเข้าชนและอนุภาคเป้าหมาย ภาคตัดขวางของอันตรกิริยา

**การคำนวณ**

การคำนวณใช้เทคนิคหลัก 2 ชนิด คือ Particle-in-Cell (PIC) และ Monte Carlo (MC)

Particle-in-Cell

ใช้สำหรับการคำนวณผลของสนามไฟฟ้าและสนามแม่เหล็กที่มีต่อการเคลื่อนที่ของอนุภาคมีประจุต่างๆ และผลของความหน่าแน่นของอนุภาคมีประจุต่างๆ ที่มีต่อสนามไฟฟ้า โดยเริ่มด้วยการกำหนดจุดการคำนวณ (Grid) และหน่วยการคำนวณ (Cell) ภายใน ROI ออกดังรูปที่ 6




**รูปที่ 6** Grid และ Cell

Grid(i, j, k) แสดงถึง Grid ซึ่งอยู่ในตำแหน่ง i, j และ k ในแนวแกน x, y และ z ตามลำดับ โดยกำหนดให i, j และ k เป็นจำนวนเต็มตั้งแต่ 0 ถึงจำนวนของ Grid ในแต่ละแนว

ในการนี้ ได้กำหนดให้อนุภาคที่อยู่ใน Cell รอบๆ Grid เท่านั้นที่มีผลต่อ Grid ดังนั้น Cell จึงควรมีขนาดใกล้เคียงกับ Debye Length

$$\Delta x, \Delta y, \Delta z \approx \lambda_D$$

ซึ่งหากความเร็วของอิเล็กตรอนมีการกระจายตัวแบบ Maxwellian Distribution ที่มีค่าเฉลี่ยและค่าเบี่ยงเบนความเร็ว $u$ และ $\sigma$ ตามลำดับ

$$f(v) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(v-u)^2}{2\sigma^2}}$$

โดยที่ $\sigma = (k_B T_e / m_e)^{1/2}$ ดังนั้น

$$\lambda_D = \left(\frac{\varepsilon_0 m_e \sigma^2}{e^2 n_e}\right)^{1/2}$$

ศักย์ไฟฟ้า ($\Phi$) ที่ Grid ต่างๆ สามารถคำนวณได้จากสมการ (9) ด้วยวิธีคำนวณวนรอบแบบ Gauss-Seidel โดยสำหรับ Grid ที่อยู่ภายใน ROI สามารถเขียนสมการ (9) ในรูปของ

$$\frac{\Phi_{i-1,j,k} - 2\Phi_{i,j,k} + \Phi_{i+1,j,k}}{(\Delta x)^2} + \frac{\Phi_{i,j-1,k} - 2\Phi_{i,j,k} + \Phi_{i,j+1,k}}{(\Delta y)^2}$$

$$+ \frac{\Phi_{i,j,k-1} - 2\Phi_{i,j,k} + \Phi_{i,j,k+1}}{(\Delta z)^2} = -\left(\sum_l \frac{q_l n_l}{\varepsilon_0}\right)_{i,j,k} = -f_{i,j,k}$$

ซึ่งสามารถจัดรูปต่อเพื่อคำนวณศักย์ไฟฟ้าที่ Grid i, j, k โดยใช้ค่าจาก Grid ข้างเคียง ดังนี้

$$\Phi_{i,j,k} = \frac{1}{2[(\Delta x)^2(\Delta z)^2+(\Delta y)^2(\Delta z)^2+(\Delta x)^2(\Delta y)^2]}$$
$$\times\Big[f_{i,j,k}(\Delta x)^2(\Delta y)^2(\Delta z)^2$$
$$+\big(\Phi_{i-1,j,k}+\Phi_{i+1,j,k}\big)(\Delta y)^2(\Delta z)^2$$
$$+\big(\Phi_{i,j-1,k}+\Phi_{i,j+1,k}\big)(\Delta x)^2(\Delta z)^2$$
$$+\big(\Phi_{i,j,k-1}+\Phi_{i,j,k+1}\big)(\Delta x)^2(\Delta y)^2\Big]$$

วิธีคำนวณวนรอบแบบ Gauss-Seidel ใช้ค่าจากการคำนวณในรอบก่อนหน้า (m) ในการคำนวณรอบปัจจุบัน (m+1)

$$\Phi_{i,j,k}^{m+1} = \frac{1}{2[(\Delta x)^2(\Delta z)^2+(\Delta y)^2(\Delta z)^2+(\Delta x)^2(\Delta y)^2]}$$
$$\times\Big[f_{i,j,k}(\Delta x)^2(\Delta y)^2(\Delta z)^2$$
$$+\big(\Phi_{i-1,j,k}^{m+1}+\Phi_{i+1,j,k}^{m}\big)(\Delta y)^2(\Delta z)^2$$
$$+\big(\Phi_{i,j-1,k}^{m+1}+\Phi_{i,j+1,k}^{m}\big)(\Delta x)^2(\Delta z)^2$$
$$+\big(\Phi_{i,j,k-1}^{m+1}+\Phi_{i,j,k+1}^{m}\big)(\Delta x)^2(\Delta y)^2\Big]$$

โดยในการคำนวณแต่ละรอบจะให้ค่าคงเหลือ (Residual, r) ที่แต่ละ Grid

$$r_{i,j,k}^{m+1} = \frac{\phi_{i-1,j,k}^{m+1}-2\phi_{i,j,k}^{m}+\phi_{i+1,j,k}^{m}}{(\Delta x)^2} + \frac{\phi_{i,j-1,k}^{m+1}-2\phi_{i,j,k}^{m}+\phi_{i,j+1,k}^{m}}{(\Delta y)^2}$$
$$+\frac{\phi_{i,j,k-1}^{m+1}-2\phi_{i,j,k}^{m}+\phi_{i,j,k+1}^{m}}{(\Delta z)^2} + f_{i,j,k}$$

ซึ่งสามารถคำนวณค่าคงเหลือรวมด้วยสมการ

$$r_{total}^{m+1} = \sqrt{\sum_{i,j,k}\big(r_{i,j,k}^{m+1}\big)^2}$$

การวนรอบจะดำเนินไปจนกว่า

$$r_{total}^{m+1} \le tol \quad \text{และ} \quad \left|\frac{r_{total}^{m+1}-r_{total}^{m}}{r_{total}^{m}}\right| \le dtol$$

โดย *tol* และ *dtol* คือ ค่าคงเหลือสะสมที่ยอมรับได้ และ อัตราการเปลี่ยนแปลงค่าคงเหลือสะสมที่ยอมรับได้ตามลำดับ ซึ่งควรมีค่าเข้าใกล้ศูนย์

เพื่อให้ $\boldsymbol{\Phi}$ จากการคำนวณในแต่ละรอบมีค่าเข้าใกล้ค่าที่ถูกต้องรวดเร็วขึ้น ได้ใช้วิธี Over Relaxation ด้วย Relaxation Constant (w) ในการปรับค่า $\boldsymbol{\Phi}$ จากการคำนวณในแต่ละรอบ ดังสมการ

$$\left[\Phi_{i,j,k}^{m+1}\right]_{new} = (1-w)\Phi_{i,j,k}^{m} + w\Phi_{i,j,k}^{m+1}$$

สำหรับ Grid ที่อยู่รอบ ROI ค่าศักย์ไฟฟ้าที่ใช้ในการคำนวณจะเป็นไปตาม Boundary Condition ต่างๆ เช่น

- Grid ที่อยู่บนระนาบ $y=y_{min}$ มี $\boldsymbol{\Phi_{i,j,k}} = 0$ (Dirichlet)
- Grid ที่อยู่บนระนาบ $y=y_{max}$ มี $\boldsymbol{\Phi_{i,j,k}} =$ ค่าที่กำหนดด้วยแหล่งจ่ายแรงดันไฟฟ้า (Dirichlet)
- Grid ที่อยู่บนระนาบ $x=x_{min}$ และ $x=x_{max}$ มี $\boldsymbol{\Phi_{i-1,j,k}} = \boldsymbol{\Phi_{i+1,j,k}}$ ซึ่งทำให้ $\boldsymbol{\Phi_x(x,y,z)} = \left(\Phi_{i+1,j,k} - \Phi_{i-1,j,k}\right)/(2\Delta x) = 0$ (Neumann)
- Grid ที่อยู่บนระนาบ $z=z_{min}$ และ $z=z_{max}$ มี $\boldsymbol{\Phi_{i,j,k-1}} = \boldsymbol{\Phi_{i,j,k+1}}$ ซึ่งทำให้ $\boldsymbol{\Phi_z(x,y,z)} = \left(\Phi_{i,j,k+1} - \Phi_{i,j,k-1}\right)/(2\Delta z) = 0$ (Neumann)

สนามไฟฟ้าที่แต่ละ Grid คำนวณได้จากสมการ (8)

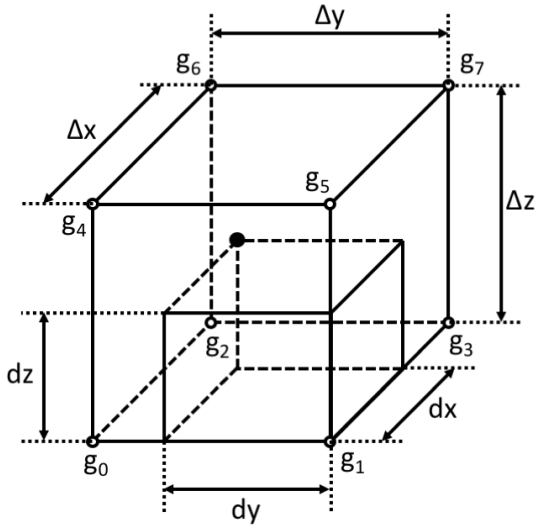
$$\vec{E}_{i,j,k} = E_{x|i,j,k}\hat{x} + E_{y|i,j,k}\hat{y} + E_{z|i,j,k}\hat{z} \quad \text{เมื่อ}$$

$$E_{x|i,j,k} = -\frac{\Phi_{i+1,j,k}-\Phi_{i-1,j,k}}{2\Delta x}$$

$$E_{y|i,j,k} = -\frac{\Phi_{i,j+1,k}-\Phi_{i,j-1,k}}{2\Delta y}$$

$$E_{z|i,j,k} = -\frac{\Phi_{i,j,k+1}-\Phi_{i,j,k-1}}{2\Delta z}$$

ความหนาแน่น ($n_l$) ที่ใช้ในการคำนวณศักย์และสนามไฟฟ้าเป็นค่า ณ ตำแหน่งของ Grid ซึ่งคำนวณจากปริมาณอนุภาคที่กระจายตัวอยู่ใน Cell ที่ล้อมรอบ Grid นั้น หากพิจารณาอนุภาค 1 ตัวใน Cell ที่ล้อมรอบด้วย Grid $g_0$ – $g_7$ ดังรูปที่ 7 แต่ละ Grid จะได้รับอิทธิพลจากอนุภาคนั้นในสัดส่วนที่ไม่เท่ากัน มิฉะนั้นจะเป็นการฝ่าฝืนกฎการอนุรักษ์มวล (Conservation for Mass)




**รูปที่ 7** อนุภาคใน Cell ที่ล้อมรอบด้วย Grid

สัดส่วนของอนุภาคบน Grid ใดๆ จะขึ้นอยู่กับระยะห่างระหว่างอนุภาคกับ Grid นั้น ซึ่ง Grid ที่อยู่ห่างมากกว่าจะได้รับสัดส่วนที่น้อยกว่า โดยสามารถคำนวณสัดส่วนของอนุภาคบนแต่ละ Grid ดังตัวอย่างในรูปที่ 7

สัดส่วนของอนุภาคบน Grid $g_6$ = $\frac{dxdydz}{\Delta x\Delta y\Delta z}$

และผลรวมของสัดส่วนของอนุภาคหนึ่งๆ ที่กระจายไปบน Grid $g_0$ – $g_7$ จะต้องเท่ากับ 1 ดังนั้น

$$n_{l|i,j,k} = (\text{ผลรวมของสัดส่วนของอนุภาคใน Cell รอบๆ Grid i, j, k})/(\Delta x\Delta y\Delta z)$$

ความเร็วของอนุภาคมีประจุแต่ละตัวจะเปลี่ยนแปลงภายใต้สนามไฟฟ้าและสนามแม่เหล็กตามสมการ (1) ซึ่งจะทำให้อนุภาคเกิดการเคลื่อนที่ โดยสามารถคำนวณค่าความเร็วและตำแหน่งใหม่ของอนุภาคด้วย Leapfrog Method ร่วมกับ Boris Scheme ดังนี้

$$\vec{r}_{t+\Delta t} = \vec{r}_t + \Delta t\vec{v}_{t+\Delta t/2} \quad ; \vec{r} = \langle x, y, z\rangle$$

$$\vec{v}_{t+\Delta t/2} = \vec{u}^{+} + \Delta t\frac{q}{2m}\vec{E}_t$$

โดย $$\vec{u}^{+} = \vec{u}^{-} + \left(\vec{u}^{-} + \left(\vec{u}^{-}\times\vec{t}\right)\right)\times\vec{s}$$

$$\vec{u}^{-} = \vec{v}_{t-\Delta t/2} + \Delta t\frac{q}{2m}\vec{E}_t$$

$$\vec{t} = \Delta t \frac{q}{2m} \vec{B}_t$$

$$\vec{s} = \frac{2\vec{t}}{\left(1+|\vec{t}|^2\right)}$$

ที่เวลาเริ่มต้น $\vec{v}_{-\Delta t/2}$ คำนวณได้จากสมการ

$$\frac{\vec{v}_0 - \vec{v}_{-\Delta t/2}}{\Delta t/2} = \frac{q}{m}\left(\vec{E}_0 + \vec{v}_0 \times \vec{B}_0\right)$$

$$\vec{v}_{-\Delta t/2} = \vec{v}_0 - \Delta t \frac{q}{2m}\left(\vec{E}_0 + \vec{v}_0 \times \vec{B}_0\right)$$

ค่าของ $\vec{E}$ และ $\vec{B}$ ที่ใช้ในการคำนวณข้างต้น เป็นค่าที่ตำแหน่งของอนุภาค แต่ค่าที่คำนวณได้จาก Poisson Equation เป็นค่าที่ตำแหน่งของ Grid ดังนั้นจึงจำเป็นต้องทำการประเมินค่า ณ ตำแหน่งของอนุภาค โดยใช้ค่าจาก Grid ที่อยู่ล้อมรอบ Cell ที่อนุภาคนั้นอยู่ในการ Interpolate ซึ่งสำหรับการศึกษานี้ จะใช้ Linear Interpolation ด้วย B-Spline Function

$$b(\xi) = \left|\frac{\xi - \xi_p}{\Delta\xi}\right| \quad \text{เมื่อ } \xi \text{ คือ } x, y \text{ หรือ } z$$

อิเล็กตรอนซึ่งมีความสามารถในการเคลื่อนที่ได้สูงกว่าอนุภาคอื่นๆ ในพลาสมา จะมีการเปลี่ยนแปลงที่เป็นไปตามความถี่ของพลาสมา (Plasma Frequency, $\omega_{pe}$) ซึ่งสามารถคำนวณได้จากความหนาแน่นของอิเล็กตรอน

$$\omega_{pe} = \sqrt{\frac{e^2 n_e}{\varepsilon_0 m_e}}$$

ดังนั้น เพื่อให้สามารถติดตามการเปลี่ยนแปลงนี้ได้ จึงต้องกำหนด $\Delta t$ ให้มีค่าน้อยกว่าคาบของการเปลี่ยนแปลงนี้ โดยทั่วไปจะกำหนดให้

$$\Delta t < \frac{2}{\omega_{pe}}$$

Monte Carlo

ใช้สำหรับจำลองการชนแบบยืดหยุ่นระหว่างอิเล็กตรอนกับอนุภาคก๊าซไม่มีประจุ โดยคำนวณความน่าจะเป็นที่อิเล็กตรอน "$i$" จะเกิดการชน ($p_i$) จาก

$$p_i = 1 - e^{-v_i n_t \sigma \Delta t}$$

เมื่อ $v_i$ คือ ความเร็วสัมพัทธ์ระหว่างอิเล็กตรอนกับอนุภาคก๊าซ (โดยทั่วไปความเร็วของอนุภาคก๊าซจะน้อยกว่าความเร็วของอิเล็กตรอนมาก จึงใช้ความเร็วของอิเล็กตรอนแทน $v_i$ ได้)

$n_t$ คือ ความหนาแน่นของอนุภาคเป้าหมาย (อนุภาคก๊าซ) ซึ่งค่อนข้างจะคงที่

$\sigma$ คือ ภาคตัดขวางของการชนซึ่งขึ้นอยู่กับพลังงานของอิเล็กตรอน

$\Delta t$ คือ เวลาการชน (Collision Time) ซึ่งในที่นี้คือ Time Step ของการคำนวณ

การชนจะถือว่าเกิดขึ้นเมื่อ

$$R_1 \leq p_i$$

โดย $R_1$ เป็นค่าสุ่มแบบสม่ำเสมอที่มีค่าระหว่าง [0, 1]

เมื่อเกิดการชนขึ้น ความเร็ว (ขนาดและทิศทาง) ของอนุภาคจะเปลี่ยนแปลงตามกฎการอนุรักษ์โมเมนตัม (12) และกฎการอนุรักษ์พลังงาน (13) ซึ่งสำหรับการชนในกรณีนี้ที่มวลของอนุภาคก๊าซมากกว่ามวลของอิเล็กตรอนมาก จะถือได้ว่าขนาดความเร็วของทั้งอนุภาคก๊าซและอิเล็กตรอนจะคงที่ แต่ทิศทางความเร็วของอิเล็กตรอนจะเปลี่ยนไป ซึ่งแต่ละทิศทางจะมีความน่าจะเป็นไม่เท่ากัน

อย่างไรก็ตามเนื่องจากอนุพันธ์ภาคตัดขวางของการชนมีลักษณะดังรูปที่ 3 โดยเฉพาะเมื่อพลังงานของอิเล็กตรอนสูงขึ้น จึงสามารถประมาณได้ว่าภาคตัดขวางการชนในแต่ละทิศทางมีค่าใกล้เคียงกับภาคตัดขวางรวม และสามารถคำนวณความเร็วใหม่ของอิเล็กตรอนหลักชนได้เป็น

$$\vec{u} = \langle v_i sin\theta cos\phi, v_i sin\theta sin\phi, v_i cos\theta \rangle$$

$$\theta = \pi R_2$$

$$\phi = 2\pi R_3$$

โดย $R_2$ และ $R_3$ เป็นค่าสุ่มแบบสม่ำเสมอที่มีค่าระหว่าง [0, 1]

สำหรับเทคนิคนี้ สามารถใช้ได้กับการชนรูปแบบอื่นๆ เช่นกัน โดยโปรแกรมที่พัฒนาขึ้นมีลักษณะเป็น Module ซึ่งสามารถเพิ่มการชนชนิดอื่นๆ ได้ตามต้องการ

การจัดการกับอนุภาค

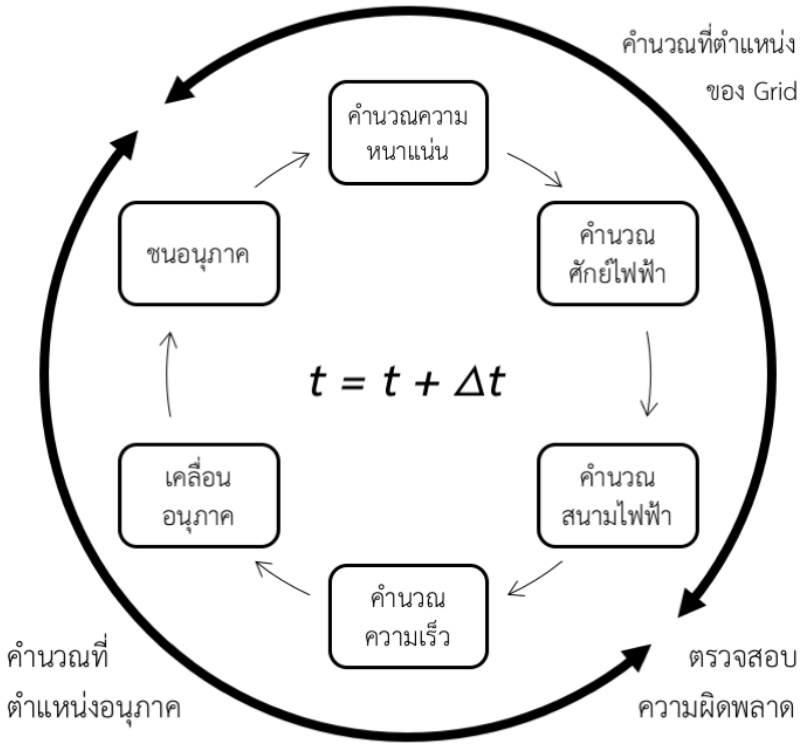
อนุภาคที่เคลื่อนที่ออกนอกขอบเขตของระบบ สามารถจัดการได้ 2 วิธี

- ให้ถือว่าสูญเสีย ในที่นี้จำนวนของอนุภาคในระบบจะลดลง และพลังงานจะไม่ถูกอนุรักษ์
- ให้นำกลับเข้าระบบจากฝั่งตรงข้ามด้วยความเร็วเท่าเดิม ในที่นี้จำนวนของอนุภาคในระบบจะเท่าเดิม และพลังงานจะถูกอนุรักษ์ (วิธี Default ของโปรแกรมนี้)

ขั้นตอนการคำนวณ

รูปที่ 8 แสดงวงรอบของการคำนวณที่ใช้ในแต่ละ Time Step โดยในแต่ละวงรอบจะทำการเก็บข้อมูลเกี่ยวกับตำแหน่ง ศักย์ไฟฟ้า สนามไฟฟ้าของ Grid และตำแหน่ง ความเร็ว พลังงาน และระยะทางการเคลื่อนที่รวมของอนุภาคไว้ในรูปของ Text File เพื่อใช้ในการวิเคราะห์และแสดงผลภายหลัง การคำนวณจะหยุดลงเมื่อครบตามจำนวนรอบที่ตั้งไว้

นอกจากนี้ ในแต่ละรอบจะทำการบันทึกค่าต่างๆ ที่เกี่ยวข้องกับประสิทธิภาพการคำนวณ และทำการประเมินความผิดพลาดที่เกิดขึ้น โดยเช็คกับฟิสิกส์ต่างๆ เช่น พลังงานรวม จำนวนอนุภาครวม เป็นต้น




**รูปที่ 8** วงรอบการคำนวณ

**Normalization**

แม้ว่าคอมพิวเตอร์ในปัจจุบันมีความเร็วเพิ่มขึ้นมากกว่าในยุคแรกๆ แต่การจำลองทางคอมพิวเตอร์ก็ยังมีข้อจำกัดในเรื่องของหน่วยความจำและความเร็วในการคำนวณเมื่อเปรียบเทียบกับจำนวนอนุภาคที่แท้จริงและขั้นตอนการคำนวณในแต่ละขั้นตอน จึงได้ทำการ Normalize ปริมาณและสมการต่างๆ ที่เกี่ยวข้อง เพื่อลดการใช้หน่วยความจำและเพิ่มความเร็วในการคำนวณ ดังนี้

| **ปริมาณ** | **ตัวแปร** | Normalization Factor | |
|---|---|---|---|
| | | **ปริมาณ** | **ค่า** |
| สภาพยอมผ่านสนามไฟฟ้า | $\varepsilon$ | สภาพยอมผ่านสนามไฟฟ้าในสุญญากาศ $\varepsilon_0$ | 8.854E-12 F/m |
| สภาพยอมผ่านสนามแม่เหล็ก | $\mu$ | สภาพยอมผ่านสนามแม่เหล็กในสุญญากาศ $\mu_0$ | 1.257E-6 H/m |
| มวล | $m$ | มวลของอิเล็กตรอน $m_e$ | 9.109E-31 kg |
| ประจุ | $q$ | ประจุของอิเล็กตรอน $e$ | 1.602E-19 C |
| ความเร็ว | $v$ | ความเร็วแสง $c$ | 2.998E-31 m/s |
| ความหนาแน่น | $n$ | $n_0$ | กำหนดขึ้นโดยผู้ใช้ |
| ความถี่ของพลาสมา | $\omega_p$ | $\sqrt{\frac{e^2 n_0}{\varepsilon_0 m_e}}$ | |
| ความยาว | $l$ | $\frac{c}{\omega_p}$ | |
| เวลา | $t$ | $\frac{1}{\omega_p}$ | |
| ศักย์ไฟฟ้า | $\Phi$ | $\frac{m_e c^2}{e}$ | |
| สนามไฟฟ้า | $E$ | $\frac{m_e \omega_p c}{q_e}$ | |

| ปริมาณ | ตัวแปร | Normalization Factor | |
|---|---|---|---|
| | | ปริมาณ | ค่า |
| สนามแม่เหล็ก | $B$ | $\frac{m_e \omega_p}{q_e}$ | |
| พลังงาน | $KE, PE$ | $\frac{m_e c^5 n_0}{\omega_p^3}$ | |

**โครงสร้างของโปรแกรม**

โปรแกรมที่พัฒนาขึ้นมีโครงสร้างที่สามารถแบ่งออกเป็นชุดคำนวณต่างๆ ดังนี้

- Configuration Module สำหรับกำหนดค่าตั้งต้นในการคำนวณ ได้แก่
  - ค่าคงที่ต่างๆ รวมถึง ค่าสภาพยอมผ่านสนามไฟฟ้าและสนามแม่เหล็กในสุญญากาศ เป็นต้น
  - ชื่อแฟ้มข้อมูลและชื่อไฟล์สำหรับเก็บข้อมูล
  - ค่าคงที่สำหรับการคำนวณ รวมถึง ค่าคงเหลือ (Residual) ที่ยอมรับได้ อัตราการเปลี่ยนแปลงค่าคงเหลือ (Residual Rate) ที่ยอมรับได้ ค่าคงที่การผ่อนคลาย (Relaxation Constant) ก้าวเวลา (Time Step) และรอบการคำนวณ เป็นต้น
  - พื้นที่การคำนวณ รวมถึง ระยะการคำนวณในแนว x, y และ z จำนวนจุดการคำนวณ (Grid) ในแนว x, y และ z ขนาดของศักย์ไฟฟ้า และสนามแม่เหล็ก
  - ข้อมูลของอนุภาคในระบบ รวมถึง ชนิด จำนวน ค่าความถ่วงจำนวน (เนื่องจากข้อจำกัดของคอมพิวเตอร์ อนุภาค 1 ตัวจึงต้องเป็นตัวแทนของอนุภาคชนิดเดียวกันตามค่าความถ่วงจำนวน) ความเร็วเฉลี่ยของอนุภาค และค่าการกระจายตัวของความเร็ว
  - ข้อมูลอันตรกิริยา รวมถึง อนุภาคเข้าชน อนุภาคเป้าหมาย ชนิดของอันตรกิริยา ค่าภาคตัดขวาง
  - เรียกค่าตั้งต้นต่างๆ ใช้งาน
- Grid Module สำหรับ
  - สร้างและตั้งค่าเริ่มต้นของ Grid

- เก็บข้อมูลเกี่ยวกับ Grid ได้แก่ ค่าระบุตัวตน (id) ตำแหน่ง ศักย์ไฟฟ้า สนามไฟฟ้า และสนามแม่เหล็กของแต่ละ Grid
- คำนวณความหนาแน่นที่แต่ละ Grid
- คำนวณศักย์ไฟฟ้าที่แต่ละ Grid
- คำนวณสนามไฟฟ้าที่แต่ละ Grid
- เก็บข้อมูลของ Grid เข้าไฟล์
- เรียกข้อมูลต่างๆ ของแต่ละ Grid ใช้งาน

- Cell Module สำหรับ
  - สร้างและตั้งค่าเริ่มต้นของ Cell
  - เก็บข้อมูลเกี่ยวกับ Cell ได้แก่ ค่าระบุตัวตน (id) และค่าระบุตัวตนของ Grid ที่อยู่ล้อมรอบ
  - เรียกข้อมูลต่างๆ ของแต่ละ Cell ใช้งาน
- Particle Module สำหรับ
  - สร้างและตั้งค่าเริ่มต้นของอนุภาค
  - เก็บข้อมูลเกี่ยวกับอนุภาคได้แก่ ค่าระบุตัวตน (id) ชนิด ตำแหน่ง ความเร็ว ระยะทางที่เคลื่อนที่รวม พลังงานจลน์ และค่าระบุตัวตนของ Cell ที่อนุภาคอยู่
  - หาตำแหน่งของ Cell ที่อนุภาคอยู่
  - คำนวณความเร็วและตำแหน่งใหม่ของอนุภาคที่เคลื่อนที่ภายใต้สนามไฟฟ้าและสนามแม่เหล็กด้วยเทคนิค Particle-in-Cell
  - คำนวณความเร็วใหม่ของอนุภาคหลังเกิดอันตรกิริยาด้วยเทคนิค Monte Carlo
  - เก็บข้อมูลของอนุภาคเข้าไฟล์
  - เรียกข้อมูลต่างๆ ของแต่ละอนุภาคใช้งาน
- Interaction Module สำหรับ
  - สร้างและตั้งค่าเริ่มต้นของอันตรกิริยา
  - เก็บข้อมูลเกี่ยวกับอันตรกิริยา ได้แก่ ชนิดของอันตรกิริยา ภาคตัดขวาง อนุภาคเข้าชน และอนุภาคเป้าหมาย

- เรียกข้อมูลจากฐานข้อมูลอันตรกิริยาใช้งาน
- เรียกข้อมูลต่างๆ ของแต่ละอันตรกิริยาใช้งาน

- Database Module สำหรับ
  - เก็บข้อมูลต่างๆ เกี่ยวกับอนุภาค เช่น ชื่อ มวล ประจุ และเลขอะตอม
  - เก็บข้อมูลต่างๆ เกี่ยวกับภาคตัดขวาง เช่น ชื่อ ค่าภาคตัดขวาง
  - เก็บข้อมูลต่างๆ เกี่ยวกับอัตรกิริยา เช่น ชื่อ
- Graphic Module สำหรับแสดงผลการจำลองแบบ

รายละเอียดและ Source Code ของโปรแกรมได้ถูกรายงานไว้ในภาคผนวก ก

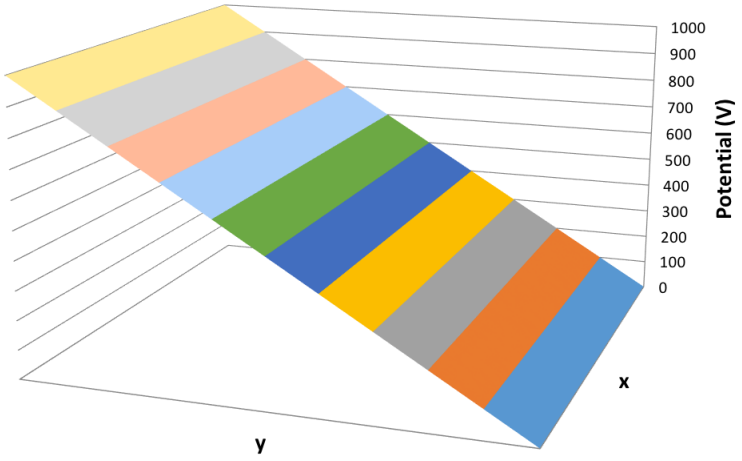
# บทที่ 4 ผลการทดสอบโปรแกรมและเครื่องกำเนิดพลาสมาแบบ DBD

## 4.1 การทดสอบการคำนวณศักย์ไฟฟ้า

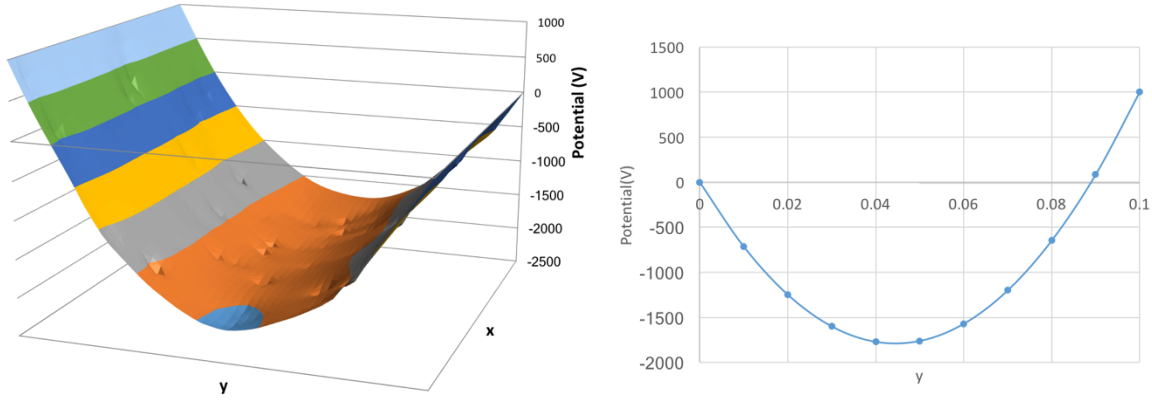
ได้ทำการประเมินความถูกต้องของโปรแกรมในการคำนวณศักย์ไฟฟ้า ได้ทำการจำลองด้วยเงื่อนไขดังต่อไปนี้

- $\Phi(y_{min}) = 0, \Phi(y_{max}) = 1000\ V$, ไม่มีอนุภาคในระบบ ซึ่งได้ผลถูกต้องตามที่แสดงในรูปที่ 9
- $\Phi(y_{min}) = 0, \Phi(y_{max}) = 1000\ V$, อิเล็กตรอนความหนาแน่น $10^{14}$ $m^{-3}$ ซึ่งได้ผลดังแสดงในรูปที่ 10 ซึ่งพบว่ามีค่าใกล้เคียงค่าประมาณด้วยการแก้ Poisson Equation (9) โดยตรง
- $\Phi(y_{min}) = 0, \Phi(y_{max}) = 1000\ V$, โปรตอนความหนาแน่น $10^{14}$ $m^{-3}$ ซึ่งได้ผลดังแสดงในรูปที่ 11 ซึ่งพบว่ามีค่าใกล้เคียงค่าประมาณด้วยการแก้ Poisson Equation (9) โดยตรง

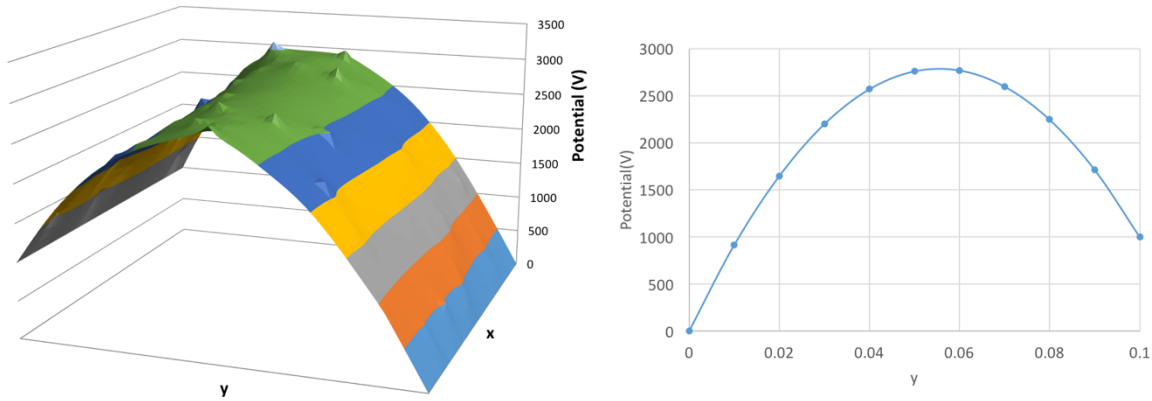
ทั้งนี้ได้กำหนดค่าความคงเหลือและอัตราการเปลี่ยนแปลงค่าคงเหลือที่ยอมรับได้ไว้ที่ $10^{-8}$ และ $10^{-4}$ ตามลำดับ




**รูปที่ 9** ศักย์ไฟฟ้าบนระนาบ $z=z_{max}/2$ เมื่อไม่มีอนุภาคในระบบ




**รูปที่ 10** ศักย์ไฟฟ้าบนระนาบ $z=z_{max}/2$ เมื่อมีอิเล็กตรอนความหนาแน่น $10^{14}$ $m^{-3}$ (ซ้าย) จากการจำลองแบบและ (ขวา) จากการแก้สมการใน 1-D




**รูปที่ 11** ศักย์ไฟฟ้าบนระนาบ $z=z_{max}/2$ เมื่อมีโปรตอนความหนาแน่น $10^{14}$ $m^{-3}$ (ซ้าย) จากการจำลองแบบและ (ขวา) จากการแก้สมการใน 1-D

### 4.2 การอนุรักษ์พลังงาน

ได้ทำการประเมินความถูกต้องของโปรแกรมด้วยการตรวจสอบการอนุรักษ์พลังงานในระบบปิด โดยใช้โปรแกรมแบบ 1D และตั้งค่าการจำลองดังนี้

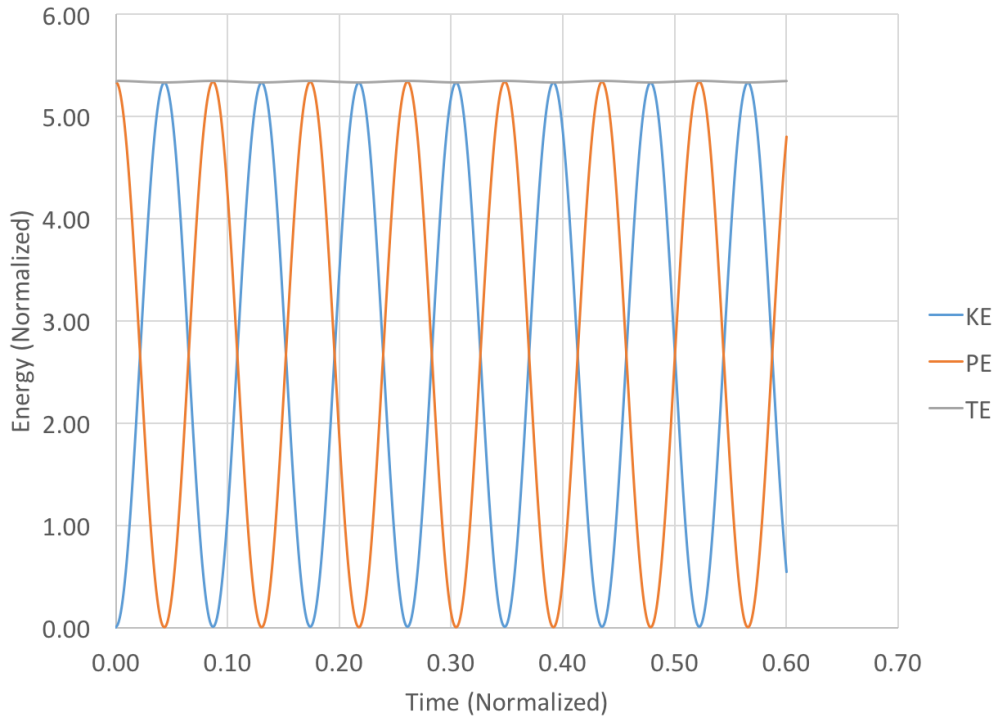
- ศักย์ไฟฟ้าเป็นศูนย์ทั้ง 2 ด้านและไม่มีสนามแม่เหล็ก
- ใส่อิเล็กตรอนและอาร์กอนมีประจุเข้าไปในระบบให้มีการกระจายตัวแบบ Uniform Distribution ซึ่งเบื้องต้นทั้งอิเล็กตรอนและอาร์กอนจะอยู่ในตำแหน่งเดียวกัน จึงทำให้ไม่มีความต่างศักย์ไฟฟ้าภายใน (สนามไฟฟ้า ณ ตำแหน่งต่างๆ เป็น 0)

- ทำการรบกวนตำแหน่งของอิเล็กตรอนเล็กน้อยให้มีความเปลี่ยนแปลงจากเดิมด้วยฟังค์ชัน $A\,sin(x)$ ซึ่งจะทำให้เกิดความต่างศักย์ภายในขึ้น

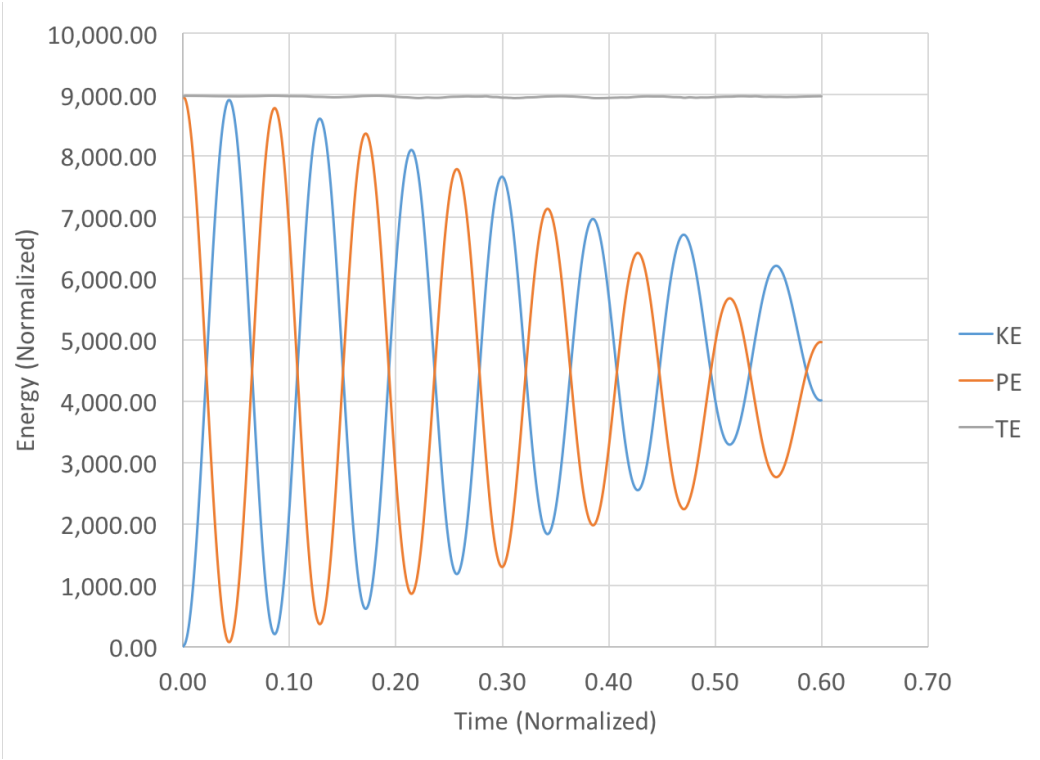
ผลการจำลองด้วย $dt = 0.002$ จำนวน 300 รอบ ด้วยจำนวน 513 Grid Points (512 ช่อง) ระหว่าง $x = [0,4\pi]$ ใช้ Periodic Boundary Condition และอนุภาคแต่ละชนิดจำนวน 16384 อนุภาค ได้ผลดังแสดงในรูปที่ 12 ซึ่งเป็นไปตามที่คาดการณ์ไว้ คือ

- พลังงานโดยรวมคงที่
- มีการแลกเปลี่ยนพลังงานระหว่างพลังงานศักย์ของสนามไฟฟ้าและพลังงานจลน์ของอนุภาค

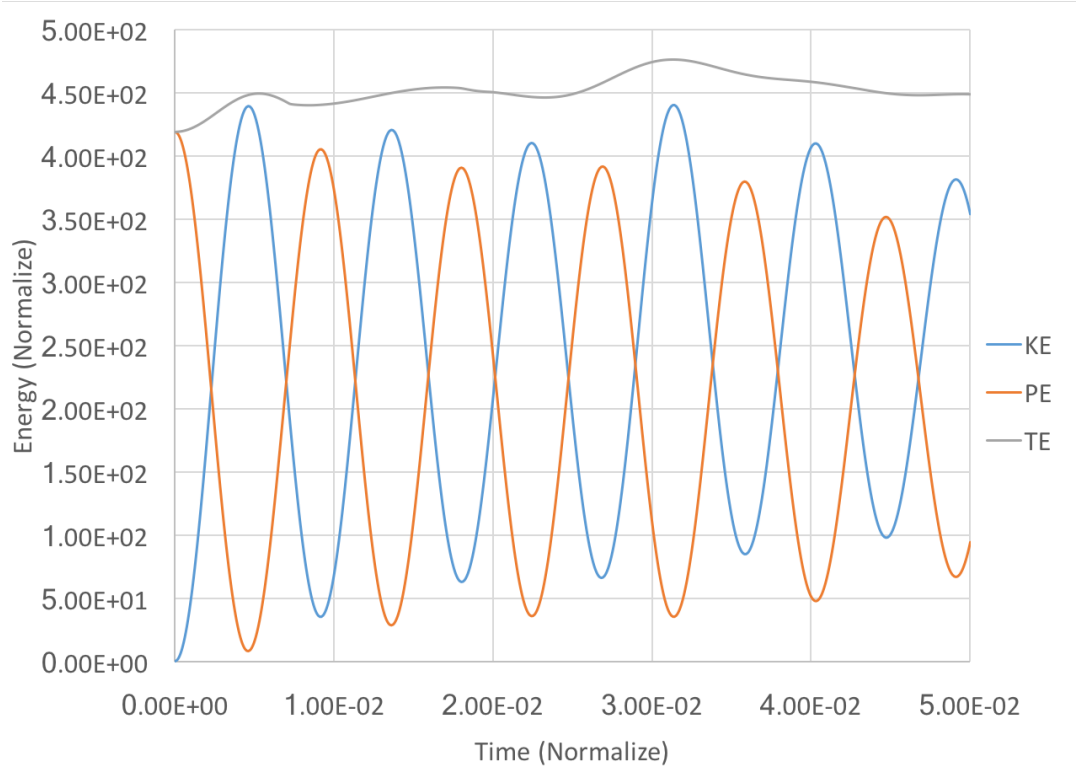
รูปที่ 13 แสดงการจำลองเมื่อให้อนุภาคมีการกระจายตัวแบบสุ่ม ซึ่งพลังงานโดยรวมยังคงที่เช่นเดิมแม้ว่าพลังงานจลน์และพลังงานศักย์จะเปลี่ยนไป




**รูปที่ 12** การถ่ายเทระหว่างพลังงานศักย์ของสนามไฟฟ้าและพลังงานจลน์ของอนุภาคในระบบปิด Uniform Distribution ใน 1D




**รูปที่ 13** การถ่ายเทระหว่างพลังงานศักย์ของสนามไฟฟ้าและพลังงานจลน์ของอนุภาคในระบบปิด Random Distribution ใน 1D

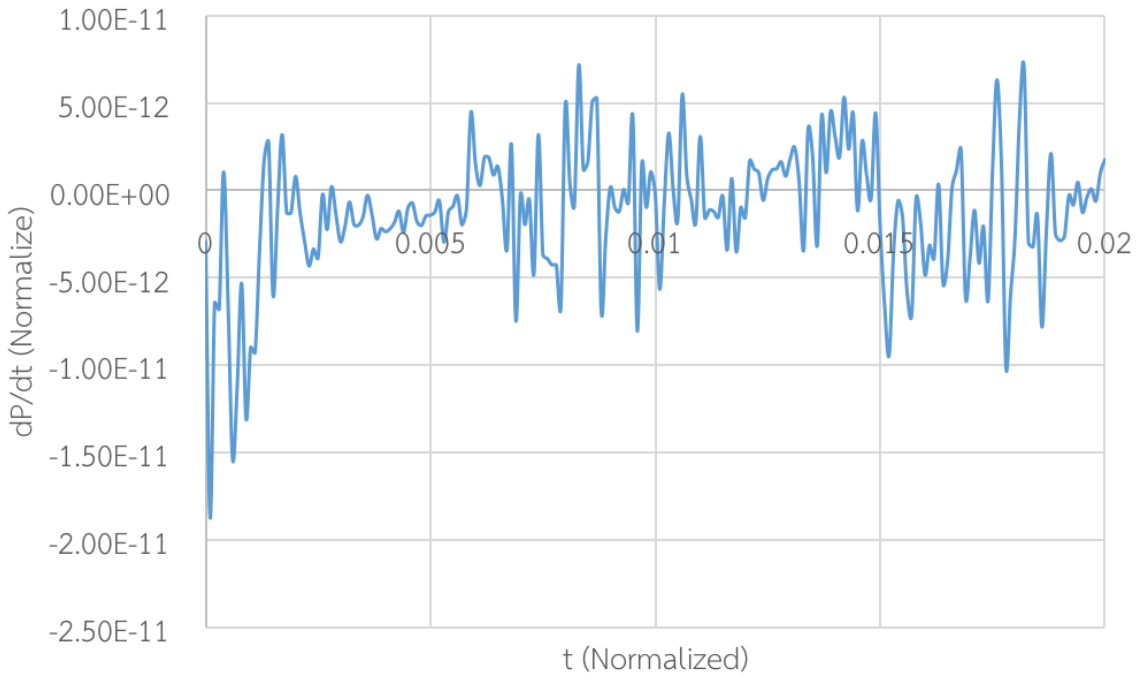



**รูปที่ 14** การถ่ายเทระหว่างพลังงานศักย์ของสนามไฟฟ้าและพลังงานจลน์ของอนุภาคในระบบปิด Uniform Distribution ใน 3D

ในการจำลองแบบโดยใช้โปรแกรม 3D พบว่าไม่สามารถอนุรักษ์พลังงานได้ดีนักเมื่อเทียบกับการคำนวณใน 1D ดังรูปที่ 14 ทั้งนี้ ปรากฏการณ์ดังกล่าวเป็นไปตามทฤษฎีที่คาดการณ์ไว้สำหรับโปรแกรมที่มีการคำนวณแบบอนุรักษ์โมเมนตัม[31] เนื่องจากการคำนวณสนามไฟฟ้าบน Grid ทำให้มีความผิดพลาดสะสมเกิดขึ้นอย่างรวดเร็วใน 3D ซึ่งแม้ว่าสามารถปรับเปลี่ยนการคำนวณให้เกิดการอนุรักษ์พลังงานได้โดยการคำนวณแรงที่กระทำกับอนุภาคจากศักย์ไฟฟ้าโดยตรง แต่วิธีนี้จะทำให้สูญเสียความสามารถในการอนุรักษ์โมเมนตัมได้แทน ซึ่งในทางปฏิบัติไม่ทำให้โปรแกรมมีความถูกต้องยิ่งขึ้น อย่างไรก็ดี การเปลี่ยนแปลงของพลังงานโดยรวมยังน้อยกว่า 10% สำหรับ 500 รอบการคำนวณซึ่งอยู่ในข่ายที่สามารถยอมรับได้ และจะเห็นว่าพลังงานยังมีการถ่ายเทไปมาระหว่างพลังงานศักย์ในสนามไฟฟ้าและพลังงานจลน์ของอนุภาคอยู่

### 4.3 การอนุรักษ์โมเมนตัม

ได้ทำการทดสอบการอนุรักษ์โมเมนตัมของระบบปิด โดยใช้โปรแกรม 3D พบว่ามีอัตราการเปลี่ยนแปลงของโมเมนตัมอยู่ในระดับ $10^{-12}$-$10^{-11}$ (Normalized Unit) ซึ่งใกล้เคียงค่า (ศูนย์) ที่คาดการณ์ไว้ ดังรูปที่ 15




**รูปที่ 15** การเปลี่ยนแปลงโมเมนตัมของระบบปิด Uniform Distribution ใน 3D

**4.4 ความเร็วในการคำนวณ**

จำนวนการคำนวณที่มีปริมาณสำคัญจะอยู่ในช่วงของการแก้ Poisson Equation ซึ่งจะขึ้นอยู่กับความเร็วในการ converge ของค่า และตัวแปร tol และ dtol ที่กำหนดขึ้น ซึ่งหากในเชิงเปรียบเทียบได้ทำการเปรียบเทียบความเร็วในการคำนวณกับความถูกต้องสำหรับโปรแกรมใน 3D โดยทำการคำนวณที่ 500 Time step จำนวนอนุภาค 12800x2 Uniform Distribution 21x21x21 grid points และใช้ tol และ dtol ซึ่งเป็นระดับค่าคงเหลือสะสมและอัตราการเปลี่ยนแปลงของค่าคงเหลือสะสมที่ยอมรับได้ดังนี้

| tol | dtol | เวลาการคำนวณ | พลังงานรวมที่ 500 รอบ |
|---|---|---|---|
| $10^{-8}$ | $10^{-4}$ | 5765 s | 448.403937568018 |
| $10^{-7}$ | $10^{-3}$ | 3951 s | 448.403937568019 |
| $10^{-6}$ | $10^{-2}$ | 3867 s | 448.403937568019 |
| $10^{-5}$ | $10^{-1}$ | 2721 s | 448.403937568114 |

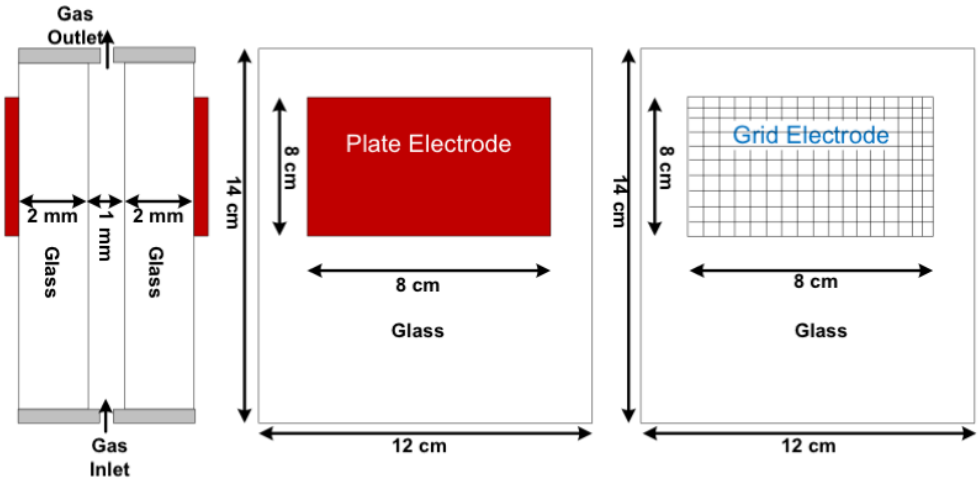
จากการทดสอบ พบว่าสามารถลดเวลาในการคำนวณได้อย่างน้อยครึ่งหนึ่งโดยการปรับค่าคงเหลือสะสมและอัตราการเปลี่ยนแปลงค่าคงเหลือสะสมที่ยอมรับได้เพิ่มขึ้น โดยยังสามารถคงความถูกต้องได้ถึงทศนิยม 9 ตำแหน่ง

**4.5 เครื่องกำเนิดพลาสมาชนิด DBD**

ในงานวิจัยนี้ ได้ทำการออกแบบและสร้างเครื่องกำเนิดพลาสมาแบบ DBD ขึ้น 2 ชนิดเพื่อทำการศึกษาพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศ หรือ LTAP

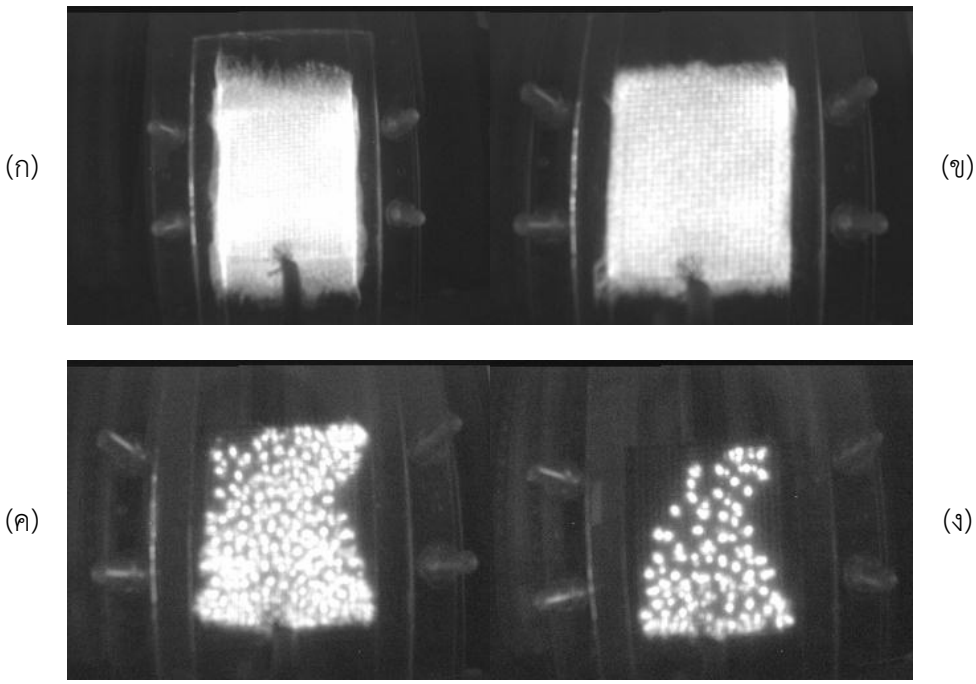
<u>เครื่องกำเนิดพลาสมาแบบ DBD ชนิด Parallel Plate</u>

มีรูปแบบดังแสดงในรูปที่ 16 โดยใช้แก้วบอโรซิลิเกตเป็นวัสดุไดอิเล็กทริค และแผ่นอลูมิเนียมเป็นขั้วไฟฟ้า ด้านหนึ่งของขั้วไฟฟ้ามีลักษณะเป็นตะแกรงเพื่อให้สามารถมองเห็นภายในได้ ในการทดลองใช้แหล่งกำเนิดไฟฟ้าขนาด 5-30 kVpp สามารถปรับความถี่ระหว่าง 50-1000 Hz ช่องว่างระหว่างไดอิเล็กทริคมีขนาด 1 มิลลิเมตรซึ่งได้ต่อเข้ากับแหล่งจ่ายก๊าซ




**รูปที่ 16** Parallel Plate DBD ที่ได้ออกแบบและสร้างขึ้น[32]

เนื่องจากเครื่องกำเนิดพลาสมานี้ถูกออกแบบเพื่อใช้ในการศึกษาการใช้พลาสมาในกระบวนการเอทานอลรีฟอร์มมิงเพื่อผลิตก๊าซไฮโดรเจน[32] จึงได้ทำการศึกษาการผลิตพลาสมาที่ระดับความต่างศักย์และความถี่ที่แตกต่างกัน โดยในการทดสอบเบื้องต้นได้ป้อนอากาศด้วยอัตราการไหล 2 มิลลิลิตรต่อนาที พบว่า พลาสมาที่เกิดขึ้นมีลักษณะแตกต่างกัน ตามความต่างศักย์และความถี่ที่ใช้ ดังรูปที่ 17

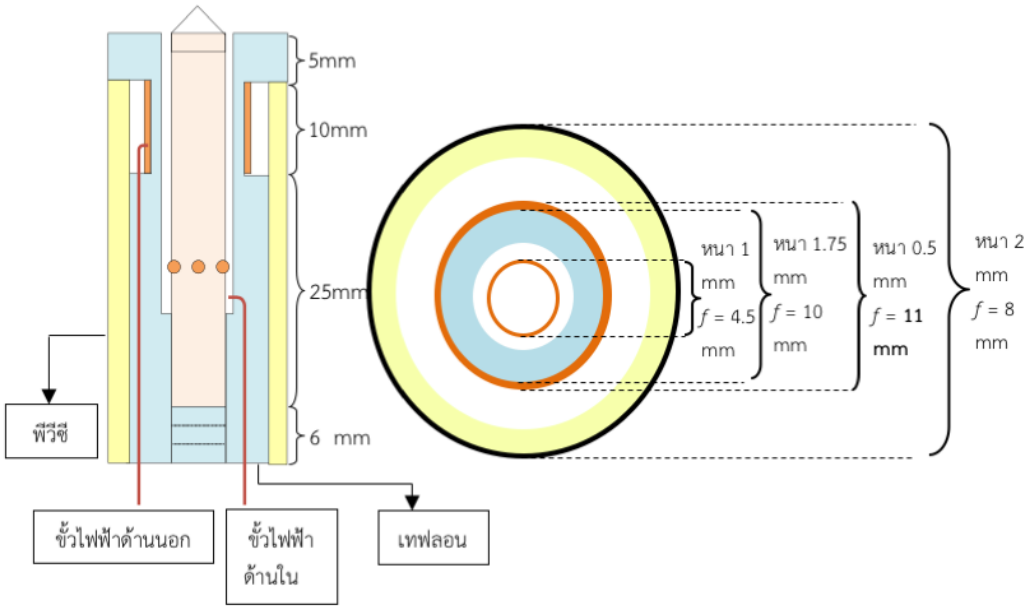



**รูปที่ 17** พลาสมาที่ความต่างศักย์ไฟฟ้าและความถี่ต่างๆ (ก) 200 Hz 30 kVpp (ข) 500 Hz 15 kVpp (ค) 800 Hz 6 kVpp และ (ง) 900 Hz 5kVpp[32]

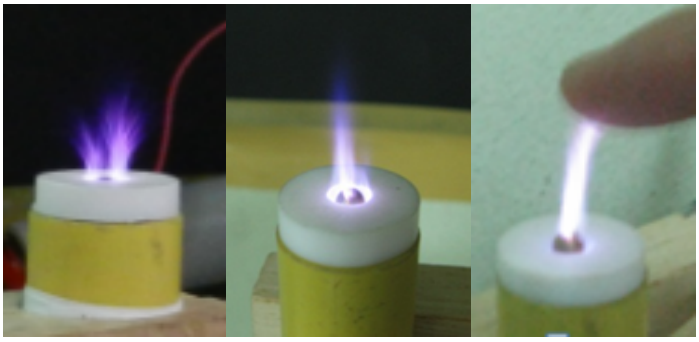
เครื่องกำเนิดพลาสมาแบบ DBD ชนิด Coaxial

มีรูปแบบดังแสดงในรูปที่ 18 โดยใช้เทฟลอนเป็นไดอิเล็กทริค ทองเหลืองเป็นขั้วฟ้าด้านใน และแผ่นทองแดงเป็นขั้วไฟฟ้าด้านนอก การทดลองใช้แหล่งกำเนิดไฟฟ้าขนาด 0-40 kVpp สามารถปรับความถี่ได้ระหว่าง 20-100 kHz ช่องว่างระหว่างไดเล็กทริคกับขั้วไฟฟ้ามีขนาด 1 มิลลิเมตรและต่อกับแหล่งจ่ายก๊าซ

เครื่องกำเนิดพลาสมาชนิดนี้ถูกออกแบบให้ผลิตและปล่อยพลาสมาออกมาเป็นเปลวเพื่อใช้ในการทดลองการฆ่าเชื้อโรค[33] ดังแสดงในรูปที่ 19 ซึ่งได้ทำการวัดความหนาแน่นของอิเล็กตรอนในพลาสมาโดยใช้วิธี Spectroscopy พบว่า เมื่อปล่อยก๊าซอาร์กอนเข้าไปในระบบด้วยอัตรา 8 ลิตรต่อนาทีและปรับความต่างศักย์ไฟฟ้าและความถี่ต่างๆ ในช่วง 16-28 kVpp และ 64-70 kHz จะได้พลาสมาที่มีความหนาแน่นอยู่ในช่วง $10^{24}$ $m^{-3}$




**รูปที่ 18** Coaxial DBD ที่ได้ออกแบบและสร้างขึ้น[33]



**รูปที่ 19** พลาสมาจาก Coaxial DBD Probe ที่สร้างขึ้น[33]

# บทที่ 5 สรุปงานวิจัย

งานวิจัยนี้แบ่งการดำเนินการออกเป็น 2 ส่วน คือ ส่วนการสร้างโปรแกรมสำหรับจำลองพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศ และ ส่วนการสร้างอุปกรณ์ผลิตพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศ โดยสามารถสรุปการดำเนินการ การศึกษา และการทดสอบในแต่ละส่วนได้ดังนี้

ส่วนโปรแกรมจำลองพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศ – ใช้วิธีการจำลองแบบ Particle-in-Cell โดยคำนวณการเปลี่ยนแปลงตำแหน่งและความเร็วของอนุภาคมีประจุภายใต้สนามแม่เหล็กและสนามไฟฟ้า คำนวณการเปลี่ยนแปลงของสนามไฟฟ้าจากความหนาแน่นของอนุภาคมีประจุ และจำลองอันตรกิริยาระหว่างอนุภาคด้วยวิธี Monte Carlo ซึ่งสามารถเพิ่มอันตรกิริยาชนิดต่างๆ ได้ตามต้องการ โดยหากเกิดอันตรกิริยาขึ้นจะทำการคำนวณความเร็วและทิศทางใหม่ของอนุภาคที่เกี่ยวข้อง

โปรแกรมที่สร้างขึ้นสามารถจำลองแบบพลาสมาใน 1D (infinite y-z space) และ 3D ใช้ขั้นตอนการคำนวณแบบอนุรักษ์โมเมนตัม ซึ่งจากการทดสอบพบว่า โปรแกรมสามารถคำนวณสนามไฟฟ้าด้วยการแก้ Poisson Equation ได้อย่างแม่นยำทั้งใน 1D และ 3D และสามารถอนุรักษ์โมเมนตัมรวมของระบบได้ แต่สำหรับการอนุรักษ์พลังงานของระบบ มีเพียงการคำนวณใน 1D เท่านั้นที่สามารถทำได้เป็นอย่างดี ส่วนการคำนวณใน 3D พบว่าเมื่อรอบการคำนวณมากขึ้นจะไม่สามารถอนุรักษ์พลังงานรวมของระบบได้ แต่การเปลี่ยนแปลงยังอยู่ในระดับที่สามารถยอมรับได้ (น้อยกว่า 10% ณ รอบเวลาที่ 500)

ส่วนอุปกรณ์ผลิตพลาสมาอุณหภูมิต่ำที่ความดันบรรยากาศ – ได้ทำการสร้างอุปกรณ์ผลิต LTAP พลาสมาชนิด Dielectric Barrier Discharge ขึ้น 2 แบบ คือ แบบ parallel plate และแบบ coaxial โดยได้ออกแบบ parallel plate DBD เพื่อนำไปใช้ในการศึกษาพลาสมาเพื่อเพิ่มประสิทธิภาพของกระบวนการเอทานอลรีฟอร์มมิงในการผลิตโฮโดรเจน และออกแบบ coaxial DBD เพื่อนำไปใช้ในการศึกษาการฆ่าเชื้อโดยใช้เปลวพลาสมา สำหรับ parallel plate DBD ได้ทำการทดสอบการเกิดพลาสมาที่ช่วงความต่างศักย์และความถี่ต่างๆ โดยรายละเอียดผลการศึกษาได้รายงานไว้ใน[32] และสำหรับ coaxial DBD ได้ทำการวัดความหนาแน่นของอิเล็กตรอนด้วยวิธี Spectroscopy และได้รายงานผลการศึกษาไว้ใน[33]

## เอกสารอ้างอิง


1. H. Conrads and M. Schmidt, Plasma generation and plasma sources, Plasma Sources Science and Technology, 9, pp. 441-454 (2000)
2. Y.-M. Sung and T. Sakoda, Optimum conditions for ozone formation in a micro dielectric barrier discharge, Surface and Coatings Technology, 197, pp. 148– 153 (2005)
3. D. Lifang, L. Xuechen, and J. Pengying, Study on pattern formation in dielectric barrier discharge by optical method, Proceedings of SPIE, 6723, p. 67235A (2007)
4. Y. Zengqian, L. Xuechen, H. Mingqiang, D. Lifang, Study on the diameter and discharge duration of filament in dielectric barrier discharge in Air/Argon, Proceedings of SPIE, 6723, p. 67235B (2007)
5. J. Rahel, M. Sira, P. Stahel, and D. Trunec, The Transition Between Different Discharge Regimes in Atmospheric Pressure Air Barrier Discharge, Contributions to Plasma Physics, 47, 1-2, pp. 34 – 39 (2007)
6. G.-Q. Yang, G.-J. Zhang, Y. Ma, and W.-Y. Zhang, Plasma Distribution Characteristics of 37-kHz AC Dielectric Barrier Discharge in Atmospheric Air, IEEE Transaction on Plasma Science, 36, 4, pp. 1346-1347 (2008)
7. G.-M. Xu, Y. Ma, and G.-J. Zhang, DBD Plasma Jet in Atmospheric Pressure Argon, IEEE Transaction on Plasma Science, 36, 4, pp. 1352-1353 (2008)
8. H. Homma, H. Katayama, and K. Yasuoka, Pulsed Dielectric Barrier Discharge of Argon Gas in Gas–Liquid Two-Phase Flow, IEEE Transaction on Plasma Science, 36, 4, pp. 1344-1345 (2008)
9. C. Tendero, c. Tixier, P. Tristant, J. Desmaison, and P. Leprince, Atmospheric pressure plasmas: A review, Spectrochimica Acta Part B, 61, pp. 2 – 30 (2006)

10. T. Sakoda, K. Matsukuma, Y.-M. Sung, K. Otsubo, M. Tahara, and Y. Nakashima, Additional Plasma Surface Texturing for Single-Crystalline Silicon Solar Cells Using Dielectric Barrier Discharge, Japanese Journal of Applied Physics, 44, pp. 1730-1731 (2005)

11. M. Simor, Y. Creyghton, A. Wypkema, and J. Zemek, The Influence of Surface DBD Plasma Treatment on the Adhesion of Coatings to High-Tech Textiles, Journal of Adhesion Science and Technology, 24, 1, pp. 77-97 (2010)

12. C. Liu, N. Cui, N. M. D. Brown, and B. J. Meenan, Effects of DBD plasma operating parameters on the polymer surface modification, Surface and Coatings Technology, 185, 2-3, p. 3 (2004)

13. V. J. Rico, J. L. Hueso, J. Cotrino, V. Gallardo, B. Sarmiento, J. J. Brey, and A. R. González-Elipe, Hybrid catalytic-DBD plasma reactor for the production of hydrogen and preferential CO oxidation (CO-PROX) at reduced temperatures, Chemical Communications, 41, pp. 6192 – 6194 (2009)

14. G. R. Dey and T. N. Das, Gas-phase and On-surface Chemical Reduction of CO2 to HCHO and CO under Dielectric Barrier Discharge, Plasma Chemistry and Plasma Processing, 26, 5, pp. 495-505 (2006)

15. G. J. Pietsch and V. I. Gibalov, Dielectric barrier discharges and ozone synthesis, Pure and Applied Chemistry, 70, 6, pp. 1169-1174 (1998)

16. G. E. Morfill, M. G. Kong, and J. L. Zimmermann, Focus on Plasma Medicine, New Journal of Physics, 11, 115011, pp. 1-8 (2009)

17. J. Li and K. Dhali, Simulation of Microdischarges in a Dielectric–barrier Discharge, Journal of Applied Physicss, 82, 9, pp. 4205 (1997)

18. E. E. Kunhardt and Y. Tzeng, Development of an Electron Avalanche and its Transition into Streamers, Physical Review A, 38, 3, pp.1410 (1988)

19. D. Braun, V. Gibalov, and G. Pietsch, Two-dimensional Modeling of the Dielectric Barrier Discharge in Air, Plasma Sources Science and Technology, 1, p. 166 (1992)

20. N. Leoni and B. Paradkar, Numerical Simulation of Townsend Discharge , Paschen Breakdown and Dielectric Barrier Discharges, NIP25: 25th International Conference on Digital Printing Technologies, September 20 – 24, 2009, Louisville, Kentucky, USA.

21. G. E. Georghiou et al, Numerical Modeling of atmospheric pressure gas discharges leading to plasma production, Journal of Physics D: Applied Physics, 38, p. R303 (2005)

22. E. Panousis et al, Numerical Modeling of an atmospheric pressure dielectric barrier discharge in nitrogen: electrical and kinetic description, Journal of Physics D: Applied Physics, 40, p. 4168 (2007)

23. Y.-M. Chiu, C.-T. Hung, F.-N. Hwang, M.-H. Chiang, J.-S. Wu, S.-H. Chen, Effect of plasma chemistry on the simulation of helium atmospheric-pressure plasmas, Computer Physics Communications, 4113, pp. 1-3 (2010)

24. C. K. Birdsall and A. B. Langdon, Plasma Physics via Computer Simulation, Adam Hilger, Bristol (1991)

25. G. Font, Boundary Layer Control with Atmospheric Plasma Discharges, American Institute of Aeronautics and Astronautics Journal, 44, 7, pp. 1572-1578 (2006)

26. D. B. VanGilder, G. I. Font, and I. D. Boyd, Hybrid Monte Carlo Particle-in-Cell Simulation of an Ion Thruster Plume, Journal of Propulsion and Power, 15, 4, p.530 (1999)

27. A. F. Fuentes, R. P. Eguiluz, R. L. Callejas, A. M. Cabrera, R. V. Alvarado, S. B. Delgado, and A. P. Beneitez, Electrical Model of an Atmospheric Pressure Dielectric Barrier Discharge Cell, IEEE Transaction on Plasma Science, 37, 1, pp. 128-134 (2009)

28. Z. Chen, PSpice simulation of one atmosphere uniform glow discharge plasma (OAUGDP) reactor systems, IEEE Transaction on Plasma Science, 31, 4, pp. 511–520 (2003)

29. L. Bardos and H. Barankova, “Cold atmospheric plasma: Sources, processes, and application,” *Thin Solid Films*, 518 (2010) 6705-6713

30. NIST Electron Elastic-Scattering Cross-Section Database, SRD 64. URL: https://srdata.nist.gov/SRD64/. Retrieved July 20, 2015

31. C.K. Birdsall and A.B. Langdon, “Plasma physics via computer simulation,” The Adam Hilger Series on Plasma Physics, Adam Hilger, IOP Publishing Ltd. 1991. ISBN: 0-07-005371-5.

32. kobchai thesis

33. leeda thesis

# ภาคผนวก ก

## โปรแกรมจำลองแบบใน 1D

**cps1dn > CPS1DN.java**

```
package cps1dn;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author ppengvan
 */
public class CPS1DN {
    private Config config;
    private Grids grids;
    private Cells cells;
    private Energy energy;
    private Particles pars;
    private InteractionPairs ipairs;

    public CPS1DN() {
        config = new Config();
        grids = new Grids();
        cells = new Cells();
        energy = new Energy();
        pars = new Particles();
        ipairs = new InteractionPairs();
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
    // Initialize all elements
```

**cps1dn > CPS1DN.java**

```
        CPS1DN sim = new CPS1DN();
        sim.createSim();
        sim.createFolder();
        sim.grids.createGrids(sim.config);
        sim.cells.createCells(sim.config);
        sim.pars.createParticles(sim.config);
        sim.ipairs.createInteractionPairs(sim.config);

        System.out.println("Grid size = " + sim.config.getGridSizeX());
        System.out.println("Par dis = " + (sim.config.getXf()-
sim.config.getXi())/sim.config.getNumOfPars(1));
//    // Listing
//        sim.config.listConfig(System.out);
//        sim.ipairs.listInteractionPairs();
//        sim.grids.listGrids(sim.config);
//        sim.cells.listCells(sim.config);
//        sim.pars.listParticles(sim.config);
//        sim.ipairs.listInteractionPairs();

   // Calculate zero loop (t = 0)
        sim.pars.updateCellLoc(sim.config);
        sim.grids.updateDensity(sim.config, sim.cells, sim.pars);
        sim.grids.updatePotential(sim.config, 0);
        sim.grids.updateEField(sim.config);
        sim.energy.calculateTotalKE(sim.config, sim.pars);
        sim.energy.calculateTotalPE(sim.config, sim.grids);

        try {
            String s = sim.getNameSuffix(0);
            sim.config.saveConfig();
            sim.pars.saveParData(sim.config, s);
            sim.grids.saveGridData(sim.config, s);
            sim.energy.saveEnergyData(sim.config, s);
        } catch (FileNotFoundException | UnsupportedEncodingException ex) {
            Logger.getLogger(CPS1DN.class.getName()).log(Level.SEVERE, null, ex);
            return;
        }
```

**cps1dn > CPS1DN.java**

```
// Calculate subsequent loops
    if (sim.config.getCycle() > 0) {
        sim.pars.calculateInitialVelocity(sim.config, sim.grids, sim.cells);
        sim.pars.moveParticle(sim.config, sim.grids, sim.cells);
        sim.pars.collideParticle(sim.config, sim.grids, sim.cells, sim.ipairs);
        sim.pars.updateCellLoc(sim.config);
    }

    for (int i = 0; i < sim.config.getCycle(); i++) {

        System.out.println("Cycle: " + (i+1));
        System.out.println("--------------------------------");

        try {
            String s = sim.getNameSuffix(i+1);
            sim.pars.saveParData(sim.config, s);
        } catch (FileNotFoundException | UnsupportedEncodingException ex) {
            Logger.getLogger(CPS1DN.class.getName()).log(Level.SEVERE, null, ex);
        }

        sim.grids.updateDensity(sim.config, sim.cells, sim.pars);
        sim.grids.updatePotential(sim.config, (i+1)*sim.config.getDT());
        sim.grids.updateEField(sim.config);
        sim.energy.calculateTotalPE(sim.config, sim.grids);
        sim.pars.moveParticle(sim.config, sim.grids, sim.cells);
        sim.pars.collideParticle(sim.config, sim.grids, sim.cells, sim.ipairs);
        sim.pars.updateCellLoc(sim.config);
        sim.energy.calculateTotalKE(sim.config, sim.pars);

        try {
            String s = sim.getNameSuffix(i+1);
            sim.grids.saveGridData(sim.config, s);
            sim.energy.saveEnergyData(sim.config, s);
        } catch (FileNotFoundException | UnsupportedEncodingException ex) {
            Logger.getLogger(CPS1DN.class.getName()).log(Level.SEVERE, null, ex);
            return;
```

**cps1dn > CPS1DN.java**

```
            }
        }

        try {
            sim.grids.saveLastEFieldData(sim.config);
        } catch (FileNotFoundException | UnsupportedEncodingException ex) {
            Logger.getLogger(CPS1DN.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void createSim() {
    // Initialize Simulation Parameters
        config.setTol(Input.TOL);               // absolute tolerence for residual (for field solver)
        config.setDTol(Input.DTOL);             // tolerence for residual rate of change
        config.setW(Input.W);                   // relaxation constant
        config.setDirName(Input.DIR_NAME);          // name of data folder
        config.setSetName(Input.SET_NAME);
        config.setDT(Input.DT);
        config.setCycle(Input.CYCLE);
        config.setXi(Input.X_I);
        config.setXf(Input.X_F);
        config.setNumOfX(Input.NUM_OF_GRID_X);
        config.setVi(Input.VOLTAGE_I);
        config.setVf(Input.VOLTAGE_F);
        config.setVw(Input.VOLTAGE_W);
        config.setVt(Input.VOLTAGE_T);
        config.setGridSizeX((config.getXf()-config.getXi())/(config.getNumOfX()-1));
        config.setNumOfParType(Input.NUM_OF_PARTICLE_TYPE);
        for (int i = 0; i < Input.NUM_OF_PARTICLE_TYPE; i++) {
            config.setParClass(Input.PARTICLE_CLASS[i]);
            config.setNumOfPars(Input.NUM_OF_PARTICLE[i]);
            config.setCharge(Input.PARTICLE_CLASS[i]);
            config.setMass(Input.PARTICLE_CLASS[i]);
            config.setVXAvg(Input.V_X_AVERAGE[i]);
            config.setVXStd(Input.V_X_STANDARD_DEVIATION[i]);
        }
        config.setInType(Input.INTERACTION_TYPE);
```

**cps1dn > CPS1DN.java**

```
        for (int i = 0; i < Input.INTERACTION_TYPE; i++) {
            config.setInc(Input.INCOMING[i]);
            config.setTar(Input.TARGET[i]);
            config.setInteraction(Input.INTERACTION[i]);
            config.setXSection(Input.CROSS_SECTION[i]);
        }
    }

    public void createFolder() {
        File dir = new File(config.getDirName());
        boolean success = false;

        if (!dir.exists()) {
            try {
                dir.mkdir();
                success = true;
            }
            catch(Exception e) {
                System.out.println("Error creating data folder.");
                System.out.println("--------------------------------");
            }
            if (success) {
                System.out.println("Successfully create data folder.");
                System.out.println("--------------------------------");
            }
        } else {
            File[] files = dir.listFiles();

            for (int i = 0; i < files.length; i++) {
                if (files[i].getName().startsWith(config.getSetName() + "_")) {
                    files[i].delete();
                }
            }
        }
    }

    public String getNameSuffix(int i) {
```

**cps1dn > CPS1DN.java**

```
        String s = Integer.toString(i);
        int digit = String.valueOf(config.getCycle()).length();
        int digits = s.length();
        for (int j = 0; j < (digit-digits); j++) {
            s = "0" + s;
        }

        return s;
    }
}
```

**cps1dn > Cell.java**

```
package cps1dn;

/**
 *
 * @author ppengvan
 */
public class Cell {
    private int id;
    private int g0;
    private int g1;

    public void setID(int idVal) {
        id = idVal;
    }
    public void setG0(int g0Val) {
        g0 = g0Val;
    }
    public void setG1(int g1Val) {
        g1 = g1Val;
    }



    public int getID() {
        return id;
```

**cps1dn > Cell.java**

```
    }
    public int getG0() {
        return g0;
    }
    public int getG1() {
        return g1;
    }

}
```

**cps1dn > Cells.java**

```
package cps1dn;

import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author ppengvan
 */
public class Cells {
    private final List <Cell> cells = new ArrayList<>();
    private int numOfCells;

    void createCells(Config config) {
    // Create cells and get neighboring cells
        int i = 0;

        System.out.print("Creating cells.....");

        for (int ix = 0; ix <= config.getNumOfX() - 2; ix++) {

            Cell cll = new Cell();

            cll.setID(i);
            cll.setG0(ix);
```

**cps1dn > Cells.java**

```
            cll.setG1(ix+1);

            cells.add(cll);

            i = i + 1;

        }

        numOfCells = i;

        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public void add(Cell cell) {
        cells.add(cell);
    }
    public Cell get(int id) {
        return cells.get(id);
    }

    public void listCells(Config config) {
    // List cell values
        System.out.println("Number of Cells = " + numOfCells);
        System.out.println("--------------------------------");
        for (Cell eachCell : cells) {
            System.out.print(eachCell.getID() + "    ");
            System.out.print(eachCell.getG0() + "    ");
            System.out.println(eachCell.getG1() + "    ");
        }
    }

    public int getNumOfCells() {
        return numOfCells;
    }
}
```

**cps1dn > Config.java**

```
package cps1dn;

import crossSection.DefaultCrossSection;
import interaction.DefaultInteraction;
import java.io.FileNotFoundException;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;
import material.DefaultMaterial;
import material.Material;

/**
 *
 * @author ppengvan
 */
public class Config {
    private double tol;
    private double dtol;
    private double w;
    private String dirName;
    private String setName;
    private double dt;
    private int cycle;
    private double xi;
    private double xf;

    private double Vi;
    private double Vf;
    private double Vw;
    private double Vt;

    private double gridSizeX;

    private int numOfX;
```

**cps1dn > Config.java**

```
    private int nType;      // number of particle types
    List <DefaultMaterial> parType = new ArrayList<>();
    List <Integer> numOfPars = new ArrayList<>();
    List <Double> charge = new ArrayList<>();
    List <Double> mass = new ArrayList<>();
    List <Double> vxavg = new ArrayList<>();
    List <Double> vxstd = new ArrayList<>();
    private int inType;
    List <Integer> inc = new ArrayList<>();
    List <Integer> tar = new ArrayList<>();
    List <DefaultInteraction> interaction = new ArrayList<>();
    List <DefaultCrossSection> xSection = new ArrayList<>();

    public void setTol(double val) {
        tol = val;
    }
    public void setDTol(double val) {
        dtol = val;
    }
    public void setW(double val) {
        w = val;
    }
    public void setDirName(String name) {
        dirName = name;
    }
    public void setSetName(String name) {
        setName = name;
    }
    public void setDT(double dtVal) {
        dt = dtVal;
    }
    public void setCycle(int cVal) {
        cycle = cVal;
    }
    public void setXi(double xiVal) {
        xi = xiVal;
    }
```

**cps1dn > Config.java**

```
    public void setXf(double xfVal) {
        xf = xfVal;
    }

    public void setVi(double viVal) {
        Vi = viVal;
    }
    public void setVf(double vfVal) {
        Vf = vfVal;
    }
    public void setVw(double vwVal) {
        Vw = vwVal;
    }
    public void setVt(double vtVal) {
        Vt = vtVal;
    }

    public void setGridSizeX(double gridXVal) {
        gridSizeX = gridXVal;
    }

    public void setNumOfX(int noxVal) {
        numOfX = noxVal;
    }

    public void setNumOfParType(int typeVal) {
        nType = typeVal;
    }
    public void setParClass(DefaultMaterial val) {
        parType.add(val);
    }
    public void setNumOfPars(int nopVal) {
        numOfPars.add(nopVal);
    }
    public void setCharge(DefaultMaterial val) {
        charge.add(Material.getMaterial(val).getCharge());
    }
```

**cps1dn > Config.java**

```
    public void setMass(DefaultMaterial val) {
        mass.add(Material.getMaterial(val).getMass());
    }
    public void setVXAvg(double vxaVal) {
        vxavg.add(vxaVal);
    }
    public void setVXStd(double vxsVal) {
        vxstd.add(vxsVal);
    }

    public void setInType(int val) {
        inType = val;
    }
    public void setInc(int val) {
        inc.add(val);
    }
    public void setTar(int val) {
        tar.add(val);
    }
    public void setInteraction(DefaultInteraction val) {
        interaction.add(val);
    }
    public void setXSection(DefaultCrossSection val) {
        xSection.add(val);
    }

    public double getTol() {
        return tol;
    }
    public double getDTol() {
        return dtol;
    }
    public double getW() {
        return w;
    }
    public String getDirName() {
        return dirName;
```

**cps1dn > Config.java**

```
    }
    public String getSetName() {
        return setName;
    }
    public double getDT() {
        return dt;
    }
    public int getCycle() {
        return cycle;
    }
    public double getXi() {
        return xi;
    }
    public double getXf() {
        return xf;
    }

    public double getVi() {
        return Vi;
    }
    public double getVf() {
        return Vf;
    }
    public double getVw() {
        return Vw;
    }
    public double getVt() {
        return Vt;
    }

    public double getGridSizeX() {
        return gridSizeX;
    }

    public int getNumOfX() {
        return numOfX;
    }
```

**cps1dn > Config.java**

```
    public int getNType() {
        return nType;
    }
    public DefaultMaterial getParType(int num) {
        return parType.get(num);
    }
    public int getNumOfPars(int num) {
        return numOfPars.get(num);
    }
    public double getCharge(int num) {
        return charge.get(num);
    }
    public double getMass(int num) {
        return mass.get(num);
    }
    public double getVXAvg(int num) {
        return vxavg.get(num);
    }
    public double getVXStd(int num) {
        return vxstd.get(num);
    }

    public int getInType() {
        return inType;
    }
    public int getInc(int num) {
        return inc.get(num);
    }
    public int getTar(int num) {
        return tar.get(num);
    }
    public DefaultInteraction getInteraction(int num) {
        return interaction.get(num);
    }
    public DefaultCrossSection getXSection(int num) {
        return xSection.get(num);
```

**cps1dn > Config.java**

```
    }

    public void listConfig(PrintStream out) {
        out.println("Simulation Config");
        out.println("---------------------------------");
        out.println("dt = " + dt);
        out.println("Cycle = " + cycle);
        out.println("Xi = " + xi);
        out.println("Xo = " + xf);
        out.println("Grid X = " + gridSizeX);
        out.println("Number of X Grids = " + numOfX);
        out.println("---------------------------------");
        out.println("Number of Charge Super-Particle Types = " + nType);
        for (int i = 0; i < nType; i++) {
            out.println("======= Type " + i + " =======");
            out.println("Type = " + parType.get(i).toString());
            out.println("Number of Super-Particles = " + numOfPars.get(i));
            out.println("Charge = " + charge.get(i));
            out.println("Mass = " + mass.get(i));
            out.println("Average Velocity in X = " + vxavg.get(i));
            out.println("STDIV Velocity in X = " + vxstd.get(i));
        }
        out.println("---------------------------------");
//        out.println("Number of Background Super-Particle Types = " + nBType);
//        for (int i = 0; i < nBType; i++) {
//            out.println("======= Type " + i + " =======");
//            out.println("Type = " + bParType.get(i).toString());
//            out.println("Number of Super-Particles = " + numOfBPars.get(i));
//            out.println("Density = " + bdensity.get(i));
//            out.println("Weight = " + bweight.get(i));
//            out.println("Charge = " + bcharge.get(i));
//            out.println("Mass = " + bmass.get(i));
//            out.println("Average Velocity in X = " + bvxavg.get(i));
//            out.println("STDIV Velocity in X = " + bvxstd.get(i));
//            out.println("Average Velocity in Y = " + bvyavg.get(i));
//            out.println("STDIV Velocity in Y = " + bvystd.get(i));
//            out.println("Average Velocity in Z = " + bvzavg.get(i));
```

**cps1dn > Config.java**

```
//            out.println("STDIV Velocity in Z = " + bvzstd.get(i));
//        }
//        out.println("--------------------------------");
        out.println("Number of Interaction = " + inType);
        for (int i = 0; i < inType; i++) {
            out.println("======= Type " + i + " =======");
            out.println("Incoming Particle ID = " + inc.get(i));
            out.println("Target Particle ID = " + tar.get(i));
            out.println("Interaction = " + interaction.get(i).toString());
            out.println("Cross Section = " + xSection.get(i).toString());
        }
        out.println("--------------------------------");
    }

    public void saveConfig() throws FileNotFoundException, UnsupportedEncodingException
{
        String configFname = this.getDirName() + "/" + this.getSetName() + "_config.txt";
        System.out.print("Saving config data.....");

        try (PrintWriter file = new PrintWriter(configFname, "UTF-8")) {
            file.println("tol = " + tol);
            file.println("dtol = " + dtol);
            file.println("w = " + w);
            file.println("dt = " + dt);
            file.println("Cycle = " + cycle);
            file.println("======================");
            file.println("Xi = " + xi);
            file.println("Xo = " + xf);
            file.println("Vi = " + Vi);
            file.println("Vf = " + Vf);
            file.println("Vw = " + Vw);
            file.println("Vt = " + Vt);
            file.println("Grid Size X = " + gridSizeX);
            file.println("Number of X Grids = " + numOfX);
            file.println("======================");
            file.println("Number of Charge Super-Particle Types = " + nType);
            for (int i = 0; i < nType; i++) {
```

**cps1dn > Config.java**

```
                file.println("======= Type " + i + " =======");
                file.println("Type = " + parType.get(i).toString());
                file.println("Number of Super-Particles = " + numOfPars.get(i));
    //            out.println("Weight = " + weight.get(i));
                file.println("Charge = " + charge.get(i));
                file.println("Mass = " + mass.get(i));
                file.println("Average Velocity in X = " + vxavg.get(i));
                file.println("STDIV Velocity in X = " + vxstd.get(i));
            }
            file.println("======================");
            file.println("Number of Interaction = " + inType);
            for (int i = 0; i < inType; i++) {
                file.println("======= Type " + i + " =======");
                file.println("Incoming Particle ID = " + inc.get(i));
                file.println("Target Particle ID = " + tar.get(i));
                file.println("Interaction = " + interaction.get(i).toString());
                file.println("Cross Section = " + xSection.get(i).toString());
            }
        }
    }
}
```

**cps1dn > Energy.java**

```
package cps1dn;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.math.MathContext;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author ppengvan
```

**cps1dn > Energy.java**

```
 */
public class Energy {
    private double totalKE;
    private double totalPE;

    public void calculateTotalKE(Config config, Particles pars) {
        totalKE = 0;
        for (Particle p : pars.get()) {
            if (p.getCell() != -1) {
                totalKE = totalKE + (0.5 * config.getMass(p.getTypeID()) * 0.5 * ((p.getVXP() *
p.getVXP()) + (p.getVX() * p.getVX())));
            }
        }
    }

    public void calculateTotalPE(Config config, Grids grids) {
        totalPE = 0;
        double v = config.getGridSizeX();
        for (int gx = 0; gx < config.getNumOfX(); gx++) {
            if ((gx == 0) || (gx == config.getNumOfX()-1)) {
                totalPE = totalPE + (0.5 * grids.get(gx).getEX() *  grids.get(gx).getEX() * v / 2);
            } else {
                totalPE = totalPE + (0.5 * grids.get(gx).getEX() *  grids.get(gx).getEX() * v);
            }
        }
    }

    public double getTotalKE() {
        return totalKE;
    }

    public double getTotalPE() {
        return totalPE;
    }

    public void saveEnergyData(Config config, String slide) {
        String energyFname = config.getDirName() + "/" + config.getSetName() + "_energy.txt";
```

**cps1dn > Energy.java**

```
        System.out.print("Saving energy data from cycle " + slide + ".....");

        try (PrintWriter file = new PrintWriter(new FileOutputStream(
                new File(energyFname),
                true /* append = true */))) {
            file.println(slide + " " + config.getDT()*Integer.valueOf(slide) + " " + totalKE + " " +
totalPE + " " + (totalKE+totalPE));
            System.out.println("done.");
            System.out.println("--------------------------------");
        } catch (FileNotFoundException ex) {
            Logger.getLogger(CPS1DN.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

**cps1dn > Grid.java**

```
package cps1dn;

import java.math.MathContext;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author ppengvan
 */
public class Grid {
    private int id;
    private double x;       // location in x
    private double V;       // Voltage
    private double ex;      // E field in x direction

    List <Double> n = new ArrayList<>();    //

    public Grid(int num) {
```

**cps1dn > Grid.java**

```
        for (int i = 0; i < num; i++) {
            n.add(0.0);
        }
    }
    public void setID(int idVal) {
        id = idVal;
    }
    public void setX(double xVal) {
        x = xVal;
    }
    public void setV(double vVal) {
        V = vVal;
    }
    public void setEX(double exVal) {
        ex = exVal;
    }
    public void setN(int i, double nVal) {
        n.set(i,nVal);
    }

    public int getID() {
        return id;
    }
    public double getX() {
        return x;
    }
    public double getV() {
        return V;
    }
    public double getEX() {
        return ex;
    }
    public double getN(int i) {
        return n.get(i);
    }
}
```

**cps1dn > Grids.java**

```
package cps1dn;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

/**
 *
 * @author ppengvan
 */
public class Grids {
    private Grid[] grids;
    private int numOfX;
    private int numOfGrids;

    public void createGrids(Config config) {
    // Create grids, set boundary voltage, get neighboring grids, initialize some grid values
        int i = 0;
        int gx;
        int nType = config.getNType();
        double x = config.getXi();
        grids = new Grid[config.getNumOfX()];
        numOfX = config.getNumOfX();

        System.out.print("Creating grids.....");

        for (gx = 0; gx < numOfX; gx++) {
            Grid grd = new Grid(nType);
            grd.setID(i);
            grd.setX(x);
            this.set(grd,gx);

            i = i + 1;

            x = x + config.getGridSizeX();
```

**cps1dn > Grids.java**

```
        }

        numOfGrids = i;

        for (gx = 0; gx < numOfX; gx++) {
            if (gx == 0) {
                this.get(gx).setV(config.getVi());
            }
            else if (gx == (numOfX - 1)) {
                this.get(gx).setV(config.getVf() * Math.cos(config.getVt()));
            }
            else {
                this.get(gx).setV(0);
            }

            this.get(gx).setEX(0);
        }
        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public void set(Grid grd, int i) {
        grids[i] = grd;
    }

    public Grid get(int i) {
        return grids[i];
    }

    public void updateDensity(Config config, Cells cells, Particles pars) {
    // Calculate density at each grid
        double tv, v, v0, v1;
        double x;
        int c, oc;
        int typeID;
        int g0, g1;
        double tpar = 0;
```

**cps1dn > Grids.java**

```
        System.out.print("Calculating particles .....");

        // total volume
        tv = config.getXf() - config.getXi();
        // differential volume
        v = config.getGridSizeX();

        for (int i = 0; i < config.getNType(); i++) {
            for (int gx = 0; gx < numOfX; gx++) {       // Reset density of each grid
                this.get(gx).setN(i, 0);
            }
        }

        for (Particle eachPar : pars.get()) {
            if (eachPar.getCell() != -1) {
                c = eachPar.getCell();
                typeID = eachPar.getTypeID();
                x = eachPar.getX();
                g0 = cells.get(c).getG0();
                g1 = cells.get(c).getG1();
                v0 = x - this.get(g0).getX();
                v1 = this.get(g1).getX() - x;

                this.get(g0).setN(typeID, this.get(g0).getN(typeID) + (v1 / v));
                this.get(g1).setN(typeID, this.get(g1).getN(typeID) + (v0 / v));

                if ((g0 == 31) || (g1==31)) {
                    System.out.println();
                    System.out.println(this.get(31).getN(typeID)/v);
                }
            }
        }

        // Also add particle contribution to the boundary cell on opposite side to handle
continuity
        for (int i = 0; i < config.getNType(); i++) {
```

**cps1dn > Grids.java**

```
            double temp;

            temp = this.get(0).getN(i) + this.get(config.getNumOfX()-1).getN(i);
            this.get(0).setN(i, temp);
            this.get(config.getNumOfX()-1).setN(i, temp);
        }

        for (int i = 0; i < config.getNType(); i++) {
            for (int gx = 0; gx < numOfX; gx++) {
                this.get(gx).setN(i, this.get(gx).getN(i) / v);
            }
        }

        for (int i = 0; i < config.getNType(); i++) {
            tpar = tpar + (this.get(0).getN(i) * v / 2);
            for (int gx = 1; gx < numOfX-1; gx++) {
                tpar = tpar + (this.get(gx).getN(i) * v);
//                      System.out.print(this.get(gx,gy,gz).getN(i) + " ");
            }
            tpar = tpar + (this.get(numOfX-1).getN(i) * v / 2);

//                  System.out.println();
        }

        System.out.println("done.");
        System.out.println("Total number of particles = " + tpar);
        System.out.println("--------------------------------");
    }

    public void updatePotential(Config config, double time) {
    // Calculate potential at each grid, i.e. solving Poisson's equation
        double V = 0;
        double res = 0;
        double f = 0;
        double tres = 100;
        double ptres = 1;
        double dtres = 100;
```

**cps1dn > Grids.java**

```
        double dx = config.getGridSizeX();
        int ct = 0;

        System.out.println("Calculating grid potential.....");

        this.get(0).setV(config.getVi());
        this.get(numOfX-1).setV(config.getVf() * Math.cos((config.getVw() * time) +
config.getVt()));

        while ((tres > config.getTol()) || (dtres > config.getDTol())) {
            ct = ct + 1;
            tres = 0;
            for (int gx = 0; gx < numOfX; gx++) {
                if ((gx != 0) && (gx != (numOfX - 1))) {
                    f = 0;
                    for (int i = 0; i < config.getNType(); i++) {
                        f = f + (config.getCharge(i) * this.get(gx).getN(i));
                    }
                    if (gx == 0) {
                        res = f + (1 / (dx * dx) * (this.get(numOfX-2).getV() + this.get(gx+1).getV() -
(2 * this.get(gx).getV())));
                        V = (1.0 / 2.0) * ((f * dx * dx) + (this.get(numOfX-2).getV() +
this.get(gx+1).getV()));
                    }
                    else if (gx == (numOfX-1)) {
                        res = f + (1 / (dx * dx) * (this.get(gx-1).getV() + this.get(1).getV() - (2 *
this.get(gx).getV())));
                        V = (1.0 / 2.0) * ((f * dx * dx) + (this.get(gx-1).getV() + this.get(1).getV()));
                    }
                    else {
                        res = f + (1 / (dx * dx) * (this.get(gx-1).getV() + this.get(gx+1).getV() - (2 *
this.get(gx).getV())));
                        V = (1.0 / 2.0) * ((f * dx * dx) + (this.get(gx-1).getV() + this.get(gx+1).getV()));
                    }

                    this.get(gx).setV((1 - config.getW()) * this.get(gx).getV() + config.getW() * V);
                }
```

**cps1dn > Grids.java**

```
                tres = tres + (res * res);
//                System.out.println(tres);
            }
            tres = Math.sqrt(tres);
            dtres = Math.abs(tres - ptres) / ptres;
            ptres = tres;
            if (ct%1000 == 0) {
                System.out.println("Iternation #: " + ct);
                System.out.println("Total residual = " + tres);
            }
        }

        System.out.println("Number of iterations: " + ct);
        System.out.println("Total residual = " + tres);
        System.out.println("DTotal residual = " + dtres);
        System.out.println("--------------------------------");
    }

    public void updateEField(Config config) {
    // Calculate electric field at each grid from the neighboring potentials
        double Ex = 0;

        System.out.print("Calculating grid electric field.....");

        for (int gx = 0; gx < numOfX; gx++) {
            if (gx == 0) {
                Ex = -(this.get(gx+1).getV() - this.get(numOfX-2).getV()) / (2 *
config.getGridSizeX());
            }
            else if (gx == (numOfX - 1)) {
                Ex = -(this.get(1).getV() - this.get(gx-1).getV()) / (2 * config.getGridSizeX());
            }
            else {
                Ex = -(this.get(gx+1).getV() - this.get(gx-1).getV()) / (2 * config.getGridSizeX());
            }
```

**cps1dn > Grids.java**

```
            this.get(gx).setEX(Ex);
        }

        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public int getNumOfGrids() {
        return numOfGrids;
    }

    public double getDensity(Config config, int type, double x) {
        int i = (int)((x - config.getXi()) / config.getGridSizeX());

        double v = config.getGridSizeX();
        double v0 = x - this.get(i).getX();
        double v1 = this.get(i+1).getX() - x;

        double n = (this.get(i).getN(type) * v1 / v) + (this.get(i+1).getN(type) * v0 / v);

        return n;
    }

    public void listGrids(Config config) {
    // List grid values
        System.out.println("Number of Grids = " + numOfGrids);
        System.out.println("Number of X = " + numOfX);
        System.out.println("--------------------------------");
        for (int gx = 0; gx < numOfX; gx++) {
            System.out.print(this.get(gx).getID() + "    ");
        //          System.out.print(eachGrid.getGX0() + "    ");
        //          System.out.print(eachGrid.getGX1() + "    ");
        //          System.out.print(eachGrid.getGY0() + "    ");
        //          System.out.print(eachGrid.getGY1() + "    ");
            System.out.print(this.get(gx).getX() + "    ");
            System.out.print(this.get(gx).getV() + "    ");
            System.out.print(this.get(gx).getEX() + "    ");
```

**cps1dn > Grids.java**

```
            for (int i = 0; i < config.getNType(); i++) {
                System.out.print(this.get(gx).getN(i) + "    ");
            }
            System.out.println();
        }
        System.out.println("--------------------------------");
    }

    public void saveGridData(Config config, String slide) throws FileNotFoundException,
UnsupportedEncodingException {
        String gridFname = config.getDirName() + "/" + config.getSetName() + "_grid_" + slide
+ ".txt";
        String evtFname = config.getDirName() + "/" + config.getSetName() + "_VE_t.txt";
        System.out.print("Saving voltage data from cycle " + slide + ".....");

        try (PrintWriter file = new PrintWriter(gridFname, "UTF-8")) {
            file.println(slide);
            file.println(config.getNType());
            for (int i = 0; i < numOfGrids; i++) {
                file.print(this.get(i).getID() + " ");
                file.print(this.get(i).getX() + " ");
                file.print(this.get(i).getV() + " ");
                file.print(this.get(i).getEX() + " ");
                for (int j = 0; j < config.getNType(); j++) {
                    file.print(this.get(i).getN(j) + " ");
                }
                file.println();
            }
        }
        try (PrintWriter file1 = new PrintWriter(new FileOutputStream(
                new File(evtFname),
                true /* append = true */))) {

            file1.println(slide + " " + config.getDT() + " " + this.get(numOfX/2).getV() + " " +
this.get(numOfX/2).getEX());
        }
```

**cps1dn > Grids.java**

```
        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public void saveLastEFieldData(Config config) throws FileNotFoundException,
UnsupportedEncodingException {
        String exxFname = config.getDirName() + "/" + config.getSetName() + "_Ex_x.txt";

        System.out.print("Saving electric field data from last slide.....");

        try (PrintWriter file1 = new PrintWriter(exxFname, "UTF-8")) {
            file1.println(config.getNType());
            for (int i = 0; i < numOfX; i++) {
                file1.print(this.get(i).getX() + " ");
                file1.print(this.get(i).getEX() + " ");
                file1.println();
            }
        }

        System.out.println("done.");
        System.out.println("--------------------------------");
    }

}
```

**cps1dn > Input.java**

```
package cps1dn;

import interaction.DefaultInteraction;
import crossSection.DefaultCrossSection;
import material.DefaultMaterial;

/**
 *
 * @author ppengvan
 */
```

**cps1dn > Input.java**

```
public class Input {

// File parameters
    // name of data folder
    final static String DIR_NAME = "/Users/ppengvan/Documents/Data1D";
//    final static String DIR_NAME = "D:\\Data1D";
    // data set name to append
    final static String SET_NAME = "3dset5";

// Calculation parameters
    // absolute tolerence for residual (for field solver)
    final static double TOL = 1e-8;
    // tolerence for residual rate of change
    final static double DTOL = 1e-4;
    // relaxation constant
    final static double W = 1.9;
    // Time step, normalized by 2pi/omega_p
    final static double DT = 0.002;
    // Time cycle to run;
    final static int CYCLE = 300;

// Grid parameters
    // Starting x, normailzed by lamda_d
    final static double X_I = 0; // Do not change
    // Ending x, normailzed by lamda_d
    final static double X_F = 4*Math.PI;

    // Number of grids in x, must (X_F - X_I)
    final static int NUM_OF_GRID_X = 513;

    // Voltage at X_I, normalized by kT_e/e
    final static double VOLTAGE_I = 0; // Do not change
    // Voltage at X_F, normalized by kT_e/e
    final static double VOLTAGE_F = 0;
    // Voltage frequency, normalized by omega_p
    final static double VOLTAGE_W = 0;
    // Initial phase of voltage V_f*cos(w*t+theta)
```

**cps1dn > Input.java**

```
    final static double VOLTAGE_T = 0;

// Charge particle parameters
    // Types of charge particles
    final static int NUM_OF_PARTICLE_TYPE = 2;
    // Classes of charge particles
    final static DefaultMaterial[] PARTICLE_CLASS = {DefaultMaterial.ELECTRON,
DefaultMaterial.ARGON_1};
    // Number of super-particle, Suggestion: Should be < 1e7, normalized by
(epsilon_0*kT_e)/(lamda_d^2*e^2)
    final static int[] NUM_OF_PARTICLE = {16384, 16384};
    // Distribution of particle's position is assumed to be uniform
    // Distribution of particle's velopity is assumed to be Gaussian
    // Average velocity in x, normalized by lamda_d*omega_p/(2pi)
    final static double[] V_X_AVERAGE = {0, 0};
    // Standard deviation of velocity in x, normalized by lamda_d*omega_p/(2pi), = 1 when
kT = 1 eV
    final static double[] V_X_STANDARD_DEVIATION = {0, 0};

// Monte Carlo Interaction parameters
    // Interaction pairs are setup using 2 fields to reference incoming
    // and target particles: "INCOMING[i]" and "TARGET[i]".
    // "INCOMING[i]" represents the incoming particle type i whereas
    // "TARGET[i]" represents the target particle type i.
    // INCOMING[i] will interact with TARGET[i]
    final static int INTERACTION_TYPE = 0;
    final static int[] INCOMING = {0};
    final static int[] TARGET = {2};
    final static DefaultInteraction[] INTERACTION =
{DefaultInteraction.CLASSIC_ELASTIC_COLLISION};
    final static DefaultCrossSection[] CROSS_SECTION = {DefaultCrossSection.E_AR};
}
```

**cps1dn > InteractionPair.java**

```
package cps1dn;
```

**cps1dn > InteractionPair.java**

```
import crossSection.DefaultCrossSection;
import interaction.DefaultInteraction;
import java.util.ArrayList;
import java.util.List;
import material.DefaultMaterial;

/**
 *
 * @author ppengvan
 */
public class InteractionPair {
    private DefaultMaterial inc;
    private DefaultMaterial tar;
    private DefaultInteraction inter;
    private DefaultCrossSection xsec;

    public void setInteraction(DefaultMaterial incVal, DefaultMaterial tarVal,
DefaultInteraction interVal, DefaultCrossSection xsVal) {
        inc = incVal;
        tar = tarVal;
        inter = interVal;
        xsec = xsVal;
    }

    public DefaultMaterial getInc() {
        return inc;
    }
    public DefaultMaterial getTar() {
        return tar;
    }
    public DefaultInteraction getInter() {
        return inter;
    }
    public DefaultCrossSection getXSec() {
        return xsec;
    }
}
```

**cps1dn > InteractionPairs.java**

```
package cps1dn;

import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author ppengvan
 */
public class InteractionPairs {
    private List<InteractionPair> ipairs = new ArrayList<>();
    private int inType;

    public void createInteractionPairs(Config config) {
        inType = config.getInType();
        for (int i = 0; i < inType; i++) {
            InteractionPair ipair = new InteractionPair();
            ipair.setInteraction(config.getParType(config.getInc(i)),
config.getParType(config.getTar(i)), config.getInteraction(i), config.getXSection(i));

            ipairs.add(ipair);
        }
    }

    public void listInteractionPairs() {
        System.out.println("Interaction Pairs");
        System.out.println("--------------------------------");
        if (inType == 0) {
            System.out.println("No interation");
        } else {
            for (int i = 0; i < inType; i++) {
                System.out.print("Pair #" + i + " : ");
                System.out.println(ipairs.get(i).getInter().toString() + " between " +
ipairs.get(i).getInc().toString() + " and " + ipairs.get(i).getTar().toString());
            }
```

**cps1dn > InteractionPairs.java**

```
        }

        System.out.println("--------------------------------");
    }

    public List<InteractionPair> getInteractionPairs() {
        return ipairs;
    }
}
```

**cps1dn > Particle.java**

```
package cps1dn;

/**
 *
 * @author ppengvan
 */
public class Particle {
    private int id;
    private int cell;
    private double x;
    private double vx;
    private double vxp;
    private int typeID;

    public void setID(int idVal) {
        id = idVal;
    }
    public void setTypeID(int typeIDVal) {
        typeID = typeIDVal;
    }
    public void setCell(int cellVal) {
        cell = cellVal;
    }
    public void setX(double xVal) {
        x = xVal;
```

**cps1dn > Particle.java**

```
    }
    public void setVX(double vxVal) {
        vx = vxVal;
    }
    public void setVXP(double vxpVal) {
        vxp = vxpVal;
    }

    public int getID() {
        return id;
    }
    public int getTypeID() {
        return typeID;
    }
    public int getCell() {
        return cell;
    }
    public double getX() {
        return x;
    }
    public double getVX() {
        return vx;
    }
    public double getVXP() {
        return vxp;
    }
}
```

**cps1dn > Particles.java**

```
package cps1dn;

import crossSection.CrossSection;
import crossSection.DefaultCrossSection;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
```

**cps1dn > Particles.java**

```
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import material.DefaultMaterial;

/**
 *
 * @author ppengvan
 */
public class Particles {
    private final List<Particle> pars = new ArrayList<>();
    private int numOfPars = 0;
    private int numOfParsInSys = 0;
    private int numOfInteractions = 0;

    public void createParticles(Config config) {
    // Create particles, initialize particle values
        int k = 0;

        System.out.print("Creating super-particles.....");

        for (int j = 0; j < config.getNType(); j++) {
            Random randx = new Random();
            Random randvx = new Random();

            for (int i = 0; i < config.getNumOfPars(j); i++) {
                Particle par = new Particle();
                par.setID(k);
                par.setTypeID(j);

                par.setX(randx.nextDouble()*(config.getXf()-config.getXi())+config.getXi());  // or
nextGaussian()
//                Instead of randomspacing, evenly spacing the electron and perturb a small
amount to simulate the plasma frequency
```

**cps1dn > Particles.java**

```
//                double x = ((i+0.5) * (config.getXf() - config.getXi()) / config.getNumOfPars(j)) +
config.getXi();
//                if (j == 0) {
//                    par.setX(x + (0.001 * Math.sin(x)));
////                    par.setX(x+0.001);
//                }
//                else {
//                    par.setX(x);
//                }

                double vx = (randvx.nextGaussian() * config.getVXStd(j)) + config.getVXAvg(j);
                par.setVX(vx);
                par.setVXP(vx);
                par.setCell(0);

                pars.add(par);
                k = k + 1;
            }
        }

        numOfPars = k;

        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public void updateCellLoc(Config config) {
    // Find out whether each particle is still inside the system, and in which cell the
particle resides
        numOfParsInSys = 0;
        double dx = config.getGridSizeX();

        System.out.print("Counting super-particles in the system.....");

        for (Particle eachPar : pars) {
            double x = eachPar.getX();
```

**cps1dn > Particles.java**

```
            // Continuous boundary in x
            if (x < config.getXi()) {
                x = config.getXf() - ((config.getXi() - x) % (config.getXf() - config.getXi()));
                eachPar.setX(x);
            } else if (x > config.getXf()) {
                x = config.getXi() + ((x - config.getXf()) % (config.getXf() - config.getXi()));
                eachPar.setX(x);
            }

            if (isInside(config, eachPar)) {
                int cellLoc = (int)((x - config.getXi()) / dx);

                eachPar.setCell(cellLoc);
                numOfParsInSys = numOfParsInSys + 1;
            }
            else {
                eachPar.setCell(-1);
            }
        }
        System.out.println("done.");
        System.out.println("Number of super-particles in the system = " + numOfParsInSys);
        System.out.println("--------------------------------");
    }

    public void calculateInitialVelocity(Config config, Grids grids, Cells cells) {
    // Move particle based on Lorentz force due to electric & magnetic fields back half
time step
        double Epx = 0;
        int c;
        int typeID;
        double x;
        double dx;
        int g0, g1;
        double v, v0, v1;
        double vx;
        double dt;
```

**cps1dn > Particles.java**

```
        System.out.print("Calculate First Velocity...");

        dt = config.getDT();

        v = config.getGridSizeX();

        for (Particle eachPar : pars) {
            if (eachPar.getCell() != -1) {
                c = eachPar.getCell();
                typeID = eachPar.getTypeID();
                x = eachPar.getX();
                g0 = cells.get(c).getG0();
                g1 = cells.get(c).getG1();
                v0 = x - grids.get(g0).getX();
                v1 = grids.get(g1).getX() - x;

                Epx = (grids.get(g0).getEX() * v1 /v) + (grids.get(g1).getEX() * v0 / v);

                vx = eachPar.getVX() - (dt / 2 * config.getCharge(eachPar.getTypeID()) /
config.getMass(eachPar.getTypeID()) * Epx);

                eachPar.setVX(vx);
                eachPar.setVXP(vx);
            }
        }
        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public void moveParticle(Config config, Grids grids, Cells cells) {
    // Move particle based on Lorentz force due to electric & magnetic fields
        double Epx = 0;
        int c;
        int typeID;
        double x;
        double dx;
        int g0, g1;
```

**cps1dn > Particles.java**

```
        double v, v0, v1;
        double vx;
        double dt;
        double qprime;

        System.out.print("Moving charge super-particles.....");

        dt = config.getDT();

        v = config.getGridSizeX();

        for (Particle eachPar : pars) {
            if (eachPar.getCell() != -1) {
                c = eachPar.getCell();
                typeID = eachPar.getTypeID();
                x = eachPar.getX();
                g0 = cells.get(c).getG0();
                g1 = cells.get(c).getG1();
                v0 = x - grids.get(g0).getX();
                v1 = grids.get(g1).getX() - x;

                Epx = (grids.get(g0).getEX() * v1 / v) + (grids.get(g1).getEX() * v0 / v);

                qprime = dt * config.getCharge(eachPar.getTypeID()) /
config.getMass(eachPar.getTypeID());

                vx = eachPar.getVX();
                eachPar.setVXP(vx);

                vx = vx + (qprime * Epx);

                dx = dt * vx;

                x = x + dx;

                eachPar.setX(x);
                eachPar.setVX(vx);
```

**cps1dn > Particles.java**

```
            }
        }
        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public void collideParticle(Config config, Grids grids, Cells cells, InteractionPairs ipairs) {
        System.out.print("Colliding particles.....");
        for (int i = 0; i < config.getInType(); i++) {
            DefaultCrossSection xsec = ipairs.getInteractionPairs().get(i).getXSec();
            Random rand1 = new Random();
            Random rand2 = new Random();
            Random rand3 = new Random();
            Double r1 = rand1.nextDouble();
            Double r2, r3;
            double theta, phi;
            double tv = config.getXf() - config.getXi();

            for (Particle p : pars) {
                if (config.getInc(i) == p.getTypeID()) {
                    if (isInside(config, p)) {
                        double u = Math.abs(p.getVX());
                        double nTar = config.getNumOfPars(config.getTar(i)) / tv;
                        double xs = CrossSection.getCrossSection(xsec).getTotalXSection();
                        double ts = config.getDT();
                        double prob = 1 - Math.exp(-u * nTar * xs * ts);
                        System.out.println("u=" + u + ",nTar=" + nTar + ",xs=" + xs + ",ts=" + ts +
",prob=" + prob);
                        if (prob > r1) {
                            r2 = rand2.nextDouble();
                            r3 = rand3.nextDouble();
                            theta = Math.PI*r2;
                            phi = Math.PI * 2 * r3;
                            p.setVX(u * Math.sin(theta)*Math.cos(phi));
                            numOfInteractions++;
                        }
                    }
```

**cps1dn > Particles.java**

```
            }
        }
    }
    System.out.println("done.");
    System.out.println("--------------------------------");
}

public boolean isInside(Config c, Particle p) {
    double x = p.getX();

    if ((x >= c.getXi()) && (x <= c.getXf())) {
        return true;
    }
    else {
        return false;
    }
}

public List<Particle> get(){
    return pars;
}

public Particle get(int i){
    return pars.get(i);
}

public int getNumOfPars() {
    return numOfPars;
}

public int getNumParsInSys() {
    return numOfParsInSys;
}

public void listParticles(Config config) {
// List particle values
    System.out.println("Number of Super-Particles = " + numOfPars);
```

**cps1dn > Particles.java**

```
        System.out.println("--------------------------------");
        for (Particle eachPar : pars) {
            System.out.print(eachPar.getID() + "    ");
            System.out.print(eachPar.getCell() + "    ");
            System.out.print(eachPar.getTypeID() + "    ");
            System.out.print(eachPar.getX() + "    ");
            System.out.print(eachPar.getVX() + "    ");
        }
        System.out.println("--------------------------------");
    }

    public void saveParData(Config config, String slide) throws FileNotFoundException,
UnsupportedEncodingException {
        String parFname = config.getDirName() + "/" + config.getSetName() + "_XYZ_" + slide
+ ".txt";
        String phaseFname1 = config.getDirName() + "/" + config.getSetName() +
"_phase1.txt";
        String phaseFname2 = config.getDirName() + "/" + config.getSetName() +
"_phase2.txt";
        String phaseFname3 = config.getDirName() + "/" + config.getSetName() +
"_phase3.txt";

        System.out.print("Saving particle data for excel from cycle " + slide + ".....");

        try (PrintWriter file = new PrintWriter(parFname, "UTF-8")) {
            file.println(slide);
            for (int i = 0; i < numOfPars; i++) {
                if (pars.get(i).getCell() != -1) {
                    file.println(pars.get(i).getTypeID() + " " + pars.get(i).getID() + " " +
pars.get(i).getCell() + " " + pars.get(i).getX() + " " + pars.get(i).getVX());
                    if (pars.get(i).getID() == 0) {
                        try (PrintWriter file1 = new PrintWriter(new FileOutputStream(
                                new File(phaseFname1),
                                true /* append = true */))) {
                            file1.println(pars.get(i).getX() + " " + pars.get(i).getVX());
                        }
                    }
```

**cps1dn > Particles.java**

```
                if (pars.get(i).getID() == 10) {
                    try (PrintWriter file2 = new PrintWriter(new FileOutputStream(
                            new File(phaseFname2),
                            true /* append = true */))) {
                        file2.println(pars.get(i).getX() + " " + pars.get(i).getVX());
                    }
                }
                if (pars.get(i).getID() == 100) {
                    try (PrintWriter file3 = new PrintWriter(new FileOutputStream(
                            new File(phaseFname3),
                            true /* append = true */))) {
                        file3.println(pars.get(i).getX() + " " + pars.get(i).getVX());
                    }
                }
            }
        }
    }

    System.out.println("done.");
    System.out.println("--------------------------------");
  }
}
```

# ภาคผนวก ข

## โปรแกรมจำลองแบบใน 3D

**cps3dn > Cps3dn.java**

```
package cps3dn;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.time.LocalDateTime;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.geometry.Rectangle2D;
import javafx.scene.Scene;
import javafx.stage.Screen;
import javafx.stage.Stage;

/**
 *
 * @author ppengvan
 */
public class CPS3DN extends Application {
    Config config;
    Normalization normal;
    Grids grids;
    Cells cells;
    Particles pars;
    Energy energy;
    Momentum momentum;
    InteractionPairs ipairs;

    @Override
    public void start(Stage primaryStage) {
//    // Testing Math
```

**cps3dn > Cps3dn.java**

```
//        double[][] a = {{1,2,1},{2,1,3},{3,1,1.5}};
//        System.out.println(CPSMath.determinant(a));
//        System.exit(0);

    // Initialize
        long tStart = System.nanoTime();
        config = new Config();
        normal = new Normalization(config);
        grids = new Grids(config);
        cells = new Cells(config);
        pars = new Particles(config);
        energy = new Energy();
        momentum = new Momentum();
        ipairs = new InteractionPairs(config);
        createFolder();

        try {
            config.saveConfig();
            cells.saveCellData(config);
        } catch (FileNotFoundException | UnsupportedEncodingException ex) {
            Logger.getLogger(CPS3DN.class.getName()).log(Level.SEVERE, null, ex);
        }

//    // Listing
//        config.listConfig(System.out);
//        ipairs.listInteractionPairs();
//        grids.listGrids(config);
//        cells.listCells(config);
//        pars.listCParticles(config);
//        ipairs.listInteractionPairs();

    // Calculate first loop
        pars.updateCellLoc(config);
        grids.updateDensity(config, cells, pars);
        grids.updatePotentialPeriodic(config, 0);
        grids.updateEFieldPeriodic(config);
        energy.calculateTotalKE(config, pars);
```

**cps3dn > Cps3dn.java**

```
        energy.calculateTotalPE(config, grids);
        momentum.calculateParticleMomentum(config, pars);
        momentum.calculateFieldMomentum(config, grids);

        try {
            String s = getNameSuffix(0);
            pars.saveParData(config, s);
            grids.saveGridData(config, s);
            grids.savePotentialXData4Excel(config, s, 0);
            grids.savePotentialYData4Excel(config, s, 0);
            grids.savePotentialZData4Excel(config, s, 0);
            energy.saveEnergyData(config, s);
            momentum.saveMomentumData(config, s);
        } catch (FileNotFoundException | UnsupportedEncodingException ex) {
            Logger.getLogger(CPS3DN.class.getName()).log(Level.SEVERE, null, ex);
            return;
        }

    // Calculate subsequent loops
        if (config.getCycle() > 0) {
            pars.calculateInitialVelocity(config, grids, cells);
            pars.moveParticle(config, grids, cells);
            pars.collideParticle(config, grids, cells, ipairs);
            pars.updateCellLoc(config);
        }

        for (int i = 0; i < config.getCycle(); i++) {
            System.out.println("Cycle: " + (i+1));
            System.out.println("-------------------------------");



            try {
                String s = getNameSuffix(i+1);
                pars.saveParData(config, s);
            } catch (FileNotFoundException | UnsupportedEncodingException ex) {
                Logger.getLogger(CPS3DN.class.getName()).log(Level.SEVERE, null, ex);
            }
```

**cps3dn > Cps3dn.java**

```
            grids.updateDensity(config, cells, pars);
            grids.updatePotentialPeriodic(config, (i+1)*config.getDT());
            grids.updateEFieldPeriodic(config);
            energy.calculateTotalPE(config, grids);
            momentum.calculateFieldMomentum(config, grids);
            pars.moveParticle(config, grids, cells);
            pars.collideParticle(config, grids, cells, ipairs);
            pars.updateCellLoc(config);
            energy.calculateTotalKE(config, pars);
            momentum.calculateParticleMomentum(config, pars);

            try {
                String s = getNameSuffix(i+1);
                grids.saveGridData(config, s);
                grids.savePotentialXData4Excel(config, s, 0);
                grids.savePotentialYData4Excel(config, s, 0);
                grids.savePotentialZData4Excel(config, s, 0);
                energy.saveEnergyData(config, s);
                momentum.saveMomentumData(config, s);
            } catch (FileNotFoundException | UnsupportedEncodingException ex) {
                Logger.getLogger(CPS3DN.class.getName()).log(Level.SEVERE, null, ex);
                return;
            }
        }

        System.out.println("Number of interaction: " + pars.getNumOfInteractions());

    // UI
//        createUI(primaryStage);
        long duration = System.nanoTime()-tStart;
        System.out.println("Total run-time: " + (duration/1e9) + " seconds.");
        System.exit(0);
    }

    /**
     * @param args the command line arguments
```

**cps3dn > Cps3dn.java**

```
 */
public static void main(String[] args) {
    launch(args);
}

void createFolder() {
    File dir = new File(config.getDirName());
    boolean success = false;

    if (!dir.exists()) {
        try {
            dir.mkdir();
            success = true;
        }
        catch(Exception e) {
            System.out.println("Error creating data folder.");
            System.out.println("--------------------------------");
        }
        if (success) {
            System.out.println("Successfully create data folder.");
            System.out.println("--------------------------------");
        }
    } else {
        File[] files = dir.listFiles();

        for (int i = 0; i < files.length; i++) {
            if (files[i].getName().startsWith(config.getSetName() + "_")) {
                files[i].delete();
            }
        }
    }
}

String getNameSuffix(int i) {
    String s = Integer.toString(i);
    int digit = String.valueOf(config.getCycle()).length();
    int digits = s.length();
```

**cps3dn > Cps3dn.java**

```
        for (int j = 0; j < (digit-digits); j++) {
            s = "0" + s;
        }

        return s;
    }

    void createUI(Stage primaryStage) {
        UIPrimary ui = new UIPrimary(pars);
        Rectangle2D visualBounds = Screen.getPrimary().getVisualBounds();
        Scene scene = new Scene(ui.getPane(), visualBounds.getWidth(),
visualBounds.getHeight());

        ui.factory.addSceneSizeChangedListener(ui.chart, scene);

        primaryStage.setTitle("Particle Simulator");
        primaryStage.setScene(scene);
        primaryStage.show();
    }



}
```

**cps3dn > CPSMath.java**

```
package cps3dn;

/**
 *
 * @author Administrator
 */
public class CPSMath {
    public static double determinant(double[][] mat) {
        int row = mat.length;
        int col = 0;
        double[][] minor;
```

**cps3dn > CPSMath.java**

```
        double det = 0;
        if (row > 0) {
            col = mat[0].length;
        }

        if (row != col) {
            System.out.println("Mathematic Error!!!!!");
        } else if (row > 1) {
            printMatrix(mat);
            minor = new double[row-1][col-1];

            for (int i = 0; i < col; i++) {
                int m = 1;
                int n = 0;
                int p = 0;
                int q = 0;

                while (m < row) {
                    n = 0;
                    q = 0;
                    while (n < col) {
                        if (n != i) {
                            minor[p][q] = mat[m][n];
                            q++;
                        }
                        n++;
                    }
                    p++;
                    m++;
                }

                det = det + mat[0][i]*Math.pow(-1,i)*determinant(minor);
            }
        } else {
            det = mat[0][0];
        }
```

**cps3dn > CPSMath.java**

```
        return det;
    }

    public static void printMatrix(double[][] mat) {
        System.out.println();
        System.out.print("{");
        for (int i = 0; i < mat.length; i++) {
            System.out.print("{");
            for (int j = 0; j < mat[0].length; j++) {
                System.out.print(mat[i][j] + " ");
            }
            System.out.print("}");
        }
        System.out.println("}");
    }

    public static void crossproduct(double[] a, double[] b, double[] c) {
        a[0] = b[1]*c[2]-b[2]*c[1];
        a[1] = b[2]*c[0]-b[0]*c[2];
        a[2] = b[0]*c[1]-b[1]*c[0];
    }

    public static void addproduct(double[] a, double[] b, double[] c) {
        a[0] = b[0]+c[0];
        a[1] = b[1]+c[1];
        a[2] = b[2]+c[2];
    }
}
```

**cps3dn > Cell.java**

```
package cps3dn;

/**
 *
 * @author ppengvan
 */
```

**cps3dn > Cell.java**

```
public class Cell {
    private int id;
    private int g0;
    private int g1;
    private int g2;
    private int g3;
    private int g4;
    private int g5;
    private int g6;
    private int g7;

    public void setID(int idVal) {
        id = idVal;
    }
    public void setG0(int g0Val) {
        g0 = g0Val;
    }
    public void setG1(int g1Val) {
        g1 = g1Val;
    }
    public void setG2(int g2Val) {
        g2 = g2Val;
    }
    public void setG3(int g3Val) {
        g3 = g3Val;
    }
    public void setG4(int g4Val) {
        g4 = g4Val;
    }
    public void setG5(int g5Val) {
        g5 = g5Val;
    }
    public void setG6(int g6Val) {
        g6 = g6Val;
    }
    public void setG7(int g7Val) {
        g7 = g7Val;
```

**cps3dn > Cell.java**

```
    }

    public int getID() {
        return id;
    }
    public int getG0() {
        return g0;
    }
    public int getG1() {
        return g1;
    }
    public int getG2() {
        return g2;
    }
    public int getG3() {
        return g3;
    }
    public int getG4() {
        return g4;
    }
    public int getG5() {
        return g5;
    }
    public int getG6() {
        return g6;
    }
    public int getG7() {
        return g7;
    }
}
```

**cps3dn > Cells.java**

```
package cps3dn;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
```

**cps3dn > Cells.java**

```
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author ppengvan
 */
public class Cells {
    private final List <Cell> cells = new ArrayList<>();
    private int numOfCells;

    public Cells(Config config) {
    // Create cells and get neighboring cells
        int i = 0;

        System.out.print("Creating cells.....");

        for (int ix = 0; ix <= config.getNumOfX() - 2; ix++) {
            for (int iy = 0; iy <= config.getNumOfY() - 2; iy++) {
                for (int iz = 0; iz <= config.getNumOfZ() - 2; iz++) {
                    Cell cll = new Cell();

                    cll.setID(i);

cll.setG0(ix*(config.getNumOfY()*config.getNumOfZ())+iy*config.getNumOfZ()+iz);

cll.setG1(ix*(config.getNumOfY()*config.getNumOfZ())+iy*config.getNumOfZ()+iz+1);

cll.setG2(ix*(config.getNumOfY()*config.getNumOfZ())+(iy+1)*config.getNumOfZ()+iz);

cll.setG3(ix*(config.getNumOfY()*config.getNumOfZ())+(iy+1)*config.getNumOfZ()+iz+1);

cll.setG4((ix+1)*(config.getNumOfY()*config.getNumOfZ())+iy*config.getNumOfZ()+iz);

cll.setG5((ix+1)*(config.getNumOfY()*config.getNumOfZ())+iy*config.getNumOfZ()+iz+1);
```

**cps3dn > Cells.java**

```
cll.setG6((ix+1)*(config.getNumOfY()*config.getNumOfZ())+(iy+1)*config.getNumOfZ()+iz);

cll.setG7((ix+1)*(config.getNumOfY()*config.getNumOfZ())+(iy+1)*config.getNumOfZ()+iz+1);

                    cells.add(cll);

                    i = i + 1;
                }
            }
        }

        numOfCells = i;

        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public void add(Cell cell) {
        cells.add(cell);
    }
    public Cell get(int id) {
        return cells.get(id);
    }

    public void listCells(Config config) {
    // List cell values
        System.out.println("Number of Cells = " + numOfCells);
        System.out.println("--------------------------------");
        for (Cell eachCell : cells) {
            System.out.print(eachCell.getID() + "    ");
            System.out.print(eachCell.getG0() + "    ");
            System.out.print(eachCell.getG1() + "    ");
            System.out.print(eachCell.getG2() + "    ");
            System.out.print(eachCell.getG3() + "    ");
            System.out.print(eachCell.getG4() + "    ");
            System.out.print(eachCell.getG5() + "    ");
```

**cps3dn > Cells.java**

```
            System.out.print(eachCell.getG6() + "    ");
            System.out.print(eachCell.getG7());
            System.out.println();
        }
    }

    public void saveCellData(Config config) throws FileNotFoundException,
UnsupportedEncodingException {
        String cellFname = config.getDirName() + "/" + config.getSetName() +
"_cell_setting.txt";
        System.out.print("Saving cell data.....");

        try (PrintWriter file = new PrintWriter(cellFname, "UTF-8")) {
            for (int i = 0; i < numOfCells; i++) {
                file.print(this.get(i).getID() + " ");
                file.print(this.get(i).getG0() + " ");
                file.print(this.get(i).getG1() + " ");
                file.print(this.get(i).getG2() + " ");
                file.print(this.get(i).getG3() + " ");
                file.print(this.get(i).getG4() + " ");
                file.print(this.get(i).getG5() + " ");
                file.print(this.get(i).getG6() + " ");
                file.print(this.get(i).getG7());
                file.println();
            }
        }

        System.out.println("done.");
        System.out.println("--------------------------------");
    }
}
```

**cps3dn > Config.java**

```
package cps3dn;

import constant.Constant;
```

**cps3dn > Config.java**

```
import crossSection.DefaultCrossSection;
import interaction.DefaultInteraction;
import java.io.FileNotFoundException;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;
import material.DefaultMaterial;
import material.Material;

/**
 *
 * @author ppengvan
 */
public class Config {
    private final double tol;
    private final double dtol;
    private final double w;
    private final String dirName;
    private final String setName;
    private final double dt;
    private final int cycle;
    private final double xi;
    private final double xf;
    private final double yi;
    private final double yf;
    private final double zi;
    private final double zf;
    private final double Vi;
    private final double Vf;
    private final double Vw;
    private final double Vt;
    private final double Bx;
    private final double By;
    private final double Bz;
    private final double gridSizeX;
```

**cps3dn > Config.java**

```
    private final double gridSizeY;
    private final double gridSizeZ;
    private final int numOfX;
    private final int numOfY;
    private final int numOfZ;
    private final double no;
    private final int nType;      // number of particle types
    private final List <DefaultMaterial> parType = new ArrayList<>();
    private final List <Integer> numOfPars = new ArrayList<>();
//    private final List <Double> weight = new ArrayList<>();
    private final List <Double> charge = new ArrayList<>();
    private final List <Double> mass = new ArrayList<>();
    private final List <Double> vxavg = new ArrayList<>();
    private final List <Double> vxstd = new ArrayList<>();
    private final List <Double> vyavg = new ArrayList<>();
    private final List <Double> vystd = new ArrayList<>();
    private final List <Double> vzavg = new ArrayList<>();
    private final List <Double> vzstd = new ArrayList<>();
    private final int inType;
    private final List <Integer> inc = new ArrayList<>();
    private final List <Integer> tar = new ArrayList<>();
    private final List <DefaultInteraction> interaction = new ArrayList<>();
    private final List <DefaultCrossSection> xSection = new ArrayList<>();

    public Config() {
        tol = Input.TOL;              // absolute tolerence for residual (for field solver)
        dtol = Input.DTOL;            // tolerence for residual rate of change
        w = Input.W;                  // relaxation constant
        dirName = Input.DIR_NAME;     // name of data folder
        setName = Input.SET_NAME;
        dt = Input.DT;
        cycle = Input.CYCLE;
        xi = Input.X_I;
        xf = Input.X_F;
        yi = Input.Y_I;
        yf = Input.Y_F;
        zi = Input.Z_I;
```

**cps3dn > Config.java**

```
        zf = Input.Z_F;
        numOfX = Input.NUM_OF_GRID_X;
        numOfY = Input.NUM_OF_GRID_Y;
        numOfZ = Input.NUM_OF_GRID_Z;
        Vi = Input.VOLTAGE_I;
        Vf = Input.VOLTAGE_F;
        Vw = Input.VOLTAGE_W;
        Vt = Input.VOLTAGE_T;
        Bx = Input.B_X;
        By = Input.B_Y;
        Bz = Input.B_Z;
        gridSizeX = (Input.X_F - Input.X_I)/(Input.NUM_OF_GRID_X-1);
        gridSizeY = (Input.Y_F - Input.Y_I)/(Input.NUM_OF_GRID_Y-1);
        gridSizeZ = (Input.Z_F - Input.Z_I)/(Input.NUM_OF_GRID_Z-1);
        no = Input.REFERENCE_DENSITY;
        nType = Input.NUM_OF_PARTICLE_TYPE;
        for (int i = 0; i < Input.NUM_OF_PARTICLE_TYPE; i++) {
            parType.add(Input.PARTICLE_CLASS[i]);
            numOfPars.add(Input.NUM_OF_PARTICLE[i]);
//              config.setWeight(DefaultInput.PARTICLE_WEIGHT[i]);
            charge.add(Material.getMaterial(Input.PARTICLE_CLASS[i]).getCharge());
            mass.add(Material.getMaterial(Input.PARTICLE_CLASS[i]).getMass());
            vxavg.add(Input.V_X_AVERAGE[i]);
            vxstd.add(Input.V_X_STANDARD_DEVIATION[i]);
            vyavg.add(Input.V_Y_AVERAGE[i]);
            vystd.add(Input.V_Y_STANDARD_DEVIATION[i]);
            vzavg.add(Input.V_Z_AVERAGE[i]);
            vzstd.add(Input.V_Z_STANDARD_DEVIATION[i]);
        }
//        weight =
(Constant.EPS*Constant.ME*Input.PLASMA_FREQUENCY*Input.PLASMA_FREQUENCY)/(Const
ant.QE*Constant.QE)/Input.NUM_OF_PARTICLE[0];
        inType = Input.INTERACTION_TYPE;
        for (int i = 0; i < Input.INTERACTION_TYPE; i++) {
            inc.add(Input.INCOMING[i]);
            tar.add(Input.TARGET[i]);
            interaction.add(Input.INTERACTION[i]);
```

**cps3dn > Config.java**

```
            xSection.add(Input.CROSS_SECTION[i]);
        }
    }

    public double getTol() {
        return tol;
    }
    public double getDTol() {
        return dtol;
    }
    public double getW() {
        return w;
    }
    public String getDirName() {
        return dirName;
    }
    public String getSetName() {
        return setName;
    }
    public double getDT() {
        return dt;
    }
    public int getCycle() {
        return cycle;
    }
    public double getXi() {
        return xi;
    }
    public double getXf() {
        return xf;
    }
    public double getYi() {
        return yi;
    }
    public double getYf() {
        return yf;
    }
```

**cps3dn > Config.java**

```
    public double getZi() {
        return zi;
    }
    public double getZf() {
        return zf;
    }
    public double getVi() {
        return Vi;
    }
    public double getVf() {
        return Vf;
    }
    public double getVw() {
        return Vw;
    }
    public double getVt() {
        return Vt;
    }
    public double getBx() {
        return Bx;
    }
    public double getBy() {
        return By;
    }
    public double getBz() {
        return Bz;
    }
    public double getGridSizeX() {
        return gridSizeX;
    }
    public double getGridSizeY() {
        return gridSizeY;
    }
    public double getGridSizeZ() {
        return gridSizeZ;
    }
    public int getNumOfX() {
```

**cps3dn > Config.java**

```
        return numOfX;
    }
    public int getNumOfY() {
        return numOfY;
    }
    public int getNumOfZ() {
        return numOfZ;
    }
    public double getNo() {
        return no;
    }
    public int getNType() {
        return nType;
    }
    public DefaultMaterial getParType(int num) {
        return parType.get(num);
    }
    public int getNumOfPars(int num) {
        return numOfPars.get(num);
    }
//    public double getWeight(int num) {
//        return weight.get(num);
//    }
    public double getCharge(int num) {
        return charge.get(num);
    }
    public double getMass(int num) {
        return mass.get(num);
    }
    public double getVXAvg(int num) {
        return vxavg.get(num);
    }
    public double getVXStd(int num) {
        return vxstd.get(num);
    }
    public double getVYAvg(int num) {
        return vyavg.get(num);
```

**cps3dn > Config.java**

```
    }
    public double getVYStd(int num) {
        return vystd.get(num);
    }
    public double getVZAvg(int num) {
        return vzavg.get(num);
    }
    public double getVZStd(int num) {
        return vzstd.get(num);
    }
    public int getInType() {
        return inType;
    }
    public int getInc(int num) {
        return inc.get(num);
    }
    public int getTar(int num) {
        return tar.get(num);
    }
    public DefaultInteraction getInteraction(int num) {
        return interaction.get(num);
    }
    public DefaultCrossSection getXSection(int num) {
        return xSection.get(num);
    }

    public void listConfig(PrintStream out) {
        out.println("Simulation Config");
        out.println("-----------------------------");
        out.println("dt = " + dt);
        out.println("Cycle = " + cycle);
        out.println("Xi = " + xi);
        out.println("Xf = " + xf);
        out.println("Yi = " + yi);
        out.println("Yf = " + yf);
        out.println("Zi = " + zi);
        out.println("Zf = " + zf);
```

**cps3dn > Config.java**

```
        out.println("Grid X = " + gridSizeX);
        out.println("Grid Y = " + gridSizeY);
        out.println("Grid Z = " + gridSizeZ);
        out.println("Number of X Grids = " + numOfX);
        out.println("Number of Y Grids = " + numOfY);
        out.println("Number of Z Grids = " + numOfZ);
        out.println("--------------------------------");
        out.println("Number of Charge Super-Particle Types = " + nType);
        for (int i = 0; i < nType; i++) {
            out.println("======= Type " + i + " =======");
            out.println("Type = " + parType.get(i).toString());
            out.println("Number of Super-Particles = " + numOfPars.get(i));
//              out.println("Weight = " + weight.get(i));
            out.println("Charge = " + charge.get(i));
            out.println("Mass = " + mass.get(i));
            out.println("Average Velocity in X = " + vxavg.get(i));
            out.println("STDIV Velocity in X = " + vxstd.get(i));
            out.println("Average Velocity in Y = " + vyavg.get(i));
            out.println("STDIV Velocity in Y = " + vystd.get(i));
            out.println("Average Velocity in Z = " + vzavg.get(i));
            out.println("STDIV Velocity in Z = " + vzstd.get(i));
        }
        out.println("--------------------------------");
        out.println("Number of Interaction = " + inType);
        for (int i = 0; i < inType; i++) {
            out.println("======= Type " + i + " =======");
            out.println("Incoming Particle ID = " + inc.get(i));
            out.println("Target Particle ID = " + tar.get(i));
            out.println("Interaction = " + interaction.get(i).toString());
            out.println("Cross Section = " + xSection.get(i).toString());
        }
        out.println("--------------------------------");
    }

    public void saveConfig() throws FileNotFoundException, UnsupportedEncodingException
{
        String configFname = this.getDirName() + "/" + this.getSetName() + "_config.txt";
```

**cps3dn > Config.java**

```
        System.out.print("Saving config data.....");

        try (PrintWriter file = new PrintWriter(configFname, "UTF-8")) {
            file.println("tol = " + tol);
            file.println("dtol = " + dtol);
            file.println("w = " + w);
            file.println("dt = " + dt);
            file.println("Cycle = " + cycle);
            file.println("======================");
            file.println("Xi = " + xi);
            file.println("Xf = " + xf);
            file.println("Yi = " + yi);
            file.println("Yf = " + yf);
            file.println("Zi = " + zi);
            file.println("Zf = " + zf);
            file.println("Vi = " + Vi);
            file.println("Vf = " + Vf);
            file.println("Vw = " + Vw);
            file.println("Vt = " + Vt);
            file.println("Bx = " + Bx);
            file.println("By = " + By);
            file.println("Bz = " + Bz);
            file.println("Grid Size X = " + gridSizeX);
            file.println("Grid Size Y = " + gridSizeY);
            file.println("Grid Size Z = " + gridSizeZ);
            file.println("Number of X Grids = " + numOfX);
            file.println("Number of Y Grids = " + numOfY);
            file.println("Number of Z Grids = " + numOfZ);
            file.println("======================");
            file.println("Reference Density = " + no);
            file.println("Number of Charge Super-Particle Types = " + nType);
            for (int i = 0; i < nType; i++) {
                file.println("======= Type " + i + " =======");
                file.println("Type = " + parType.get(i).toString());
                file.println("Number of Super-Particles = " + numOfPars.get(i));
    //            out.println("Weight = " + weight.get(i));
                file.println("Charge = " + charge.get(i));
```

**cps3dn > Config.java**

```
                file.println("Mass = " + mass.get(i));
                file.println("Average Velocity in X = " + vxavg.get(i));
                file.println("STDIV Velocity in X = " + vxstd.get(i));
                file.println("Average Velocity in Y = " + vyavg.get(i));
                file.println("STDIV Velocity in Y = " + vystd.get(i));
                file.println("Average Velocity in Z = " + vzavg.get(i));
                file.println("STDIV Velocity in Z = " + vzstd.get(i));
            }
            file.println("======================");
            file.println("Number of Interaction = " + inType);
            for (int i = 0; i < inType; i++) {
                file.println("======= Type " + i + " =======");
                file.println("Incoming Particle ID = " + inc.get(i));
                file.println("Target Particle ID = " + tar.get(i));
                file.println("Interaction = " + interaction.get(i).toString());
                file.println("Cross Section = " + xSection.get(i).toString());
            }
        }
    }
}
```

**cps3dn > Energy.java**

```
package cps3dn;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author ppengvan
 */
```

**cps3dn > Energy.java**

```
public class Energy {
    private double totalKE;
    private double totalPE;

    public void calculateTotalKE(Config config, Particles pars) {
        totalKE = 0;
        for (Particle p : pars.get()) {
            if (p.getCell() != -1) {
//                totalKE = totalKE + p.getKE()*config.getWeight(p.getTypeID());
//                totalKE = totalKE + (0.5 * config.getMass(p.getTypeID()) * 0.5 * ((p.getVXP() *
p.getVXP()) + (p.getVX() * p.getVX()))*config.getWeight(p.getTypeID()) +
//                                    (0.5 * config.getMass(p.getTypeID()) * 0.5 * ((p.getVYP() *
p.getVYP()) + (p.getVY() * p.getVY()))*config.getWeight(p.getTypeID()) +
//                                    (0.5 * config.getMass(p.getTypeID()) * 0.5 * ((p.getVZP() *
p.getVZP()) + (p.getVZ() * p.getVZ()))*config.getWeight(p.getTypeID());
                totalKE = totalKE + (0.5 * config.getMass(p.getTypeID()) * 0.5 * ((p.getVXP() *
p.getVXP()) + (p.getVX() * p.getVX())) +
                                    (0.5 * config.getMass(p.getTypeID()) * 0.5 * ((p.getVYP() *
p.getVYP()) + (p.getVY() * p.getVY()))) +
                                    (0.5 * config.getMass(p.getTypeID()) * 0.5 * ((p.getVZP() *
p.getVZP()) + (p.getVZ() * p.getVZ())));
            }
        }
    }

    public void calculateTotalPE(Config config, Grids grids) {
        totalPE = 0;

        double v = config.getGridSizeX()*config.getGridSizeY()*config.getGridSizeZ();
//        double rho;
        double numOfX = config.getNumOfX();
        double numOfY = config.getNumOfY();
        double numOfZ = config.getNumOfZ();

        double vol = 0;

        for (int gx = 0; gx < numOfX; gx++) {
```

**cps3dn > Energy.java**

```
            for (int gy = 0; gy < numOfY; gy++) {
                for (int gz = 0; gz < numOfZ; gz++) {
//                    rho = 0;
//                    for (int i = 0; i < config.getNType(); i++) {
//                        rho = rho + (config.getCharge(i) * (grids.get(gx,gy,gz).getN(i)));
//                    }
                    if (((gx == 0) || (gx == numOfX-1)) &&
                        ((gy == 0) || (gy == numOfY-1)) &&
                        ((gz == 0) || (gz == numOfZ-1))) {
//                        totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/8);
                        totalPE = totalPE + (0.5*(grids.get(gx, gy, gz).getEX()*grids.get(gx, gy,
gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy, gz).getEY() + grids.get(gx, gy,
gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/8.0);
                        vol = vol + v/8;
                    }
                    else if (((gx != 0) && (gx != numOfX-1)) &&
                            ((gy == 0) || (gy == numOfY-1)) &&
                            ((gz == 0) || (gz == numOfZ-1))) {
//                        totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/4);
                        totalPE = totalPE + (0.5*(grids.get(gx, gy, gz).getEX()*grids.get(gx, gy,
gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy, gz).getEY() + grids.get(gx, gy,
gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/4.0);
                        vol = vol + v/4;
                    }
                    else if (((gx == 0) || (gx == numOfX-1)) &&
                            ((gy != 0) && (gy != numOfY-1)) &&
                            ((gz == 0) || (gz == numOfZ-1))) {
//                        totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/4);
                        totalPE = totalPE + (0.5*(grids.get(gx, gy, gz).getEX()*grids.get(gx, gy,
gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy, gz).getEY() + grids.get(gx, gy,
gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/4.0);
```

**cps3dn > Energy.java**

```
                    vol = vol + v/4;
                }
                else if (((gx == 0) || (gx == numOfX-1)) &&
                        ((gy == 0) || (gy == numOfY-1)) &&
                        ((gz != 0) && (gz != numOfZ-1))) {
//                      totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/4);
                    totalPE = totalPE + (0.5*(grids.get(gx, gy, gz).getEX()*grids.get(gx, gy,
gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy, gz).getEY() + grids.get(gx, gy,
gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/4.0);
                    vol = vol+v/4;
                }
                else if (((gx != 0) && (gx != numOfX-1)) &&
                        ((gy != 0) && (gy != numOfY-1)) &&
                        ((gz != 0) && (gz != numOfZ-1))) {
//                      totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v);
                    totalPE = totalPE + (0.5*(grids.get(gx, gy, gz).getEX()*grids.get(gx, gy,
gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy, gz).getEY() + grids.get(gx, gy,
gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v);
                    vol = vol + v;
                }
                else {
//                      totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/2);
                    totalPE = totalPE + (0.5*(grids.get(gx, gy, gz).getEX()*grids.get(gx, gy,
gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy, gz).getEY() + grids.get(gx, gy,
gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/2.0);
                    vol = vol + v/2;
                }
            }
        }
    }
```

**cps3dn > Energy.java**

```
        System.out.println("Total integrated volume for calculating total potential energy = "
+ vol);
//        System.out.println("total: "+i+" "+j+" "+k+" "+l);
    }

    public double getTotalKE() {
        return totalKE;
    }

    public double getTotalPE() {
        return totalPE;
    }

    public void saveEnergyData(Config config, String slide) throws FileNotFoundException,
UnsupportedEncodingException {
        String energyFname = config.getDirName() + "/" + config.getSetName() + "_energy.txt";

        System.out.print("Saving energy data from cycle " + slide + ".....");

        try (PrintWriter file = new PrintWriter(new FileOutputStream(
                new File(energyFname),
                true /* append = true */))) {
            file.print(slide + " ");
            file.print((config.getDT()*Integer.valueOf(slide)) + " ");
            file.print(totalKE + " ");
            file.print(totalPE + " ");
            file.print((totalKE+totalPE));
            file.println();
        }

        System.out.println("done.");
        System.out.println("--------------------------------");
    }
}
```

**cps3dn > Grid.java**

```
package cps3dn;

import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author ppengvan
 */
public class Grid {
    private int id;
    private double x;       // location in x
    private double y;       // location in y
    private double z;       // location in z
    private double V;       // Voltage
    private double ex;      // E field in x direction
    private double ey;      // E field in y direction
    private double ez;      // E field in z direction
    private double bx;      // B field in x direction
    private double by;      // B field in y direction
    private double bz;      // B field in z direction

    List <Double> n = new ArrayList<>();    //

    public Grid(int num) {
        for (int i = 0; i < num; i++) {
            n.add(0.0);
        }
    }
    public void setID(int idVal) {
        id = idVal;
    }
    public void setX(double xVal) {
        x = xVal;
    }
    public void setY(double yVal) {
        y = yVal;
```

**cps3dn > Grid.java**

```
    }
    public void setZ(double zVal) {
        z = zVal;
    }
    public void setV(double vVal) {
        V = vVal;
    }
    public void setEX(double exVal) {
        ex = exVal;
    }
    public void setEY(double eyVal) {
        ey = eyVal;
    }
    public void setEZ(double ezVal) {
        ez = ezVal;
    }
    public void setBX(double bxVal) {
        bx = bxVal;
    }
    public void setBY(double byVal) {
        by = byVal;
    }
    public void setBZ(double bzVal) {
        bz = bzVal;
    }
    public void setN(int i, double nVal) {
        n.set(i,nVal);
    }

    public int getID() {
        return id;
    }
    public double getX() {
        return x;
    }
    public double getY() {
        return y;
```

**cps3dn > Grid.java**

```
    }
    public double getZ() {
        return z;
    }
    public double getV() {
        return V;
    }
    public double getEX() {
        return ex;
    }
    public double getEY() {
        return ey;
    }
    public double getEZ() {
        return ez;
    }
    public double getBX() {
        return bx;
    }
    public double getBY() {
        return by;
    }
    public double getBZ() {
        return bz;
    }
    public double getN(int i) {
        return n.get(i);
    }
}
```

**cps3dn > Grids.java**

```
package cps3dn;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
```

**cps3dn > Grids.java**

```
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

/**
 *
 * @author ppengvan
 */
public class Grids {
    private Grid[][][] grids;
    private int numOfX;
    private int numOfY;
    private int numOfZ;
    private int numOfGrids;

    public Grids(Config config) {
    // Create grids, set boundary voltage, get neighboring grids, initialize some grid values
        int i = 0;
        int gx;
        int gy;
        int gz;
        double xc = config.getXi();
        double yc;
        double zc;
        grids = new Grid[config.getNumOfX()][config.getNumOfY()][config.getNumOfZ()];
        numOfX = config.getNumOfX();
        numOfY = config.getNumOfY();
        numOfZ = config.getNumOfZ();

        System.out.print("Creating grids.....");

        for (gx = 0; gx < numOfX; gx++) {
            yc = config.getYi();

            for (gy = 0; gy < numOfY; gy++) {
                zc = config.getZi();

                for (gz = 0; gz < numOfZ; gz++) {
```

cps3dn > Grids.java

```
                Grid grd = new Grid(config.getNType());
                grd.setID(i);
                grd.setX(xc);
                grd.setY(yc);
                grd.setZ(zc);
                this.set(grd,gx,gy,gz);

                i = i + 1;

                zc = zc + config.getGridSizeZ();
            }

            yc = yc + config.getGridSizeY();
        }

        xc = xc + config.getGridSizeX();
    }

    numOfGrids = i;

    for (gx = 0; gx < numOfX; gx++) {
        for (gy = 0; gy < numOfY; gy++) {
            for (gz = 0; gz < numOfZ; gz++) {
                if (gy == 0) {
                    this.get(gx,gy,gz).setV(config.getVi());
                }
                else if (gy == (numOfY - 1)) {
                    this.get(gx,gy,gz).setV(config.getVf()*Math.cos(config.getVt()));
                }
                else {
                    this.get(gx,gy,gz).setV(0);
                }

                this.get(gx,gy,gz).setEX(0);
                this.get(gx,gy,gz).setEY(0);
                this.get(gx,gy,gz).setEZ(0);
                this.get(gx,gy,gz).setBZ(config.getBz());
```

**cps3dn > Grids.java**

```
                }
            }
        }
        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public void set(Grid grd, int i, int j, int k) {
        grids[i][j][k] = grd;
    }

    public Grid get(int id) {
        int x = id / (numOfY * numOfZ);
        int y = (id % (numOfY * numOfZ)) / numOfZ;
        int z = (id % (numOfY * numOfZ)) % numOfZ;
        return grids[x][y][z];
    }

    public Grid get(int i, int j, int k) {
        return grids[i][j][k];
    }

    public void updateDensity(Config config, Cells cells, Particles pars) {
    // Calculate density at each grid
        double tv, v, v0, v1, v2, v3, v4, v5, v6, v7;
        double x, y, z;
        int c;
        int typeID;
        int g0, g1, g2, g3, g4, g5, g6, g7;
        double tpar = 0;

        System.out.print("Calculating particles .....");

        // total volume
        tv = (config.getXf()-config.getXi())*(config.getYf()-config.getYi())*(config.getZf()-
config.getZi());
        // differential volume
```

**cps3dn > Grids.java**

```
        v = config.getGridSizeX()*config.getGridSizeY()*config.getGridSizeZ();

        for (int gx = 0; gx < numOfX; gx++) {       // Reset density of each grid
            for (int gy = 0; gy < numOfY; gy++) {
                for (int gz = 0; gz < numOfZ; gz++) {
                    for (int i = 0; i < config.getNType(); i++) {
                        this.get(gx,gy,gz).setN(i, 0.0);
                    }
                }
            }
        }
        for (Particle eachPar : pars.get()) {
            if (eachPar.getCell() != -1) {
                c = eachPar.getCell();
                typeID = eachPar.getTypeID();
                x = eachPar.getX();
                y = eachPar.getY();
                z = eachPar.getZ();
                g0 = cells.get(c).getG0();
                g1 = cells.get(c).getG1();
                g2 = cells.get(c).getG2();
                g3 = cells.get(c).getG3();
                g4 = cells.get(c).getG4();
                g5 = cells.get(c).getG5();
                g6 = cells.get(c).getG6();
                g7 = cells.get(c).getG7();
                v0 = (x-this.get(g0).getX()) * (y-this.get(g0).getY()) * (z-this.get(g0).getZ());
                v1 = (x-this.get(g1).getX()) * (y-this.get(g1).getY()) * (this.get(g1).getZ()-z);
                v2 = (x-this.get(g2).getX()) * (this.get(g2).getY()-y) * (z-this.get(g2).getZ());
                v3 = (x-this.get(g3).getX()) * (this.get(g3).getY()-y) * (this.get(g3).getZ()-z);
                v4 = (this.get(g4).getX()-x) * (y-this.get(g4).getY()) * (z-this.get(g4).getZ());
                v5 = (this.get(g5).getX()-x) * (y-this.get(g5).getY()) * (this.get(g5).getZ()-z);
                v6 = (this.get(g6).getX()-x) * (this.get(g6).getY()-y) * (z-this.get(g6).getZ());
                v7 = (this.get(g7).getX()-x) * (this.get(g7).getY()-y) * (this.get(g7).getZ()-z);

//                this.get(g0).setN(typeID,
this.get(g0).getN(typeID)+config.getWeight(typeID)*v7/v);
```

cps3dn > Grids.java

```
//                this.get(g1).setN(typeID,
this.get(g1).getN(typeID)+config.getWeight(typeID)*v6/v);
//                this.get(g2).setN(typeID,
this.get(g2).getN(typeID)+config.getWeight(typeID)*v5/v);
//                this.get(g3).setN(typeID,
this.get(g3).getN(typeID)+config.getWeight(typeID)*v4/v);
//                this.get(g4).setN(typeID,
this.get(g4).getN(typeID)+config.getWeight(typeID)*v3/v);
//                this.get(g5).setN(typeID,
this.get(g5).getN(typeID)+config.getWeight(typeID)*v2/v);
//                this.get(g6).setN(typeID,
this.get(g6).getN(typeID)+config.getWeight(typeID)*v1/v);
//                this.get(g7).setN(typeID,
this.get(g7).getN(typeID)+config.getWeight(typeID)*v0/v);

                this.get(g0).setN(typeID, this.get(g0).getN(typeID)+v7/v);
                this.get(g1).setN(typeID, this.get(g1).getN(typeID)+v6/v);
                this.get(g2).setN(typeID, this.get(g2).getN(typeID)+v5/v);
                this.get(g3).setN(typeID, this.get(g3).getN(typeID)+v4/v);
                this.get(g4).setN(typeID, this.get(g4).getN(typeID)+v3/v);
                this.get(g5).setN(typeID, this.get(g5).getN(typeID)+v2/v);
                this.get(g6).setN(typeID, this.get(g6).getN(typeID)+v1/v);
                this.get(g7).setN(typeID, this.get(g7).getN(typeID)+v0/v);
            }
        }

//        // Check if number of super particle is still conserved
//        double nxx = 0;
//        for (int i = 0; i < this.getNumOfGrids(); i++) {
//
//            nxx = nxx + this.get(i).getN(0);
//        }
//        System.out.println();
//        System.out.println("total number == " + nxx/config.getWeight(0));

        // Also add particle contribution to the boundary cell on opposite side to handle
continuity
```

cps3dn > Grids.java

```
        for (int i = 0; i < config.getNType(); i++) {
            double temp;

            for (int gx = 0; gx < numOfX; gx++) {
                for (int gy = 0; gy < numOfY; gy++) {
                    temp = this.get(gx,gy,0).getN(i) + this.get(gx,gy,numOfZ-1).getN(i);
                    this.get(gx,gy,0).setN(i, temp);
                    this.get(gx,gy,numOfZ-1).setN(i, temp);
                }
            }
            for (int gx = 0; gx < numOfX; gx++) {
                for (int gz = 0; gz < numOfZ; gz++) {
                    temp = this.get(gx,0,gz).getN(i) + this.get(gx,numOfY-1,gz).getN(i);
                    this.get(gx,0,gz).setN(i, temp);
                    this.get(gx,numOfY-1,gz).setN(i, temp);
                }
            }
            for (int gy = 0; gy < numOfY; gy++) {
                for (int gz = 0; gz < numOfZ; gz++) {
                    temp = this.get(0,gy,gz).getN(i) + this.get(numOfX-1,gy,gz).getN(i);
                    this.get(0,gy,gz).setN(i, temp);
                    this.get(numOfX-1,gy,gz).setN(i, temp);
                }
            }
        }

        // Calculate particle density
        for (int gx = 0; gx < numOfX; gx++) {
            for (int gy = 0; gy < numOfY; gy++) {
                for (int gz = 0; gz < numOfZ; gz++) {
                    for (int i = 0; i < config.getNType(); i++) {
                        this.get(gx,gy,gz).setN(i, this.get(gx,gy,gz).getN(i)/v);
                    }
                }
            }
        }
```

**cps3dn > Grids.java**

```
        // Calculate total number of particles
        for (int gx = 0; gx < numOfX; gx++) {
            for (int gy = 0; gy < numOfY; gy++) {
                for (int gz = 0; gz < numOfZ; gz++) {
                    for (int i = 0; i < config.getNType(); i++) {
                        if (((gx == 0) || (gx == numOfX-1)) &&
                            ((gy == 0) || (gy == numOfY-1)) &&
                            ((gz == 0) || (gz == numOfZ-1))) {
                            tpar = tpar + this.get(gx,gy,gz).getN(i)*v/8.0;
                        }
                        else if (((gx != 0) && (gx != numOfX-1)) &&
                                 ((gy == 0) || (gy == numOfY-1)) &&
                                 ((gz == 0) || (gz == numOfZ-1))) {
                            tpar = tpar + this.get(gx,gy,gz).getN(i)*v/4.0;
                        }
                        else if (((gx == 0) || (gx == numOfX-1)) &&
                                 ((gy != 0) && (gy != numOfY-1)) &&
                                 ((gz == 0) || (gz == numOfZ-1))) {
                            tpar = tpar + this.get(gx,gy,gz).getN(i)*v/4.0;
                        }
                        else if (((gx == 0) || (gx == numOfX-1)) &&
                                 ((gy == 0) || (gy == numOfY-1)) &&
                                 ((gz != 0) && (gz != numOfZ-1))) {
                            tpar = tpar + this.get(gx,gy,gz).getN(i)*v/4.0;
                        }
                        else if (((gx != 0) && (gx != numOfX-1)) &&
                                 ((gy != 0) && (gy != numOfY-1)) &&
                                 ((gz != 0) && (gz != numOfZ-1))) {
                            tpar = tpar + this.get(gx,gy,gz).getN(i)*v;
                        }
                        else {
                            tpar = tpar + this.get(gx,gy,gz).getN(i)*v/2.0;
                        }
//                          System.out.print(this.get(gx,gy,gz).getN(i) + " ");
                    }
//                  System.out.println();
                }
```

cps3dn > Grids.java

```
            }
        }

        System.out.println("done.");
        System.out.println("Total number of particles = " + tpar);
        System.out.println("--------------------------------");
    }

    public void updatePotentialNeumann(Config config, double time) {
    // Calculate potential at each grid, i.e. solving Poisson's equation
        double V = 0;
        double res = 0;
        double f = 0;
        double tres = 100;
        double ptres = 1;
        double dtres = 100;
        double dx = config.getGridSizeX();
        double dy = config.getGridSizeY();
        double dz = config.getGridSizeZ();
        int ct = 0;

        System.out.print("Calculating grid potential.....");

        for (int gx = 0; gx < numOfX; gx++) {
            for (int gy = 0; gy < numOfY; gy++) {
                for (int gz = 0; gz < numOfZ; gz++) {
                    if (gy == 0) {
                        this.get(gx,gy,gz).setV(config.getVi());
                    }
                    else if (gy == (numOfY - 1)) {

this.get(gx,gy,gz).setV(config.getVf()*Math.cos((config.getVw()*time)+config.getVt()));
                    }
                }
            }
        }
```

**cps3dn > Grids.java**

```
        while ((tres > config.getTol()) || (dtres > config.getDTol())) {
            ct = ct + 1;
            tres = 0;
//            for (int gy = 1; gy < numOfY-1; gy++) {
//                for (int gz = 1; gz < numOfZ-1; gz++) {
//                    double nV = 0.5*(this.get(0,gy,gz).getV()+this.get(numOfX-1,gy,gz).getV());
////                      System.out.println(this.get(0,gy,gz).getV() + " " + this.get(numOfX-
1,gy,gz).getV() + " " + nV);
//                    this.get(0,gy,gz).setV(nV);
//                    this.get(numOfX-1,gy,gz).setV(nV);
//                }
//            }
//            for (int gy = 1; gy < numOfY-1; gy++) {
//                for (int gx = 1; gx < numOfX-1; gx++) {
//                    double nV = 0.5*(this.get(gx,gy,0).getV()+this.get(gx,gy,numOfZ-1).getV());
//                    this.get(gx,gy,0).setV(nV);
//                    this.get(gx,gy,numOfZ-1).setV(nV);
//                }
//            }
//            for (int gy = 1; gy < numOfY-1; gy++) {
//                double nV = 0.25*(this.get(0,gy,0).getV()+this.get(0,gy,numOfZ-
1).getV()+this.get(numOfX-1,gy,0).getV()+this.get(numOfX-1,gy,numOfZ-1).getV());
//                this.get(0,gy,0).setV(nV);
//                this.get(0,gy,numOfZ-1).setV(nV);
//                this.get(numOfX-1,gy,0).setV(nV);
//                this.get(numOfX-1,gy,numOfZ-1).setV(nV);
//            }
            for (int gx = 0; gx < numOfX; gx++) {
                for (int gy = 0; gy < numOfY; gy++) {
                    for (int gz = 0; gz < numOfZ; gz++) {
                        if ((gy != 0) & (gy != (numOfY - 1))) {
                            f = 0;
                            for (int i = 0; i < config.getNType(); i++) {
//                                f = f +
config.getCharge(i)*this.get(gx,gy,gz).getN(i)/DefaultConstant.EPS;
                                f = f + config.getCharge(i)*this.get(gx,gy,gz).getN(i);
                            }
```

cps3dn > Grids.java

```
                        if (gx == 0) {
                            if (gz == 0) {
                                res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                      ((dy*dy*dz*dz)*(this.get(gx+1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                       (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                       (dx*dx*dy*dy)*(this.get(gx,gy,gz+1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                      2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx+1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz+1).getV() +
this.get(gx,gy,gz+1).getV()));
                            }
                            else if (gz == (numOfZ-1)) {
                                res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                      ((dy*dy*dz*dz)*(this.get(gx+1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                       (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                       (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() + this.get(gx,gy,gz-
1).getV()) -
                                      2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx+1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
```

cps3dn > Grids.java

```
                                (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() + this.get(gx,gy,gz-
1).getV()));
                        }
                        else {
                            res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                ((dy*dy*dz*dz)*(this.get(gx+1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                 (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                 (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                 2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                            V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                (f*(dx*dx*dy*dy*dz*dz) +
                                (dy*dy*dz*dz)*(this.get(gx+1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                        }
                    }
                    else if (gx == (numOfX-1)) {
                        if (gz == 0) {
                            res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() + this.get(gx-
1,gy,gz).getV()) +
                                 (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                 (dx*dx*dy*dy)*(this.get(gx,gy,gz+1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                 2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                            V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                (f*(dx*dx*dy*dy*dz*dz) +
```

cps3dn > Grids.java

```
                                (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() + this.get(gx-
1,gy,gz).getV()) +
                                (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                (dx*dx*dy*dy)*(this.get(gx,gy,gz+1).getV() +
this.get(gx,gy,gz+1).getV()));
                        }
                        else if (gz == (numOfZ-1)) {
                            res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() + this.get(gx-
1,gy,gz).getV()) +
                                 (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                 (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() + this.get(gx,gy,gz-
1).getV()) -
                                 2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                            V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                (f*(dx*dx*dy*dy*dz*dz) +
                                (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() + this.get(gx-
1,gy,gz).getV()) +
                                (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() + this.get(gx,gy,gz-
1).getV()));
                        }
                        else {
                            res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() + this.get(gx-
1,gy,gz).getV()) +
                                 (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                 (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                 2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                            V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
```

cps3dn > Grids.java

```
                                (f*(dx*dx*dy*dy*dz*dz) +
                                (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() + this.get(gx-
1,gy,gz).getV()) +
                                (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                            }
                        }
                        else {
                            if (gz == 0) {
                                res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                    ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                     (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                     (dx*dx*dy*dy)*(this.get(gx,gy,gz+1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                     2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                (f*(dx*dx*dy*dy*dz*dz) +
                                (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                (dx*dx*dy*dy)*(this.get(gx,gy,gz+1).getV() +
this.get(gx,gy,gz+1).getV()));
                            }
                            else if (gz == (numOfZ-1)) {
                                res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                    ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                     (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                     (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() + this.get(gx,gy,gz-
1).getV()) -
```

cps3dn > Grids.java

```
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() + this.get(gx,gy,gz-
1).getV()));
                                }
                                else {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                         (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                         (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                         2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                            }

                            this.get(gx,gy,gz).setV((1-
config.getW())*this.get(gx,gy,gz).getV()+config.getW()*V);
                        }

                        tres = tres + res*res;
```

**cps3dn > Grids.java**

```
                }
            }
        }

        tres = Math.sqrt(tres);
        dtres = Math.abs(tres - ptres)/ptres;
        ptres = tres;
        if (ct%1000 == 0) {
            System.out.println("Iternation #: " + ct);
            System.out.println("Total residual = " + tres);
            System.out.println("Differential total residual = " + dtres);
        }
    }

    System.out.println("done.");
    System.out.println("Number of iterations: " + ct);
    System.out.println("Total residual = " + tres);
//      System.out.println("Differential total residual = " + dtres);
    System.out.println("--------------------------------");
  }

  public void updatePotentialPeriodic(Config config, double time) {
  // Calculate potential at each grid, i.e. solving Poisson's equation
    double V = 0;
    double res = 0;
    double f = 0;
    double tres = 100;
    double ptres = 1;
    double dtres = 100;
    double dx = config.getGridSizeX();
    double dy = config.getGridSizeY();
    double dz = config.getGridSizeZ();
    int ct = 0;

    System.out.print("Calculating grid potential.....");

//      for (int gx = 0; gx < numOfX; gx++) {
```

cps3dn > Grids.java

```
//            for (int gy = 0; gy < numOfY; gy++) {
//                for (int gz = 0; gz < numOfZ; gz++) {
//                    if (gy == 0) {
//                        this.get(gx,gy,gz).setV(config.getVi());
//                    }
//                    else if (gy == (numOfY - 1)) {
//
this.get(gx,gy,gz).setV(config.getVf()*Math.cos((config.getVw()*time)+config.getVt()));
//                    }
//                }
//            }
//        }

//        int tct = 0;

        while ((tres > config.getTol()) || (dtres > config.getDTol())) {
            ct = ct + 1;
            tres = 0;
            for (int gx = 0; gx < numOfX; gx++) {
                for (int gy = 0; gy < numOfY; gy++) {
                    for (int gz = 0; gz < numOfZ; gz++) {
//                        if ((gy != 0) && (gy != (numOfY - 1))) {
                            f = 0;
                            for (int i = 0; i < config.getNType(); i++) {
//                                f = f +
config.getCharge(i)*this.get(gx,gy,gz).getN(i)/DefaultConstant.EPS;
                                f = f + config.getCharge(i)*this.get(gx,gy,gz).getN(i);
                            }
                            if (gx == 0) {
                                if (gy == 0) {
                                    if (gz == 0) {
                                        res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                            ((dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                            (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
```

cps3dn > Grids.java

```
                                    (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()) -
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                                else if (gz == (numOfZ-1)) {
//                                    V = this.get(gx, gy, 0).getV();
//                                    tct++;
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()));
                                }
                                else {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
```

cps3dn > Grids.java

```
                                    ((dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                            }
                            else if (gy == (numOfY-1)) {
//                                V = this.get(gx, 0, gz).getV();
//                                tct++;
                                if (gz == 0) {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                    ((dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()) -
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
```

cps3dn > Grids.java

```
                                    (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()));
                            }
                            else if (gz == (numOfZ-1)) {
                                res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                    ((dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()) -
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()));
                            }
                            else {
                                res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                    ((dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
```

cps3dn > Grids.java

```
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                            }
                            else {
                                if (gz == 0) {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                    V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                        (f*(dx*dx*dy*dy*dz*dz) +
                                        (dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                                else if (gz == (numOfZ-1)) {
//                                    V = this.get(gx, gy, 0).getV();
//                                    tct++;
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()) -
```

cps3dn > Grids.java

```
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()));
                                }
                                else {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(numOfX-2,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                            }
                        }
                        else if (gx == (numOfX-1)) {
//                          V = this.get(0, gy, gz).getV();
//                          tct++;
                            if (gy == 0) {
                                if (gz == 0) {
```

cps3dn > Grids.java

```
                                res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                    ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()) -
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()));
                            }
                            else if (gz == (numOfZ-1)) {
                                res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                    ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()) -
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()));
                            }
```

cps3dn > Grids.java

```
                                else {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                    V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                        (f*(dx*dx*dy*dy*dz*dz) +
                                        (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                                }

                            }
                            else if (gy == (numOfY-1)) {
                                if (gz == 0) {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                    V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                        (f*(dx*dx*dy*dy*dz*dz) +
                                        (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
```

cps3dn > Grids.java

```
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                                else if (gz == (numOfZ-1)) {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                    V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                        (f*(dx*dx*dy*dy*dz*dz) +
                                        (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()));
                                }
                                else {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                    V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                        (f*(dx*dx*dy*dy*dz*dz) +
```

cps3dn > Grids.java

```
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                            }
                            else {
                                if (gz == 0) {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                    V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                        (f*(dx*dx*dy*dy*dz*dz) +
                                        (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                                else if (gz == (numOfZ-1)) {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()) -
```

cps3dn > Grids.java

```
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()));
                                }
                                else {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                            }
                        }
                        else {
                            if (gy == 0) {
                                if (gz == 0) {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
```

cps3dn > Grids.java

```
                                    ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()) -
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                                else if (gz == (numOfZ-1)) {
//                                    V = this.get(gx, gy, 0).getV();
//                                    tct++;
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                    ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()) -
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()));
```

cps3dn > Grids.java

```
                                }
                                else {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                    V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                        (f*(dx*dx*dy*dy*dz*dz) +
                                        (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,numOfY-2,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                            }
                            else if (gy == (numOfY-1)) {
//                                V = this.get(gx, 0, gz).getV();
//                                tct++;
                                if (gz == 0) {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                    V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                        (f*(dx*dx*dy*dy*dz*dz) +
```

cps3dn > Grids.java

```
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                                else if (gz == (numOfZ-1)) {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                    V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                        (f*(dx*dx*dy*dy*dz*dz) +
                                        (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()));
                                }
                                else {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                    V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
```

cps3dn > Grids.java

```
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                            }
                            else {
                                if (gz == 0) {
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                    V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                        (f*(dx*dx*dy*dy*dz*dz) +
                                        (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                        (dx*dx*dy*dy)*(this.get(gx,gy,numOfZ-2).getV() +
this.get(gx,gy,gz+1).getV()));
                                }
                                else if (gz == (numOfZ-1)) {
//                                    V = this.get(gx, gy, 0).getV();
//                                    tct++;
                                    res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                        ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                        (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
```

cps3dn > Grids.java

```
                                        (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()) -
                                        2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,1).getV()));
                            }
                            else {
                                res = f + (1.0/(dx*dx*dy*dy*dz*dz)) *
                                    ((dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()) -
                                    2.0*(dy*dy*dz*dz + dx*dx*dz*dz +
dx*dx*dy*dy)*this.get(gx,gy,gz).getV());
                                V = (1.0/(2.0*(dy*dy*dz*dz + dx*dx*dz*dz + dx*dx*dy*dy))) *
                                    (f*(dx*dx*dy*dy*dz*dz) +
                                    (dy*dy*dz*dz)*(this.get(gx-1,gy,gz).getV() +
this.get(gx+1,gy,gz).getV()) +
                                    (dx*dx*dz*dz)*(this.get(gx,gy-1,gz).getV() +
this.get(gx,gy+1,gz).getV()) +
                                    (dx*dx*dy*dy)*(this.get(gx,gy,gz-1).getV() +
this.get(gx,gy,gz+1).getV()));
                            }
                        }
                    }

                    this.get(gx,gy,gz).setV((1-
config.getW())*this.get(gx,gy,gz).getV()+config.getW()*V);
```

**cps3dn > Grids.java**

```
//                      }

                        tres = tres + (res*res);
                    }
                }
            }

//            if (ct == 1) {
//                System.out.println("TCT = " + tct);
//            }

            tres = Math.sqrt(tres);
            dtres = Math.abs(tres - ptres)/ptres;
            ptres = tres;

//            System.out.println(tres + " " + dtres);
            if (ct%1000 == 0) {
                System.out.println("Iternation #: " + ct);
                System.out.println("Total residual = " + tres);
                System.out.println("Differential total residual = " + dtres);
            }
        }

        System.out.println("done.");
        System.out.println("Number of iterations: " + ct);
        System.out.println("Total residual = " + tres);
        System.out.println("Differential total residual = " + dtres);
        System.out.println("-------------------------------");
    }

    public void updateEFieldNeumann(Config config) {
    // Calculate electric field at each grid from the neighboring potentials
        double Ex = 0;
        double Ey = 0;
        double Ez = 0;

        System.out.print("Calculating grid electric field.....");
```

cps3dn > Grids.java

```
        for (int gx = 0; gx < numOfX; gx++) {
            for (int gy = 0; gy < numOfY; gy++) {
                for (int gz = 0; gz < numOfZ; gz++) {
                    if (gx == 0) {
                        Ex = -(this.get(gx+1,gy,gz).getV()-
this.get(gx+1,gy,gz).getV())/(2.0*config.getGridSizeX());
                    }
                    else if (gx == (numOfX - 1)) {
                        Ex = -(this.get(gx-1,gy,gz).getV()-this.get(gx-
1,gy,gz).getV())/(2.0*config.getGridSizeX());
                    }
                    else {
                        Ex = -(this.get(gx+1,gy,gz).getV()-this.get(gx-
1,gy,gz).getV())/(2.0*config.getGridSizeX());
                    }

                    if (gy == 0) {
                        Ey = -(this.get(gx,gy+1,gz).getV()-
this.get(gx,gy+1,gz).getV())/(2.0*config.getGridSizeY());
                    }
                    else if (gy == (numOfY - 1)) {
                        Ey = -(this.get(gx,gy-1,gz).getV()-this.get(gx,gy-
1,gz).getV())/(2.0*config.getGridSizeY());
                    }
                    else {
                        Ey = -(this.get(gx,gy+1,gz).getV()-this.get(gx,gy-
1,gz).getV())/(2.0*config.getGridSizeY());
                    }

                    if (gz == 0) {
                        Ez = -(this.get(gx,gy,gz+1).getV()-
this.get(gx,gy,gz+1).getV())/(2.0*config.getGridSizeZ());
                    }
                    else if (gz == (numOfZ - 1)) {
                        Ez = -(this.get(gx,gy,gz-1).getV()-this.get(gx,gy,gz-
1).getV())/(2.0*config.getGridSizeZ());
```

**cps3dn > Grids.java**

```
                }
                else {
                    Ez = -(this.get(gx,gy,gz+1).getV()-this.get(gx,gy,gz-
1).getV())/(2.0*config.getGridSizeZ());
                }

                this.get(gx,gy,gz).setEX(Ex);
                this.get(gx,gy,gz).setEY(Ey);
                this.get(gx,gy,gz).setEZ(Ez);
            }
        }
    }

    System.out.println("done.");
    System.out.println("--------------------------------");
  }


  public void updateEFieldPeriodic(Config config) {
  // Calculate electric field at each grid from the neighboring potentials
    double Ex = 0;
    double Ey = 0;
    double Ez = 0;

    System.out.print("Calculating grid electric field.....");

    for (int gx = 0; gx < numOfX; gx++) {
        for (int gy = 0; gy < numOfY; gy++) {
            for (int gz = 0; gz < numOfZ; gz++) {
                if (gx == 0) {
                    Ex = -(this.get(gx+1,gy,gz).getV()-this.get(numOfX-
2,gy,gz).getV())/(2.0*config.getGridSizeX());
                }
                else if (gx == (numOfX - 1)) {
                    Ex = -(this.get(1,gy,gz).getV()-this.get(gx-
1,gy,gz).getV())/(2.0*config.getGridSizeX());
                }
```

cps3dn > Grids.java

```
                else {
                    Ex = -(this.get(gx+1,gy,gz).getV()-this.get(gx-
1,gy,gz).getV())/(2.0*config.getGridSizeX());
                }

                if (gy == 0) {
                    Ey = -(this.get(gx,gy+1,gz).getV()-this.get(gx,numOfY-
2,gz).getV())/(2.0*config.getGridSizeY());
                }
                else if (gy == (numOfY - 1)) {
                    Ey = -(this.get(gx,1,gz).getV()-this.get(gx,gy-
1,gz).getV())/(2.0*config.getGridSizeY());
                }
                else {
                    Ey = -(this.get(gx,gy+1,gz).getV()-this.get(gx,gy-
1,gz).getV())/(2.0*config.getGridSizeY());
                }

                if (gz == 0) {
                    Ez = -(this.get(gx,gy,gz+1).getV()-this.get(gx,gy,numOfZ-
2).getV())/(2.0*config.getGridSizeZ());
                }
                else if (gz == (numOfZ - 1)) {
                    Ez = -(this.get(gx,gy,1).getV()-this.get(gx,gy,gz-
1).getV())/(2.0*config.getGridSizeZ());
                }
                else {
                    Ez = -(this.get(gx,gy,gz+1).getV()-this.get(gx,gy,gz-
1).getV())/(2.0*config.getGridSizeZ());
                }

                this.get(gx,gy,gz).setEX(Ex);
                this.get(gx,gy,gz).setEY(Ey);
                this.get(gx,gy,gz).setEZ(Ez);
            }
        }
    }
```

**cps3dn > Grids.java**

```
        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public int getNumOfGrids() {
        return numOfGrids;
    }

    public double getDensity(Config config, int type, double x, double y, double z) {
        int i = (int) ((x - config.getXi()) / config.getGridSizeX());
        int j = (int) ((y - config.getYi()) / config.getGridSizeY());
        int k = (int) ((z - config.getZi()) / config.getGridSizeZ());

        double v = config.getGridSizeX()*config.getGridSizeY()*config.getGridSizeZ();
        double v0 = (x-this.get(i,j,k).getX()) * (y-this.get(i,j,k).getY()) * (z-this.get(i,j,k).getZ());
        double v1 = (x-this.get(i,j,k+1).getX()) * (y-this.get(i,j,k+1).getY()) *
(this.get(i,j,k+1).getZ()-z);
        double v2 = (x-this.get(i,j+1,k).getX()) * (this.get(i,j+1,k).getY()-y) * (z-
this.get(i,j+1,k).getZ());
        double v3 = (x-this.get(i,j+1,k+1).getX()) * (this.get(i,j+1,k+1).getY()-y) *
(this.get(i,j+1,k+1).getZ()-z);
        double v4 = (this.get(i+1,j,k).getX()-x) * (y-this.get(i+1,j,k).getY()) * (z-
this.get(i+1,j,k).getZ());
        double v5 = (this.get(i+1,j,k+1).getX()-x) * (y-this.get(i+1,j,k+1).getY()) *
(this.get(i+1,j,k+1).getZ()-z);
        double v6 = (this.get(i+1,j+1,k).getX()-x) * (this.get(i+1,j+1,k).getY()-y) * (z-
this.get(i+1,j+1,k).getZ());
        double v7 = (this.get(i+1,j+1,k+1).getX()-x) * (this.get(i+1,j+1,k+1).getY()-y) *
(this.get(i+1,j+1,k+1).getZ()-z);

        double n = this.get(i,j,k).getN(type)*v7/v + this.get(i,j,k+1).getN(type)*v6/v +
                this.get(i,j+1,k).getN(type)*v5/v + this.get(i,j+1,k+1).getN(type)*v4/v +
                this.get(i+1,j,k).getN(type)*v3/v + this.get(i+1,j,k+1).getN(type)*v2/v +
                this.get(i+1,j+1,k).getN(type)*v1/v + this.get(i+1,j+1,k+1).getN(type)*v0/v;

        return n;
```

**cps3dn > Grids.java**

```
    }

    // List grid values
    public void listGrids(Config config) {
        System.out.println("Number of Grids = " + numOfGrids);
        System.out.println("Number of X = " + numOfX);
        System.out.println("Number of Y = " + numOfY);
        System.out.println("Number of Z = " + numOfZ);
        System.out.println("--------------------------------");
        for (int gx = 0; gx < numOfX; gx++) {
            for (int gy = 0; gy < numOfY; gy++) {
                for (int gz = 0; gz < numOfZ; gz++) {
                    System.out.print(this.get(gx,gy,gz).getID() + "    ");
                    System.out.print(this.get(gx,gy,gz).getX() + "    ");
                    System.out.print(this.get(gx,gy,gz).getY() + "    ");
                    System.out.print(this.get(gx,gy,gz).getZ() + "    ");
                    System.out.print(this.get(gx,gy,gz).getV() + "    ");
                    System.out.print(this.get(gx,gy,gz).getEX() + "    ");
                    System.out.print(this.get(gx,gy,gz).getEY() + "    ");
                    System.out.print(this.get(gx,gy,gz).getEZ() + "    ");
                    System.out.print(this.get(gx,gy,gz).getBX() + "    ");
                    System.out.print(this.get(gx,gy,gz).getBY() + "    ");
                    System.out.print(this.get(gx,gy,gz).getBZ() + "    ");
                    for (int i = 0; i < config.getNType(); i++) {
                        System.out.print(this.get(gx,gy,gz).getN(i) + "    ");
                    }
                    System.out.println();
                }
            }
        }
        System.out.println("--------------------------------");
    }

    public void saveGridData(Config config, String slide) throws FileNotFoundException,
UnsupportedEncodingException {
        String gridFname = config.getDirName() + "/" + config.getSetName() + "_grid_" + slide
+ ".txt";
```

**cps3dn > Grids.java**

```
        System.out.print("Saving grid data from cycle " + slide + ".....");

        try (PrintWriter file = new PrintWriter(gridFname, "UTF-8")) {
            file.println(slide);
            file.println(config.getNType());
            for (int i = 0; i < numOfGrids; i++) {
                file.print(this.get(i).getID() + " ");
                file.print(this.get(i).getX() + " ");
                file.print(this.get(i).getY() + " ");
                file.print(this.get(i).getZ() + " ");
                file.print(this.get(i).getV() + " ");
                file.print(this.get(i).getEX() + " ");
                file.print(this.get(i).getEY() + " ");
                file.print(this.get(i).getEZ() + " ");
                file.print(this.get(i).getBX() + " ");
                file.print(this.get(i).getBY() + " ");
                file.print(this.get(i).getBZ() + " ");
                for (int j = 0; j < config.getNType(); j++) {
                    file.print(this.get(i).getN(j) + " ");
                }
                file.println();
            }
        }

        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    // Save grid data at fixed X, Y, Z at all T
    public void saveTracedGrid(Config config, String slide, int x, int y, int z) throws
FileNotFoundException, UnsupportedEncodingException {
        String gridFname = config.getDirName() + "/" + config.getSetName() +
"_traced_grid.txt";

        try (PrintWriter file1 = new PrintWriter(new FileOutputStream(
                new File(gridFname),
                true /* append = true */))) {
```

**cps3dn > Grids.java**

```
            file1.print(slide + " ");
            file1.print(config.getDT() + " ");
            file1.print(this.get(x,y,z).getV() + " ");
            file1.print(this.get(x,y,z).getEX() + " ");
            file1.print(this.get(x,y,z).getEY() + " ");
            file1.print(this.get(x,y,z).getEZ());
            file1.println();
        }
    }

    // Save potential on XY plane at a certain time T and Z in excel format
    public void savePotentialZData4Excel(Config config, String slide, int z) throws
FileNotFoundException, UnsupportedEncodingException {
        String gridFname = config.getDirName() + "/" + config.getSetName() + "_V_s" + slide +
"_z" + z + ".txt";
        System.out.print("Saving grid data for excel from cycle " + slide + ".....");

        try (PrintWriter file = new PrintWriter(gridFname, "UTF-8")) {
            file.println(slide);
            for (int gy = (numOfY-1); gy >= 0; gy--) {
                for (int gx = 0; gx < numOfX; gx++) {
                    file.print(this.get(gx,gy,z).getV() + " ");
                }
                file.println();
            }
        }

        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    // Save potential on XZ plane at a certain time T and Y in excel format
    public void savePotentialYData4Excel(Config config, String slide, int y) throws
FileNotFoundException, UnsupportedEncodingException {
        String gridFname = config.getDirName() + "/" + config.getSetName() + "_V_s" + slide +
"_y" + y + ".txt";
```

**cps3dn > Grids.java**

```
        System.out.print("Saving grid data for excel from cycle " + slide + ".....");

        try (PrintWriter file = new PrintWriter(gridFname, "UTF-8")) {
            file.println(slide);
            for (int gz = (numOfZ-1); gz >= 0; gz--) {
                for (int gx = 0; gx < numOfX; gx++) {
                    file.print(this.get(gx,y,gz).getV() + " ");
                }
                file.println();
            }
        }

        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    // Save potential on YZ plane at a certain time T and X in excel format
    public void savePotentialXData4Excel(Config config, String slide, int x) throws
FileNotFoundException, UnsupportedEncodingException {
        String gridFname = config.getDirName() + "/" + config.getSetName() + "_V_s" + slide +
"_x" + x + ".txt";
        System.out.print("Saving grid data for excel from cycle " + slide + ".....");

        try (PrintWriter file = new PrintWriter(gridFname, "UTF-8")) {
            file.println(slide);
            for (int gz = (numOfZ-1); gz >= 0; gz--) {
                for (int gy = 0; gy < numOfY; gy++) {
                    file.print(this.get(x,gy,gz).getV() + " ");
                }
                file.println();
            }
        }

        System.out.println("done.");
        System.out.println("--------------------------------");
    }
```

**cps3dn > Grids.java**

```
}
```

**cps3dn > Input.java**

```
package cps3dn;

import interaction.DefaultInteraction;
import crossSection.DefaultCrossSection;
import material.DefaultMaterial;

/**
 *
 * @author ppengvan
 */
public class Input {

// File parameters
    // name of data folder
//    final static String DIR_NAME = "/Users/ppengvan/Documents/Data3D";
    final static String DIR_NAME = "D:\\Data3D-8";
    // data set name to append
    final static String SET_NAME = "3dset6";

// Calculation parameters
    // absolute tolerence for residual (for field solver)
    final static double TOL = 1e-6;
    // tolerence for residual rate of change
    final static double DTOL = 1e-2;
    // relaxation constant
    final static double W = 1.9;
    // Time step in second normalized by 1/omega_p
    final static double DT = 0.0001;
    // Time cycle to run;
    final static int CYCLE = 1000;

// Grid parameters
```

**cps3dn > Input.java**

```
    // Starting x in meters normalized by c/omega_p
    final static double X_I = 0; // Do not change
    // Ending x in meters normalized by c/omega_p
    final static double X_F = 1;
    // Starting y in meters normalized by c/omega_p
    final static double Y_I = 0; // Do not change
    // Ending y in meters normalized by c/omega_p
    final static double Y_F = 1;
    // Starting y in meters normalized by c/omega_p
    final static double Z_I = 0; // Do not change
    // Ending y in meters normalized by c/omega_p
    final static double Z_F = 1;
    // Number of grids in x
    final static int NUM_OF_GRID_X = 21;
    // Number of grids in y
    final static int NUM_OF_GRID_Y = 21;
    // Number of grids in z
    final static int NUM_OF_GRID_Z = 21;
    // Voltage on yi plane in Volts normalized by m*c^2/e
    final static double VOLTAGE_I = 0; // Do not change
    // Voltage on yf plane in Volts normalized by m*c^2/e
    final static double VOLTAGE_F = 0;
    // Voltage frequency in Radian
    final static double VOLTAGE_W = 0;
    // Initial phase of voltage V_f*cos(w*t+theta)
    final static double VOLTAGE_T = 0;
    // Magnetic field (Bx) in Tesla normalized by m*omega_p/e
    final static double B_X = 0;
    // Magnetic field (By) in Tesla normalized by m*omega_p/e
    final static double B_Y = 0;
    // Magnetic field (Bz) in Tesla normalized by m*omega_p/e
    final static double B_Z = 0;

// Charge particle parameters
    // Electron plasma frequency used for normalization
    final static double REFERENCE_DENSITY = 1e14;
    // Types of charge particles
```

**cps3dn > Input.java**

```
    final static int NUM_OF_PARTICLE_TYPE = 2;
    // Classes of charge particles
    final static DefaultMaterial[] PARTICLE_CLASS = {DefaultMaterial.ELECTRON,
DefaultMaterial.ARGON_1};
    // Number of super-particle, Suggestion: Should be < 1e7
    final static int[] NUM_OF_PARTICLE = {128000, 128000};
    // Weight of each super-particle
//    final static double[] PARTICLE_WEIGHT = {1.6e+4, 1.6e+4, 1e11};
//    // Distribution of particle's position is assumed to be uniform
//    // Distribution of particle's velopity is assumed to be Gaussian
//    // Average velocity in x in m/s normalized by c
    final static double[] V_X_AVERAGE = {0, 0};
    // Standard deviation of velocity in x in m/s normalized by c
    final static double[] V_X_STANDARD_DEVIATION = {0, 0};
    // Average velocity in y in m/s normalized by c
    final static double[] V_Y_AVERAGE = {0, 0};
    // Standard deviation of velocity in y in m/s normalized by c
    final static double[] V_Y_STANDARD_DEVIATION = {0, 0};
    // Average velocity in z in m/s normalized by c
    final static double[] V_Z_AVERAGE = {0, 0};
    // Standard deviation of velocity in z in m/s normalized by c
    final static double[] V_Z_STANDARD_DEVIATION = {0, 0};

// Monte Carlo Interaction parameters
    // Interaction pairs are setup using 2 fields to reference incoming
    // and target particles: "INCOMING[i]" and "TARGET[i]".
    // "INCOMING[i]" represents the incoming particle type i whereas
    // "TARGET[i]" represents the target particle type i.
    // INCOMING[i] will interact with TARGET[i]
    final static int INTERACTION_TYPE = 0;
    final static int[] INCOMING = {0};
    final static int[] TARGET = {2};
    final static DefaultInteraction[] INTERACTION =
{DefaultInteraction.CLASSIC_ELASTIC_COLLISION};
    final static DefaultCrossSection[] CROSS_SECTION = {DefaultCrossSection.E_AR};
}
```

**cps3dn > InteractionPair.java**

```
package cps3dn;

import crossSection.DefaultCrossSection;
import interaction.DefaultInteraction;
import java.util.ArrayList;
import java.util.List;
import material.DefaultMaterial;

/**
 *
 * @author ppengvan
 */
public class InteractionPair {
    private DefaultMaterial inc;
    private DefaultMaterial tar;
    private DefaultInteraction inter;
    private DefaultCrossSection xsec;

    public void setInteraction(DefaultMaterial incVal, DefaultMaterial tarVal,
DefaultInteraction interVal, DefaultCrossSection xsVal) {
        inc = incVal;
        tar = tarVal;
        inter = interVal;
        xsec = xsVal;
    }

    public DefaultMaterial getInc() {
        return inc;
    }
    public DefaultMaterial getTar() {
        return tar;
    }
    public DefaultInteraction getInter() {
        return inter;
    }
    public DefaultCrossSection getXSec() {
        return xsec;
```

**cps3dn > InteractionPair.java**

```
    }
}
```

**cps3dn > InteractionPairs.java**

```
package cps3dn;

import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author ppengvan
 */
public class InteractionPairs {
    private List<InteractionPair> ipairs = new ArrayList<>();
    private int inType;

    public InteractionPairs(Config config) {
        inType = config.getInType();
        for (int i = 0; i < inType; i++) {
            InteractionPair ipair = new InteractionPair();
            ipair.setInteraction(config.getParType(config.getInc(i)),
config.getParType(config.getTar(i)), config.getInteraction(i), config.getXSection(i));

            ipairs.add(ipair);
        }
    }

    public void listInteractionPairs() {
        System.out.println("Interaction Pairs");
        System.out.println("-------------------------------");
        if (inType == 0) {
            System.out.println("No interation");
        } else {
            for (int i = 0; i < inType; i++) {
                System.out.print("Pair #" + i + " : ");
```

**cps3dn > InteractionPairs.java**

```
                System.out.println(ipairs.get(i).getInter().toString() + " between " +
ipairs.get(i).getInc().toString() + " and " + ipairs.get(i).getTar().toString());
            }
        }

        System.out.println("--------------------------------");
    }

    public List<InteractionPair> getInteractionPairs() {
        return ipairs;
    }
}
```

**cps3dn > Momentum.java**

```
package cps3dn;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author ppengvan
 */
public class Momentum {
    private double parMomentumX;
    private double parMomentumY;
    private double parMomentumZ;
    private double fieldMomentumX;
    private double fieldMomentumY;
    private double fieldMomentumZ;
```

**cps3dn > Momentum.java**

```
    public void calculateParticleMomentum(Config config, Particles pars) {
        parMomentumX = 0;
        parMomentumY = 0;
        parMomentumZ = 0;

        for (Particle p : pars.get()) {
            if (p.getCell() != -1) {
                parMomentumX = parMomentumX +
config.getMass(p.getTypeID())*0.5*(p.getVXP()+p.getVX());
                parMomentumY = parMomentumY +
config.getMass(p.getTypeID())*0.5*(p.getVYP()+p.getVY());
                parMomentumZ = parMomentumZ +
config.getMass(p.getTypeID())*0.5*(p.getVZP()+p.getVZ());
            }
        }
    }

    public void calculateFieldMomentum(Config config, Grids grids) {
        fieldMomentumX = 0;
        fieldMomentumY = 0;
        fieldMomentumZ = 0;

        double v = config.getGridSizeX()*config.getGridSizeY()*config.getGridSizeZ();
        double rho;
        double numOfX = config.getNumOfX();
        double numOfY = config.getNumOfY();
        double numOfZ = config.getNumOfZ();

        double vol = 0;

        for (int gx = 0; gx < numOfX; gx++) {
            for (int gy = 0; gy < numOfY; gy++) {
                for (int gz = 0; gz < numOfZ; gz++) {
                    rho = 0;
                    for (int i = 0; i < config.getNType(); i++) {
                        rho = rho + (config.getCharge(i) * (grids.get(gx,gy,gz).getN(i)));
                    }
```

**cps3dn > Momentum.java**

```
                if (((gx == 0) || (gx == numOfX-1)) &&
                    ((gy == 0) || (gy == numOfY-1)) &&
                    ((gz == 0) || (gz == numOfZ-1))) {
//                      totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/8);
                    fieldMomentumX = fieldMomentumX + (rho*grids.get(gx, gy,
gz).getEX()*v/8.0);
                    fieldMomentumY = fieldMomentumY + (rho*grids.get(gx, gy,
gz).getEY()*v/8.0);
                    fieldMomentumZ = fieldMomentumZ + (rho*grids.get(gx, gy,
gz).getEZ()*v/8.0);
                    vol = vol + v/8;
                }
                else if (((gx != 0) && (gx != numOfX-1)) &&
                        ((gy == 0) || (gy == numOfY-1)) &&
                        ((gz == 0) || (gz == numOfZ-1))) {
//                      totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/4);
                    fieldMomentumX = fieldMomentumX + (rho*grids.get(gx, gy,
gz).getEX()*v/4.0);
                    fieldMomentumY = fieldMomentumY + (rho*grids.get(gx, gy,
gz).getEY()*v/4.0);
                    fieldMomentumZ = fieldMomentumZ + (rho*grids.get(gx, gy,
gz).getEZ()*v/4.0);
                    vol = vol + v/4;
                }
                else if (((gx == 0) || (gx == numOfX-1)) &&
                        ((gy != 0) && (gy != numOfY-1)) &&
                        ((gz == 0) || (gz == numOfZ-1))) {
//                      totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/4);
                    fieldMomentumX = fieldMomentumX + (rho*grids.get(gx, gy,
gz).getEX()*v/4.0);
```

**cps3dn > Momentum.java**

```
                    fieldMomentumY = fieldMomentumY + (rho*grids.get(gx, gy,
gz).getEY()*v/4.0);
                    fieldMomentumZ = fieldMomentumZ + (rho*grids.get(gx, gy,
gz).getEZ()*v/4.0);
                    vol = vol + v/4;
                }
                else if (((gx == 0) || (gx == numOfX-1)) &&
                        ((gy == 0) || (gy == numOfY-1)) &&
                        ((gz != 0) && (gz != numOfZ-1))) {
//                    totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/4);
                    fieldMomentumX = fieldMomentumX + (rho*grids.get(gx, gy,
gz).getEX()*v/4.0);
                    fieldMomentumY = fieldMomentumY + (rho*grids.get(gx, gy,
gz).getEY()*v/4.0);
                    fieldMomentumZ = fieldMomentumZ + (rho*grids.get(gx, gy,
gz).getEZ()*v/4.0);
                    vol = vol+v/4;
                }
                else if (((gx != 0) && (gx != numOfX-1)) &&
                        ((gy != 0) && (gy != numOfY-1)) &&
                        ((gz != 0) && (gz != numOfZ-1))) {
//                    totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v);
                    fieldMomentumX = fieldMomentumX + (rho*grids.get(gx, gy,
gz).getEX()*v);
                    fieldMomentumY = fieldMomentumY + (rho*grids.get(gx, gy, gz).getEY()*v);
                    fieldMomentumZ = fieldMomentumZ + (rho*grids.get(gx, gy,
gz).getEZ()*v);
                    vol = vol + v;
                }
                else {
//                    totalPE = totalPE + (0.5*DefaultConstant.EPS*(grids.get(gx, gy,
gz).getEX()*grids.get(gx, gy, gz).getEX() + grids.get(gx, gy, gz).getEY()*grids.get(gx, gy,
gz).getEY() + grids.get(gx, gy, gz).getEZ()*grids.get(gx, gy, gz).getEZ())*v/2);
```

**cps3dn > Momentum.java**

```
                    fieldMomentumX = fieldMomentumX + (rho*grids.get(gx, gy,
gz).getEX()*v/2.0);
                    fieldMomentumY = fieldMomentumY + (rho*grids.get(gx, gy,
gz).getEY()*v/2.0);
                    fieldMomentumZ = fieldMomentumZ + (rho*grids.get(gx, gy,
gz).getEZ()*v/2.0);
                    vol = vol + v/2;
                }
            }
        }
    }
    System.out.println("Total integrated volume for calculating total momentum = " +
vol);
//      System.out.println("total: "+i+" "+j+" "+k+" "+l);
  }

  public double getParticleMomentumX() {
    return parMomentumX;
  }

  public double getParticleMomentumY() {
    return parMomentumY;
  }

  public double getParticleMomentumZ() {
    return parMomentumZ;
  }

  public double getFieldMomentumX() {
    return fieldMomentumX;
  }

  public double getFieldMomentumY() {
    return fieldMomentumY;
  }

  public double getFieldMomentumZ() {
```

**cps3dn > Momentum.java**

```
        return fieldMomentumZ;
    }

    public void saveMomentumData(Config config, String slide) throws
FileNotFoundException, UnsupportedEncodingException {
        String momentumFname = config.getDirName() + "/" + config.getSetName() +
"_momentum.txt";

        System.out.print("Saving mementum data from cycle " + slide + ".....");

        try (PrintWriter file = new PrintWriter(new FileOutputStream(
                new File(momentumFname),
                true /* append = true */))) {
            file.print(slide + " ");
            file.print((config.getDT()*Integer.valueOf(slide)) + " ");
            file.print(parMomentumX + " ");
            file.print(parMomentumY + " ");
            file.print(parMomentumZ + " ");
            file.print((parMomentumX+parMomentumY+parMomentumZ) + " ");
            file.print(fieldMomentumX + " ");
            file.print(fieldMomentumY + " ");
            file.print(fieldMomentumZ + " ");
            file.print((fieldMomentumX+fieldMomentumY+fieldMomentumZ));
            file.println();
        }

        System.out.println("done.");
        System.out.println("--------------------------------");
    }
}
```

**cps3dn > Normalization.java**

```
package cps3dn;

import constant.Constant;
```

**cps3dn > Normalization.java**

```
/**
 *
 * @author ppengvan
 */
public class Normalization {
    private double nepsilon;
    private double ncharge;
    private double nmass;
    private double nvelocity;
    private double nplasmafrequency;
    private double ndensity;
    private double nlength;
    private double ntime;
    private double npotential;
    private double nelectricfield;
    private double nmagneticfield;
    private double nkineticenergy;
    private double npotentialenergy;
    private double nparticleweight;

    public Normalization(Config config) {
        nepsilon = Constant.EPS;
        ncharge = Constant.QE;
        nmass = Constant.ME;
        nvelocity = Constant.C;
        ndensity = config.getNo();
        nplasmafrequency = Math.sqrt((ncharge*ncharge*ndensity)/(nepsilon*nmass));
        nlength = nvelocity/nplasmafrequency;
        ntime = 1.0/nplasmafrequency;
        npotential = (nmass*nvelocity*nvelocity)/ncharge;
        nelectricfield = (nmass*nplasmafrequency*nvelocity)/ncharge;
        nmagneticfield = (nmass*nplasmafrequency)/ncharge;
        nkineticenergy = nmass*nvelocity*nvelocity;
        npotentialenergy = nepsilon*nmass*nmass*Math.pow(nvelocity,
5)/(ncharge*ncharge*nplasmafrequency);
        nparticleweight = ndensity*((config.getXf()-config.getXi())*(config.getYf()-
config.getYi())*(config.getZf()-config.getZi()))/config.getNumOfPars(0);
```

**cps3dn > Normalization.java**

```
    }

    public double dnPotentialEnergy(double val) {
        return val*npotentialenergy;
    }
    public double dnKineticEnergy(double val) {
        return val*nkineticenergy;
    }
}
```

**cps3dn > Particle.java**

```
package cps3dn;

/**
 *
 * @author ppengvan
 */
public class Particle {
    private int id;
    private int cell;
    private double x;
    private double y;
    private double z;
    private double vx;
    private double vy;
    private double vz;
    private double vxp;
    private double vyp;
    private double vzp;
    private double txdisp;
    private double tydisp;
    private double tzdisp;
    private int typeID;

    public void setID(int idVal) {
        id = idVal;
```

**cps3dn > Particle.java**

```
    }
    public void setTypeID(int typeIDVal) {
        typeID = typeIDVal;
    }
    public void setCell(int cellVal) {
        cell = cellVal;
    }
    public void setX(double xVal) {
        x = xVal;
    }
    public void setY(double yVal) {
        y = yVal;
    }
    public void setZ(double zVal) {
        z = zVal;
    }
    public void setVX(double vxVal) {
        vx = vxVal;
    }
    public void setVY(double vyVal) {
        vy = vyVal;
    }
    public void setVZ(double vzVal) {
        vz = vzVal;
    }
    public void setVXP(double vxpVal) {
        vxp = vxpVal;
    }
    public void setVYP(double vypVal) {
        vyp = vypVal;
    }
    public void setVZP(double vzpVal) {
        vzp = vzpVal;
    }
    public void setTXDisp(double val) {
        txdisp = val;
    }
```

**cps3dn > Particle.java**

```
    public void setTYDisp(double val) {
        tydisp = val;
    }
    public void setTZDisp(double val) {
        tzdisp = val;
    }

    public int getID() {
        return id;
    }
    public int getTypeID() {
        return typeID;
    }
    public int getCell() {
        return cell;
    }
    public double getX() {
        return x;
    }
    public double getY() {
        return y;
    }
    public double getZ() {
        return z;
    }
    public double getVX() {
        return vx;
    }
    public double getVY() {
        return vy;
    }
    public double getVZ() {
        return vz;
    }
    public double getVXP() {
        return vxp;
    }
```

**cps3dn > Particle.java**

```
    public double getVYP() {
        return vyp;
    }
    public double getVZP() {
        return vzp;
    }
    public double getTXDisp() {
        return txdisp;
    }
    public double getTYDisp() {
        return tydisp;
    }
    public double getTZDisp() {
        return tzdisp;
    }
}
```

**cps3dn > Particles.java**

```
package cps3dn;

import crossSection.CrossSection;
import crossSection.DefaultCrossSection;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import material.DefaultMaterial;

/**
 *
 * @author ppengvan
 */
```

**cps3dn > Particles.java**

```
public class Particles {
    private final List<Particle> pars = new ArrayList<>();
    private int numOfPars = 0;
    private int numOfParsInSys = 0;
    private int numOfInteractions = 0;

    public Particles(Config config) {
    // Create particles, initialize particle values
        int k = 0;
        double x, y, z;
        int np, gx, gy, gz;

        System.out.print("Creating super-particles.....");

        for (int j = 0; j < config.getNType(); j++) {
            Random randx = new Random();
            Random randy = new Random();
            Random randz = new Random();
            Random randvx = new Random();
            Random randvy = new Random();
            Random randvz = new Random();

            np = (int)Math.ceil(Math.pow(config.getNumOfPars(j), 1.0/3.0));

            for (int i = 0; i < config.getNumOfPars(j); i++) {
                Particle par = new Particle();
                par.setID(k);
                par.setTypeID(j);

//                par.setX(randx.nextDouble()*(config.getXf()-config.getXi())+config.getXi());  // or
nextGaussian()
//                par.setY(randy.nextDouble()*(config.getYf()-config.getYi())+config.getYi());  // or
nextGaussian()
//                par.setZ(randz.nextDouble()*(config.getZf()-config.getZi())+config.getZi());  // or
nextGaussian()
```

**cps3dn > Particles.java**

```
            // Alternatively the particle can have uniform distribution with some
perturbation in the certain direction
            gx = i / (np * np);
            gy = (i % (np * np)) / np;
            gz = i % np;
            x = ((gx+0.5) * (config.getXf() - config.getXi()) / np) + config.getXi();
            y = ((gy+0.5) * (config.getYf() - config.getYi()) / np) + config.getYi();
            z = ((gz+0.5) * (config.getZf() - config.getZi()) / np) + config.getZi();
            par.setX(x);
            if (j == 0) {
                par.setY(y + (0.001 * Math.sin(y)));
//                  par.setY(y+0.001);
            }
            else {
                par.setY(y);
            }
            par.setZ(z);

            double vx = randvx.nextGaussian()*config.getVXStd(j)+config.getVXAvg(j);
            double vy = randvy.nextGaussian()*config.getVYStd(j)+config.getVYAvg(j);
            double vz = randvz.nextGaussian()*config.getVZStd(j)+config.getVZAvg(j);

            par.setVX(vx);
            par.setVY(vy);
            par.setVZ(vz);

            par.setVXP(vx);
            par.setVYP(vy);
            par.setVZP(vz);

            par.setCell(0);

            pars.add(par);
            k = k + 1;
        }
    }
```

**cps3dn > Particles.java**

```
        numOfPars = k;

        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public void updateCellLoc(Config config) {
    // Find out whether each particle is still inside the system, and in which cell the
particle resides
        numOfParsInSys = 0;
        double dx = config.getGridSizeX();
        double dy = config.getGridSizeY();
        double dz = config.getGridSizeZ();

        System.out.print("Counting super-particles in the system.....");

        for (Particle eachPar : pars) {
            double x = eachPar.getX();
            double y = eachPar.getY();
            double z = eachPar.getZ();

            // Continuous boundary in x
            if (x < config.getXi()) {
                x = config.getXf() - ((config.getXi() - x) % (config.getXf() - config.getXi()));
                eachPar.setX(x);
            } else if (x > config.getXf()) {
                x = config.getXi() + ((x - config.getXf()) % (config.getXf() - config.getXi()));
                eachPar.setX(x);
            }

            // Continuous boundary in y
            if (y < config.getYi()) {
                y = config.getYf() - ((config.getYi() - y) % (config.getYf() - config.getYi()));
                eachPar.setY(y);
            } else if (y > config.getYf()) {
                y = config.getYi() + ((y - config.getYf()) % (config.getYf() - config.getYi()));
                eachPar.setY(y);
```

**cps3dn > Particles.java**

```
            }

            // Continuous boundary in z
            if (z < config.getZi()) {
                z = config.getZf() - ((config.getZi() - z) % (config.getZf() - config.getZi()));
                eachPar.setZ(z);
            } else if (z > config.getZf()) {
                z = config.getZi() + ((z - config.getZf()) % (config.getZf() - config.getZi()));
                eachPar.setZ(z);
            }

            if (isInside(config, eachPar)) {
                int cellLoc = (int)((x-config.getXi())/dx)*(config.getNumOfY()-
1)*(config.getNumOfZ()-1) +
                            (int)((y-config.getYi())/dy)*(config.getNumOfZ()-1) + (int)((z-
config.getZi())/dz);

                eachPar.setCell(cellLoc);
                numOfParsInSys = numOfParsInSys + 1;
            }
            else {
                eachPar.setCell(-1);
            }
        }
        System.out.println("done.");
        System.out.println("Number of super-particles in the system = " + numOfParsInSys);
        System.out.println("--------------------------------");
    }

    public void calculateInitialVelocity(Config config, Grids grids, Cells cells) {
    // Move particle based on Lorentz force due to electric & magnetic fields back half
time step
        double Epx = 0;
        double Epy = 0;
        double Epz = 0;
        double Bpx = 0;
        double Bpy = 0;
```

**cps3dn > Particles.java**

```
        double Bpz = 0;
        int c;
        int typeID;
        double x, y, z;
        double dx, dy, dz;
        int g0, g1, g2, g3, g4, g5, g6, g7;
        double v, v0, v1, v2, v3, v4, v5, v6, v7;
        double vx, vy, vz;
        double dt;

        System.out.print("Calculate First Velocity...");

        dt = config.getDT();

        v = config.getGridSizeX()*config.getGridSizeY()*config.getGridSizeZ();

        for (Particle eachPar : pars) {
            if (eachPar.getCell() != -1) {
                c = eachPar.getCell();
                typeID = eachPar.getTypeID();
                x = eachPar.getX();
                y = eachPar.getY();
                z = eachPar.getZ();
                g0 = cells.get(c).getG0();
                g1 = cells.get(c).getG1();
                g2 = cells.get(c).getG2();
                g3 = cells.get(c).getG3();
                g4 = cells.get(c).getG4();
                g5 = cells.get(c).getG5();
                g6 = cells.get(c).getG6();
                g7 = cells.get(c).getG7();
                v0 = (x-grids.get(g0).getX()) * (y-grids.get(g0).getY()) * (z-grids.get(g0).getZ());
                v1 = (x-grids.get(g1).getX()) * (y-grids.get(g1).getY()) * (grids.get(g1).getZ()-z);
                v2 = (x-grids.get(g2).getX()) * (grids.get(g2).getY()-y) * (z-grids.get(g2).getZ());
                v3 = (x-grids.get(g3).getX()) * (grids.get(g3).getY()-y) * (grids.get(g3).getZ()-z);
                v4 = (grids.get(g4).getX()-x) * (y-grids.get(g4).getY()) * (z-grids.get(g4).getZ());
                v5 = (grids.get(g5).getX()-x) * (y-grids.get(g5).getY()) * (grids.get(g5).getZ()-z);
```

**cps3dn > Particles.java**

```
            v6 = (grids.get(g6).getX()-x) * (grids.get(g6).getY()-y) * (z-grids.get(g6).getZ());
            v7 = (grids.get(g7).getX()-x) * (grids.get(g7).getY()-y) * (grids.get(g7).getZ()-z);

            Epx = grids.get(g0).getEX()*v7/v + grids.get(g1).getEX()*v6/v +
                  grids.get(g2).getEX()*v5/v + grids.get(g3).getEX()*v4/v +
                  grids.get(g4).getEX()*v3/v + grids.get(g5).getEX()*v2/v +
                  grids.get(g6).getEX()*v1/v + grids.get(g7).getEX()*v0/v;
            Epy = grids.get(g0).getEY()*v7/v + grids.get(g1).getEY()*v6/v +
                  grids.get(g2).getEY()*v5/v + grids.get(g3).getEY()*v4/v +
                  grids.get(g4).getEY()*v3/v + grids.get(g5).getEY()*v2/v +
                  grids.get(g6).getEY()*v1/v + grids.get(g7).getEY()*v0/v;
            Epz = grids.get(g0).getEZ()*v7/v + grids.get(g1).getEZ()*v6/v +
                  grids.get(g2).getEZ()*v5/v + grids.get(g3).getEZ()*v4/v +
                  grids.get(g4).getEZ()*v3/v + grids.get(g5).getEZ()*v2/v +
                  grids.get(g6).getEZ()*v1/v + grids.get(g7).getEZ()*v0/v;
            Bpx = grids.get(g0).getBX()*v7/v + grids.get(g1).getBX()*v6/v +
                  grids.get(g2).getBX()*v5/v + grids.get(g3).getBX()*v4/v +
                  grids.get(g4).getBX()*v3/v + grids.get(g5).getBX()*v2/v +
                  grids.get(g6).getBX()*v1/v + grids.get(g7).getBX()*v0/v;
            Bpy = grids.get(g0).getBY()*v7/v + grids.get(g1).getBY()*v6/v +
                  grids.get(g2).getBY()*v5/v + grids.get(g3).getBY()*v4/v +
                  grids.get(g4).getBY()*v3/v + grids.get(g5).getBY()*v2/v +
                  grids.get(g6).getBY()*v1/v + grids.get(g7).getBY()*v0/v;
            Bpz = grids.get(g0).getBZ()*v7/v + grids.get(g1).getBZ()*v6/v +
                  grids.get(g2).getBZ()*v5/v + grids.get(g3).getBZ()*v4/v +
                  grids.get(g4).getBZ()*v3/v + grids.get(g5).getBZ()*v2/v +
                  grids.get(g6).getBZ()*v1/v + grids.get(g7).getBZ()*v0/v;

            vx = eachPar.getVX() -
(dt/2.0*config.getCharge(eachPar.getTypeID())/config.getMass(eachPar.getTypeID()))*Epx;
            vy = eachPar.getVY() -
(dt/2.0*config.getCharge(eachPar.getTypeID())/config.getMass(eachPar.getTypeID()))*Epy;
            vz = eachPar.getVZ() -
(dt/2.0*config.getCharge(eachPar.getTypeID())/config.getMass(eachPar.getTypeID()))*Epz;

            eachPar.setVX(vx);
            eachPar.setVY(vy);
```

**cps3dn > Particles.java**

```
                eachPar.setVZ(vz);

                eachPar.setVXP(vx);
                eachPar.setVYP(vy);
                eachPar.setVZP(vz);
            }
        }
        System.out.println("done.");
        System.out.println("-------------------------------");
    }

    public void moveParticle(Config config, Grids grids, Cells cells) {
    // Move particle based on Lorentz force due to electric & magnetic fields
        double Epx = 0;
        double Epy = 0;
        double Epz = 0;
        double Bpx = 0;
        double Bpy = 0;
        double Bpz = 0;
        int c;
        int typeID;
        double x, y, z;
        double dx, dy, dz;
        int g0, g1, g2, g3, g4, g5, g6, g7;
        double v, v0, v1, v2, v3, v4, v5, v6, v7;
        double vx, vy, vz;
        double dt;
        double qprime;
        double [] h = new double[3];
        double hsquare;
        double [] s = new double[3];
        double [] u = new double[3];
        double [] up = new double[3];

        System.out.print("Moving charge super-particles.....");

        dt = config.getDT();
```

**cps3dn > Particles.java**

```
v = config.getGridSizeX()*config.getGridSizeY()*config.getGridSizeZ();

for (Particle eachPar : pars) {
    if (eachPar.getCell() != -1) {
        c = eachPar.getCell();
        typeID = eachPar.getTypeID();
        x = eachPar.getX();
        y = eachPar.getY();
        z = eachPar.getZ();
        g0 = cells.get(c).getG0();
        g1 = cells.get(c).getG1();
        g2 = cells.get(c).getG2();
        g3 = cells.get(c).getG3();
        g4 = cells.get(c).getG4();
        g5 = cells.get(c).getG5();
        g6 = cells.get(c).getG6();
        g7 = cells.get(c).getG7();
        v0 = (x-grids.get(g0).getX()) * (y-grids.get(g0).getY()) * (z-grids.get(g0).getZ());
        v1 = (x-grids.get(g1).getX()) * (y-grids.get(g1).getY()) * (grids.get(g1).getZ()-z);
        v2 = (x-grids.get(g2).getX()) * (grids.get(g2).getY()-y) * (z-grids.get(g2).getZ());
        v3 = (x-grids.get(g3).getX()) * (grids.get(g3).getY()-y) * (grids.get(g3).getZ()-z);
        v4 = (grids.get(g4).getX()-x) * (y-grids.get(g4).getY()) * (z-grids.get(g4).getZ());
        v5 = (grids.get(g5).getX()-x) * (y-grids.get(g5).getY()) * (grids.get(g5).getZ()-z);
        v6 = (grids.get(g6).getX()-x) * (grids.get(g6).getY()-y) * (z-grids.get(g6).getZ());
        v7 = (grids.get(g7).getX()-x) * (grids.get(g7).getY()-y) * (grids.get(g7).getZ()-z);

        Epx = grids.get(g0).getEX()*v7/v + grids.get(g1).getEX()*v6/v +
              grids.get(g2).getEX()*v5/v + grids.get(g3).getEX()*v4/v +
              grids.get(g4).getEX()*v3/v + grids.get(g5).getEX()*v2/v +
              grids.get(g6).getEX()*v1/v + grids.get(g7).getEX()*v0/v;
        Epy = grids.get(g0).getEY()*v7/v + grids.get(g1).getEY()*v6/v +
              grids.get(g2).getEY()*v5/v + grids.get(g3).getEY()*v4/v +
              grids.get(g4).getEY()*v3/v + grids.get(g5).getEY()*v2/v +
              grids.get(g6).getEY()*v1/v + grids.get(g7).getEY()*v0/v;
        Epz = grids.get(g0).getEZ()*v7/v + grids.get(g1).getEZ()*v6/v +
              grids.get(g2).getEZ()*v5/v + grids.get(g3).getEZ()*v4/v +
```

**cps3dn > Particles.java**

```
                    grids.get(g4).getEZ()*v3/v + grids.get(g5).getEZ()*v2/v +
                    grids.get(g6).getEZ()*v1/v + grids.get(g7).getEZ()*v0/v;
                Bpx = grids.get(g0).getBX()*v7/v + grids.get(g1).getBX()*v6/v +
                    grids.get(g2).getBX()*v5/v + grids.get(g3).getBX()*v4/v +
                    grids.get(g4).getBX()*v3/v + grids.get(g5).getBX()*v2/v +
                    grids.get(g6).getBX()*v1/v + grids.get(g7).getBX()*v0/v;
                Bpy = grids.get(g0).getBY()*v7/v + grids.get(g1).getBY()*v6/v +
                    grids.get(g2).getBY()*v5/v + grids.get(g3).getBY()*v4/v +
                    grids.get(g4).getBY()*v3/v + grids.get(g5).getBY()*v2/v +
                    grids.get(g6).getBY()*v1/v + grids.get(g7).getBY()*v0/v;
                Bpz = grids.get(g0).getBZ()*v7/v + grids.get(g1).getBZ()*v6/v +
                    grids.get(g2).getBZ()*v5/v + grids.get(g3).getBZ()*v4/v +
                    grids.get(g4).getBZ()*v3/v + grids.get(g5).getBZ()*v2/v +
                    grids.get(g6).getBZ()*v1/v + grids.get(g7).getBZ()*v0/v;

                // Implementing Boris scheme
                qprime =
dt/2.0*config.getCharge(eachPar.getTypeID())/config.getMass(eachPar.getTypeID());

                eachPar.setVXP(eachPar.getVX());
                eachPar.setVYP(eachPar.getVY());
                eachPar.setVZP(eachPar.getVZ());

                h[0] = qprime*Bpx;
                h[1] = qprime*Bpy;
                h[2] = qprime*Bpz;
                hsquare = h[0]*h[0]+h[1]*h[1]+h[2]*h[2];

                s[0] = h[0]*2.0/(1+hsquare);
                s[1] = h[1]*2.0/(1+hsquare);
                s[2] = h[2]*2.0/(1+hsquare);

                u[0] = eachPar.getVX() + qprime*Epx;
                u[1] = eachPar.getVY() + qprime*Epy;
                u[2] = eachPar.getVZ() + qprime*Epz;

                CPSMath.crossproduct(up,u,h);
```

**cps3dn > Particles.java**

```
                CPSMath.addproduct(up,u,up);
                CPSMath.crossproduct(up,up,s);
                CPSMath.addproduct(up,u,up);

                vx = up[0] + qprime*Epx;
                vy = up[1] + qprime*Epy;
                vz = up[2] + qprime*Epz;

                dx = dt*vx;
                dy = dt*vy;
                dz = dt*vz;

                x = x + dx;
                y = y + dy;
                z = z + dz;

                eachPar.setX(x);
                eachPar.setY(y);
                eachPar.setZ(z);
                eachPar.setVX(vx);
                eachPar.setVY(vy);
                eachPar.setVZ(vz);
                eachPar.setTXDisp(eachPar.getTXDisp()+dx);
                eachPar.setTYDisp(eachPar.getTYDisp()+dy);
                eachPar.setTZDisp(eachPar.getTZDisp()+dz);
//                // Check velocity calculation
//                if (eachPar.getID()==4) {
//                    System.out.println("vx = " + vx + " or " + (eachPar.getVXP()+2*qprime*Epx));
//                }
            }
        }
        System.out.println("done.");
        System.out.println("--------------------------------");
    }

    public void collideParticle(Config config, Grids grids, Cells cells, InteractionPairs ipairs) {
        System.out.println("Colliding particles.");
```

**cps3dn > Particles.java**

```
        for (int i = 0; i < config.getInType(); i++) {
            DefaultCrossSection xsec = ipairs.getInteractionPairs().get(i).getXSec();
            Random rand1 = new Random();
            Random rand2 = new Random();
            Random rand3 = new Random();
            Double r1 = rand1.nextDouble();
            Double r2, r3;
            double theta, phi;
            double vol = (config.getXf()-config.getXi())*(config.getYf()-
config.getYi())*(config.getZf()-config.getZi());

            for (Particle p : pars) {
                if (config.getInc(i) == p.getTypeID()) {
                    if (isInside(config, p)) {
                        double u = Math.sqrt(p.getVX()*p.getVX() + p.getVY()*p.getVY() +
p.getVZ()*p.getVZ());
//                          double nTar =
config.getNumOfPars(config.getTar(i))*config.getWeight(config.getTar(i))/vol;
                        double nTar = config.getNumOfPars(config.getTar(i))/vol;
                        double xs = CrossSection.getCrossSection(xsec).getTotalXSection();
                        double ts = config.getDT();
                        double prob = 1 - Math.exp(-u*nTar*xs*ts);
                        System.out.println("u=" + u + ",nTar=" + nTar + ",xs=" + xs + ",ts=" + ts +
",prob=" + prob);
                        if (prob > r1) {
                            r2 = rand2.nextDouble();
                            r3 = rand3.nextDouble();
                            theta = Math.PI * r2;
                            phi = Math.PI * 2 * r3;
                            p.setVX(u*Math.sin(theta)*Math.cos(phi));
                            p.setVY(u*Math.sin(theta)*Math.sin(phi));
                            p.setVZ(u*Math.cos(theta));
                            numOfInteractions++;
                        }
                    }
                }
            }
```

**cps3dn > Particles.java**

```
        }
    }

    public boolean isInside(Config c, Particle p) {
        double x = p.getX();
        double y = p.getY();
        double z = p.getZ();

        if ((x >= c.getXi()) && (x <= c.getXf()) &
            (y >= c.getYi()) && (y <= c.getYf()) &
            (z >= c.getZi()) && (z <= c.getZf())) {

            return true;
        }
        else {
            return false;
        }
    }

    public List<Particle> get(){
        return pars;
    }

    public Particle get(int i){
        return pars.get(i);
    }

    public int getNumOfPars() {
        return numOfPars;
    }

    public int getNumOfParsInSys() {
        return numOfParsInSys;
    }

    public int getNumOfInteractions() {
        return numOfInteractions;
```

**cps3dn > Particles.java**

```
    }

    public void listParticles(Config config) {
    // List particle values
        System.out.println("Number of Super-Particles = " + numOfPars);
        System.out.println("--------------------------------");
        for (Particle eachPar : pars) {
            System.out.print(eachPar.getID() + "    ");
            System.out.print(eachPar.getCell() + "    ");
            System.out.print(eachPar.getTypeID() + "    ");
            System.out.print(eachPar.getX() + "    ");
            System.out.print(eachPar.getY() + "    ");
            System.out.print(eachPar.getZ() + "    ");
            System.out.print(eachPar.getVX() + "    ");
            System.out.print(eachPar.getVY() + "    ");
            System.out.print(eachPar.getVZ() + "    ");
        }
        System.out.println("--------------------------------");
    }

    public void saveParData(Config config, String slide) throws FileNotFoundException,
UnsupportedEncodingException {
        String parFname = config.getDirName() + "/" + config.getSetName() + "_particle_" +
slide + ".txt";

        System.out.print("Saving particle data for excel from cycle " + slide + ".....");

        try (PrintWriter file = new PrintWriter(parFname, "UTF-8")) {
            file.println(slide);
            for (int i = 0; i < numOfPars; i++) {
                if (pars.get(i).getCell() != -1) {
                    file.print(pars.get(i).getTypeID() + " ");
                    file.print(pars.get(i).getID() + " ");
                    file.print(pars.get(i).getX() + " ");
                    file.print(pars.get(i).getY() + " ");
                    file.print(pars.get(i).getZ() + " ");
                    file.print(pars.get(i).getVX() + " ");
```

**cps3dn > Particles.java**

```
                    file.print(pars.get(i).getVY() + " ");
                    file.print(pars.get(i).getVZ() + " ");
                    file.print(pars.get(i).getTXDisp() + " ");
                    file.print(pars.get(i).getTYDisp() + " ");
                    file.print(pars.get(i).getTZDisp());
                    file.println();
                }
            }
        }

        System.out.println("done.");
        System.out.println("--------------------------------");
    }
}
```

# ประวัตินักวิจัย

## ผู้ช่วยศาสตราจารย์ ดร.พงษ์แพทย์ เพ่งวาณิชย์


1. **ข้อมูลส่วนตัว** ชื่อ-นามสกุล (ภาษาไทย) นาย พงษ์แพทย์ เพ่งวาณิชย์

   (ภาษาอังกฤษ) Mr. Phongphaeth Pengvanich

2. **เลขบัตรประจำตัวประชาชน** 3619900088459

3. **ตำแหน่งวิชาการ** ผู้ช่วยศาสตราจารย์

4. **สถานที่ทำงานปัจจุบัน**

   ภาควิชา วิศวกรรมนิวเคลียร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

   ถนนพญาไท เขตปทุมวัน กรุงเทพมหานคร

   รหัสไปรษณีย์ 10330 โทรศัพท์ 02-218-6770 โทรศัพท์มือถือ 080-558-6931

   โทรสาร 02-218-6780

   E-mail: phongphaeth.p@chula.ac.th


5. **ประวัติการศึกษา**

   **ปริญญาเอก** Doctor of Philosophy (Nuclear Engineering) ปีที่จบ พ.ศ. 2550

   University of Michigan, Ann Arbor, USA

   **ปริญญาโท** Master of Science and Engineering **(**Nuclear Engineering) ปีที่จบ พ.ศ. 2546

   University of Michigan, Ann Arbor, USA

   **ปริญญาตรี** Bachelor of Science and Engineering **(**Nuclear Engineering) ปีที่จบ พ.ศ. 2545

   University of Michigan, Ann Arbor, USA

6. **สาขาวิชาการที่มีความชำนาญพิเศษ**

Theoretical modeling and simulation for high power microwave and millimeter wave sources, specifically magnetrons, klystrons, and traveling wave amplifiers; Processing of plasmas for industrial and medical applications, specifically low temperature atmospheric plasmas

**7. ประสบการณ์การทำวิจัย**

1. “ Readiness Plan for the Implementation of Nuclear Power Plant in Thailand” from 2016-2017
2. “ Development of Car-Borne Gamma Survey and Data Management Systems for Preparedness and Response to Nuclear Security Incident” from 2015 to 2017
3. “ Development of software for design and evaluation of the physical protection system of facilities utilizing nuclear and radiological materials” from 2014 to 2015
4. “ Prospect of Using Small Modular Reactor in Thailand for Power Generation” from 2013 to 2014
5. “ Policy research on methodology for establishing nuclear nonproliferation and its supporting framework in the ASEAN region” from 2011 to 2011
6. “ Software development for the study of low-temperature atmospheric plasmas” from 2011 to 2013
7. “ Development of low-temperature plasma source at atmospheric pressure” from 2010 to 2011
8. “Study of strategic environmental assessment for nuclear power plant project” for Nuclear Power Program Development Office (NPPDO), Ministry of Energy, Thailand from 2009 to 2010.

9. “Study and preparation of bill for overseeing safety of nuclear power plant” for Office of Atomic Energy for Peace (OAEP), Ministry of Science and Technology, Thailand from 2008 to 2010.

**8. การบรรยายพิเศษ**

1. Invited Talks and Presentations:
2. “Asian Network for Education in Nuclear” at Third Conference on Energy and Nuclear Power in Africa, Kenya, 2015
3. “Emergency response and preparedness of nuclear power plant” and “Nuclear safety culture” at the Electricity Generating Authority of Thailand (EGAT) in 2012, 2013, 2014, 2015.
4. “Probabilistic safety analysis of nuclear power plant” at the Electricity Generating Authority of Thailand (EGAT) in 2011.
5. “Nuclear Fusion” at the Electricity Generating Authority of Thailand (EGAT) in 2011.
6. “Probabilistic safety analysis of nuclear power plant” at the Office of Atoms for Peace (OAEP) in 2010
7. “Decommissioning of nuclear power plant” and “Power policies and technological developments” at the Electricity Generating Authority of Thailand (EGAT) in 2010.
8. “Nuclear Safety” at the Electricity Generating Authority of Thailand (EGAT) 2010-2016.
9. “Theory of magnetron injection locking” at Walailak University in 2009.
10. “Nuclear power plant safety concerning external human-induced events” at the Officer of Atoms for Peace (OAEP) in 2009

11. “Decommissioning of nuclear power plant” and “Power policies and technological developments” at the Electricity Generating Authority of Thailand (EGAT) in 2009.

12. “Development of nuclear power technology” at the Electricity Generating Authority of Thailand **(EGAT) in 2008.**

**9. ผลงานทางวิชาการ**

1. R. Kritsananuwat, S. Chanyotha, C. Kranrod, and P. Pengvanich, “Transfer factor of 226Ra, 232Th and 40K from soil to Alpinia Galangal plant grown in northern Thailand,” Journal of Physics: Conference Series, 860 (1), 012008.

2. C. Kranrod, S. Chanyotha, R. Kritsananuwat, T. Ploykrathok, P. Pengvanich, Y. Tumnoi, T. Thumvijit, and S. Sriburee, “ Preliminary survey of radioactivity level in Thai medicinal herb plants” , Journal of Physics: Conference Series, 860, Issue 1, 8 June 2017, Article number 012001.

3. S. Chanyotha, C. Kranrod, P. Pengvanich, and P. Sriploy, “Determination of radon in natural gas pipelines,” Journal of Radioanalytical and Nuclear Chemistry, Volume 307, Issue 3, 1 March 2016, Pages 2095-2099.

4. A. Mardhi, and P. Pengvanich, “Development of computer-based analytical tool for assessing physical protection system,” AIP Conference ProceedingsVolume 1704, 22 January 2016, Article number 020008.

5. S. Chanyotha, C. Kranrod, C. , and P. Pengvanich, “ Systematic approach to characterisation of NORM in Thailand,” Radiation Protection Dosimetry, Volume 167, Issue 1-3, 1 November 2015, Pages 15-21.

6. N. N. Giang and P. Pengvanich, “ Radioisotope Identification Method for Poorly Resolved Gamma-ray Spectrum of Nuclear Security Concern,” American Institute of Physics (AIP) INUSTEC2015 Proceeding, 2015.

7. I. Israngkul-Na-Ayuthaya, S. Suriyapee, P. Pengvanich, “Equivalent Dose from Secondary Neutrons and Scatter Photons in Advance Radiation Therapy Techniques with 15 MV Photon Beams,” Journal of Radiation Protection, 40, 3, 2015, pp. 147-154.

8. I. Israngkul-Na-Ayuthaya, S. Suriyapee, P. Pengvanich, “Evaluation of Equivalent Dose from Neutrons and Activation Products by a 15 MV X-ray LINAC,” Journal of Radiation Research, 5, 56, November 2015, pp. 919-926.

9. S. Chanyotha, C. Kranrod, P. Pengvanich, “Systematic Approach to Characterisation of NORM in Thailand,” Radiation Protection Dosimetry, 2015.

10. E. J. Cruz, B. W. Hoff, P. Pengvanich, Y. Y. Lau, R. M. Gilgenbach, and J. W. Luginsland, "Experiments on Peer-to-Peer Locking of Magnetrons,” Applied Physics Letters, 95, 191503, 2009.

11. P. Pengvanich, Y. Y. Lau, E. Cruz, R. M. Gilgenbach, B. Hoff, and J. W. Luginsland, “Analysis of peer-to-peer locking of magnetrons,” Physics of Plasmas, 15, 103104, October 2008.

12. P. Pengvanich, Y. Y. Lau, J. W. Luginsland, R. M. Gilgenbach, E. Cruz, and E. Schamiloglu, “Effects of frequency chirp on magnetron injection locking,” Physics of Plasmas, 15, 073110, July 2008.

13. P. Pengvanich, D. Chernin, Y. Y. Lau, J. W. Luginsland, and R. M. Gilgenbach, “Effect of random circuit fabrication errors on small signal gain and phase in traveling wave tubes,” IEEE Transactions on Electron Devices, 55, 3, March 2008.

14. N. M. Jordan, Y. Y Lau, D. M. French, R. M. Gilgenbach, and P. Pengvanich, “Electric field and electron orbits near a triple point,” Journal of Applied Physics, 102, 033301, 2007.

15. P. Pengvanich, V. B. Neculaes, Y. Y. Lau, R. M. Gilgenbach, M. C. Jones, W. M. White, and R. D. Kowalczyk, “ Modeling and experimental studies of magnetron injection locking,” Journal of Applied Physics, 98, 114903, December 2005.

16. V. B. Neculaes, P. Pengvanich, Y. Hidaka, Y. Y. Lau, R. M. Gilgenbach, W. M. White, M. C. Jones, H. L. Bosman, and J. W. Luginsland, “ Rapid Kinematic Bunching and Parametric Instability in a Crossed-Field Gap With a Periodic Magnetic Field,” IEEE Transactions on Plasma Science, 33, 2, April 2005.

17. V.B Neculaes, M.C. Jones, R.M. Gilgenbach, Y.Y Lau, J.W. Luginsland, W. White, N.M. Jordan, P. Pengvanich, Y. Hidaka and H. Bosman, "Magnetic Priming Effects on Noise, Startup and Mode Competition in Magnetrons", IEEE Transactions on Plasma Science, Special Issue of Invited Papers from ICOPS 2004, 2005.

18. V. B. Neculaes, M. C. Jones, R. M. Gilgenbach, Y. Y. Lau, J. W. Luginsland, B. W. Hoff, W. M. White, N. M. Jordan, P. Pengvanich, Y. Hidaka, H. L. Bosman, “ Magnetic perturbation effects on noise and startup in DC-operating oven magnetrons,” IEEE Transactions on Electron Devices, Special Issue of IVEC papers, 52, 5, May 2005.

19. M. C. Jones, V. B. Neculaes, W. M. White, Y. Y. Lau, R. M. Gilgenbach, J. W. Luginsland, P. Pengvanich, N. M. Jordan, Y. Hidaka, H. L. Bosman, “ Simulations of magnetic priming in a relativistic magnetron,” IEEE Transactions on Electron Devices, Special Issue of IVEC papers, 52, 5, May 2005.

**10. การนำเสนอผลงาน**

1. S. Chanyotha, P. Pengvanich, H. Haditjahyono, R. Kusumi, and P. Beeley, “Ten years of the Asian network for education in nuclear technology (ANENT),” Conference on Nuclear Training and Education 2015, CONTE 20152015, Pages 16-17.

2. P. Pengvanich, “Roles and Challenges of University in Supporting Nuclear Education and Training in an Emerging Nuclear Energy Country ,” International Conference on Challenges Faced by Technical and Scientific Support Organizations (TSOs) in Enhancing Nuclear Safety and Security, Beijing, China, 27-31 October 2014.

3. C. Thanasupsombat, P. Pengvanich, S. Aootaphao, S. S. Thongvigitmanee, “A postprocessing method for improving contrast and reducing cupping artifacts in lowenergy CBCT images,” Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), Seoul, Republic of Koreal, Oct. 27, 2013 - Nov. 2, 2013.

4. C. Thanasupsombat, P. Pengvanich, S. Aootaphao, and S. S. Thongvigitmanee, “Monte Carlo method for characterization of x-ray scattering in a cone beam computed tomography,” 10th International Conference on Nuclear Analytical Methods in the Life Sciences, Bangkok, Thailand, January 2012.

5. S. Boriboon, P. Pengvanich, and S. Laoharojanapan, “Instrumental neutron activation analysis of aromatic rice based on the K-0 method,” 10th International Conference on Nuclear Analytical Methods in the Life Sciences, Bangkok, Thailand, January 2012.

6. K. Jansee, S. Punnachaiya, P. Pengvanich, “Low-Frequency High-Voltage Power Supply for Non-Thermal Plasma Generation,” 12th Thailand Nuclear Science and Technology Conference, 2011.

7. L. Mitrayon, P. Pengvanich, and S. Punnachaiya, “Development of low-temperature atmospheric dielectric barrier discharge plasma source,” 12th Thailand Nuclear Science and Technology Conference, 2011.

8. P. Pengvanich, Y. Y. Lau, R. M. Gilgenbach, D. Chernin, and J. W. Luginsland, “Effects of random circuit fabrication errors on small signal gain and phase in traveling wave tubes,” 35th IEEE International Conference on Plasma Science, June 2008.

9. P. Pengvanich, Y. Y. Lau, R. M. Gilgenbach, E. J. Cruz, J. W. Luginsland, E. Schamiloglu, “Magnetron Phase Locking: Effects of Frequency Chirp and Locking of Multiple Magnetrons,” 35th IEEE International Conference on Plasma Science, June 2008.

10. E. Cruz, P. Pengvanich, Y. Y. Lau, R. Gilgenbach, J. Luginsland, E. Schamiloglu, “Recent advance in magnetron phase locking: effects of frequency chirps and locking of multiple magnetrons, ” American Physical Society, 49th Annual Meeting of the Division of Plasma Physics, November 2007.

11. P. Pengvanich, D. P. Chernin, Y. Y. Lau, J. W. Luginsland, and R. M. Gilgenbach, “Effect of random circuit fabrication errors on small signal gain and phase in helix traveling wave tubes,” American Physical Society, 49th Annual Meeting of the Division of Plasma Physics, November 2007.

12. P. Pengvanich, Y. Y. Lau, D. Chernin, J. W. Luginsland, and R. M. Gilgenbach, “Effects of circuit manufacturing errors on small signal gain and phase in a traveling wave tube,” 34th IEEE International Conference on Plasma Science, June 2007.

13. P. Pengvanich, Y. Y. Lau, R. M. Gilgenbach, E. J. Cruz, J. W. Luginsland, and E. Schamiloglu, “Recent advances in magnetron phase locking: effects of frequency chirps and locking of multiple magnetrons,” 34th IEEE International Conference on Plasma Science, June 2007.

14. N. M. Jordan, R. M. Gilgenbach, Y. Y. Lau, B. W. Hoff, D. M. French, and P. Pengvanich, “Metal-oxide-junction, triple-point cathodes for high current vacuum electron devices,” American Physical Society, 49th Annual Meeting of the Division of Plasma Physics, November 2007.

15. D. M. French, N. M. Jordan, Y. Y. Lau, R. M. Gilgenbach, and P. Pengvanich, “Electric field and electron orbits near a triple point,” American Physical Society, 49th Annual Meeting of the Division of Plasma Physics, November 2007.

16. N. M. Jordan, R. M. Gilgenbach, Y. Y. Lau, B. W. Hoff, E. J. Cruz, D. M. French, M. R. Gomez, P. Pengvanich, J. Zier, and M. C. Jones, “Metal-oxide-junction, triple-point cathodes for high current vacuum electron devices,” 34th IEEE International Conference on Plasma Science, June 2007.

17. P. Pengvanich, Y.Y. Lau, R.M. Gilgenbach, and J.W. Luginsland, “Modeling and magnetron injection locking characteristics,” American Physical Society, 48th Annual Meeting of the Division of Plasma Physics, October 2006.

18. P. Pengvanich, Y. Y. Lau, D. Chernin, J. W. Luginsland, and R. M. Gilgenbach, “Effect of random geometric perturbations on slow wave devices,” 33th IEEE International Conference on Plasma Science, June 2006.

19. P. Pengvanich, Y.Y. Lau, R.M. Gilgenbach, V.B. Neculaes, M.C. Jones, W.M. White, R.D. Kowalczyk, “Modeling and magnetron injection locking characteristics,” 33th IEEE International Conference on Plasma Science, June 2006.

20. R.M. Gilgenbach, B.W. Hoff, Y.Y. Lau, N.M. Jordan, E. Cruz, P. Pengvanich, W. White, T.A. Spencer, D. Price, “Enhanced performance of a relativistic magnetron by magnetic priming,” American Physical Society, 48th Annual Meeting of the Division of Plasma Physics, October 2006.

21. P. Pengvanich, D. Chernin, Y. Y. Lau, R. M. Gilgenbach, and D. Dialetis, “Effects of Random Geometrical Perturbations in Slow Wave Devices,” American Physical Society, 47th Annual Meeting of the Division of Plasma Physics, October 2005.

22. P. Pengvanich, V. B. Neculaes, Y. Y. Lau, R. M. Gilgenbach, M. C. Jones, W. M. White, and R. D. Kowalczyk, "Modeling and Experimental Studies of Magnetron Injection Locking," 32nd IEEE International Conference on Plasma Science, June 2005.

23. P. Pengvanich, V. B. Neculaes, Y. Hidaka, Y. Y. Lau, R. M. Gilgenbach, W. White, M. C. Jones, H. Bosman, J. W. Luginsland, "Rapid Kinematic Bunching and Parametric Instability in a Crossed-Field Gap with a Periodic Magnetic Field," American Physical Society, 46th Annual Meeting of the Division of Plasma Physics, November 2004.

24. W. M. White, R. M. Gilgenbach, M. C. Jones, V. B. Neculaes, Y. Y. Lau, N. Jordan, P. Pengvanich, R. Edgar, B. Hoff, T. A. Spencer, D. Price, "RF Priming Experiments and Simulations of Magnetic Priming in Relativistic Magnetrons," American Physical Society, 46th Annual Meeting of the Division of Plasma Physics, November 2004.

25. V. Neculaes, R. M. Gilgenbach, Y. Y. Lau, M. C. Jones, J. Luginsland, W. White, P. Pengvanich, N. M. Jordan, Y. Hidaka, H. Bosman, "Magnetron microwave noise reduction and magnetic priming by azimuthally varying axial magnetic fields," 5th IEEE International Vacuum Electronics Conference, April 2004

26. M. C. Jones, V. B. Neculaes, W. White, Y.Y. Lau, R. M. Gilgenbach, J. Luginsland, P. Pengvanich, N.M. Jordan, Y. Hidaka, H. Bosman, "Simulation of rapid startup in microwave magnetrons with azimuthally-varying axial magnetic fields," 5th IEEE International Vacuum Electronics Conference, April 2004

27. V. B. Neculaes, R. M. Gilgenbach, M. Lopez, Y. Y. Lau, M. Jones, W. White, P. Pengvanich, M. D. Johnson, T. Strickler, T. A. Spencer, J. Luginsland, M. Haworth, K. Cartwright, P. Mardahl, T. Murphy, and D. Price, "Mode Competition in Relativistic Magnetron and Injection Locking in KW Magnetrons," American Institute of Physics Conference Proceeding, Vol. 691, Issue 1, p301, December 2003.

28. V. B. Neculaes, R.M. Gilgenbach, Y.Y. Lau, M.C. Jones, W.M. White and P. Pengvanich, “Microwave Noise Reduction Experiments in kW Magnetrons,” American Physical Society, 45th Annual Meeting of the Division of Plasma Physics, 2003.

29. R. M. Gilgenbach, V. B. Neculaes, M. Lopez, M. Jones, W. White, Y. Y. Lau, P. Pengvanich, M. D. Johnston, T. Strickler, T. A. Spencer, J. Luginsland, M. Haworth, K. Cartwright, P. Mardahl, T. Murphy, D. Price, 4th IEEE International Vacuum Electronics Conference, May 2003.