Chapter 5:
Particle Dynamics and Solid Body Dynamics

Topics:

Part I: particle goal, particle location, particle render type, particle weighting, play back option, particle display size, velocity, and goal smoothness

Part II: connect dynamic, collision, resilience, friction, offset, gravity, magnitude, attenuation, and key-frame

Chapter 5:
Particle Dynamics and Solid Body Dynamics

เอกสารประกอบการเรียน
รายวิชา ANI 951301
สาขาวิชาแอนนิเมชันและเกม
วิทยาลัยศิลปะ สื่อ และเทคโนโลยี
มหาวิทยาลัยเชียงใหม่

951301: Advanced Animation and Rendering Techniques

วัตถุประสงค์

1. สามารถสร้างและควบคุม particles ด้วย MEL scripts ได้
2. เข้าใจถึงการตั้ง particle goals ให้กับ particles เพื่อสร้างความสัมพันธ์ระหว่าง particles ได้
3. สามารถปรับประเภทและค่าคุณลักษณะให้กับ particles ให้สอดคล้องกับความต้องการ
4. เรียนรู้เรื่องของ interaction ระหว่าง particles กับ surfaces ของวัตถุ ด้วย solid body dynamics
5. ใช้การผสมผสานการเคลื่อนที่ของ particles และการเคลื่อนที่ของ solid body dynamics ให้เกิดผลตามต้องการได้

เนื้อหาการสอน

ส่วนที่ 1: การตั้งจุดมุ่งหมายให้กับ, การระบุตำแหน่ง, การระบุรูปแบบการแสดงผล, การกำหนดค่าน้ำหนัก, การกำหนดการจำลองผล, การกำหนดขนาดการแสดงผล, การกำหนดอัตราเร่ง, และการประมวลผลแบบนุ่มนวลให้กับ particles

ส่วนที่ 2: การเชื่อมโยงความสัมพันธ์ทางกายภาพ, การกำหนดการปะทะ, การกำหนดค่าแรงเฉื่อย, การกำหนดค่าการสูญเสียแรง, การกำหนดแรงโน้มถ่วง, แรงดึง, และการประมวลผลให้กับ particles

Particle Dynamics

เนื้อหาในบทนี้เป็นการใช้คำสั่ง MEL scripts ในการควบคุมการทำงานของ particle dynamics โดยเริ่มจากพื้นฐานการสร้าง particles และ เรียนรู้ประโยชน์จากการตั้ง goals ให้กับ particles การตั้ง goals นั้นเราสามารถกำหนดการเคลื่อนที่ของ particles (particle motions) ในขณะที่เรายังสามารถใช้คุณสมบัติ fields และ collisions ได้อยู่ โดยที่เราไม่ต้องไปแทรกแซงการทำงานของ particle dynamics system แต่ goals จะส่งผลให้ particles มีพฤติกรรมเฉพาะบางอย่างในขณะที่ทำงานอยู่ภายใต้เงื่อนไขของ particle dynamics system นั่นเอง เราลองมาทำความเข้าใจถึงการตั้ง goals ให้กับ particles ใน workshop 1 ด้านล่าง

Workshop 1

เพื่อให้นักศึกษาเข้าใจถึงการใช้ particle goals แบบทดลองอันแรกถูกออกแบบด้วยแรงบันดาลใจจากการความสัมพันธ์ของหมู่ดาวที่โคจรรอบกันและกัน (orbiting) เช่นในระบอบสุริยะจักรวาลของเรานั่นเอง ลองจินตนาการถึงความสัมพันธ์อันนี้ ดาวแต่ละดวงมีค่า position, velocity และ acceleration ที่เกิดจากความสัมพันธ์กับค่าดังกล่าวของหมู่ดาวรอบๆ เริ่มแรกดาวแต่ละดวงจะมีค่าเหล่านี้เป็นของตัวเอง แต่เนื่องจากค่า gravity ในดวงดาวแต่ละดวงทำให้เมื่อมันโคจรเข้าใกล้กัน ทิศทาง ความเร็ว อัตราเร่ง ของมันจะถูกเปลี่ยนแปลงไป ในการทำความเข้าใจ particles ใน Maya ก็เช่นกัน เราสามารถตั้งค่า position, velocity และ acceleration ให้กับ particle ที่เราสร้างขึ้นได้ การตั้ง goal เปรียบเสมือนการตั้งค่า gravity ให้เกิดความสัมพันธ์ระหว่าง particles ขึ้น ในแบบทดลองนี้เราจะลองมาจำลองการโคจรของดวงดาวกัน

เริ่มจากการสร้าง particles ขึ้นมาสองอันก่อน โดยให้ตั้งชื่ออันแรกว่า Earth และอันที่สองชื่อว่า Sun โดยให้ particles ทั้งสองอยู่ห่างกันดังตัวอย่าง

```
// create 2 particles at different location
particle -p -5 0 5 -name Earth;
particle -p 5 0 -5 -name Sun;
```

คำสั่ง particle ใช้ในการสร้าง particle ส่วน flag –p ย่อมาจาก position คือตำแหน่งแบบ vector X,Y,Z ที่เราต้องการสร้าง ตามด้วย flag –name เพื่อตั้งชื่อ ในที่นี้ให้สร้าง Earth ที่ X = -5, Y = 0, Z = 5 และ Sun ที่ X = 5, Y = 0, Z = -5

951301: Advanced Animation and Rendering Techniques

เมื่อเรา execute คำสั่งเราจะยังไม่เห็นตัว particles ทั้งสองใน workspaces เนื่องจากเรายังไม่ได้กำหนดค่า display ให้กับ particles ที่สร้าง ขั้นต่อไปเราจะมากำหนดค่า display ให้กับมันดังนี้

```
// change render type
setAttr EarthShape.particleRenderType 4; // เปลี่ยนค่า display ให้ particle แรกด้วยคำสั่งแบบเต็ม
setAttr SunShape.prt 4; // เปลี่ยนค่า display ให้ particle ที่สองด้วยคำสั่งแบบย่อ
```

จากคำสั่งด้านบนเราสั่งเปลี่ยนการ display ให้กับ particle Earth โดยการทำงานกับรูปทรงของ Earth ต้องสั่งไปที่ EarthShape ส่วน particleRenderType เป็นคำสั่งในการกำหนด display โดยสามารถตั้งค่าได้กำหนดค่าได้จาก 1 -9 โดย 4 ที่เราใช้หมายถึงรูปทรง sphere นั่นเอง โดย particleRenderType มีรูปย่อคือ prt

เมื่อ execute คำสั่งเราจะได้ particles สองอันดังภาพตัวอย่าง

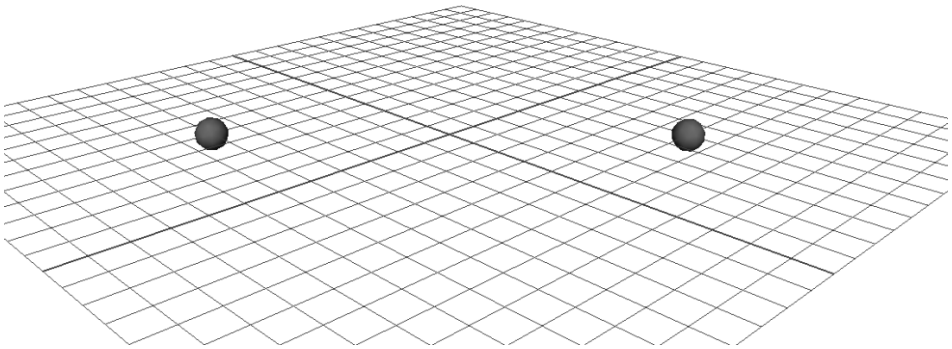

Fig 05-01: ตัวอย่าง particles ทั้งสองที่สร้างขึ้น

Particles ที่ถูกสร้างขึ้นมานั้นไม่มีความสัมพันธ์กัน ขั้นต่อไปเราจะตั้ง goal เพื่อสร้างความสัมพันธ์ให้กับ particles ทั้งสอง ในที่นี้เราจะสั่งให้ Earth เคลื่อนที่เข้าหา Sun ด้วยกำลังดึงที่ 0.3 ดังตัวอย่าง

```
// make the second follows the first
goal -goal Sun -weight 0.3 Earth;
```

เราใช้คำสั่ง goal บอกว่าทำงานที่ Sun โดยให้มีค่าแรงดึงดูด = 0.3 กระทำต่อ Earth เราสามารถปรับค่าแรงดึงต่อกันได้ที่ flag –weight โดยมีค่าสูงสุดที่ 1 และต่ำสุดที่ 0

ถ้าเราอยากดูผลลัพธ์ที่เกิดขึ้น เราต้อง playback animation ดู แต่เนื่องจากตอนนี้เรามี จำนวนเฟรมน้อยไป เราสามารถเพิ่ม playable frames ได้ด้วย MEL จากคำสั่ง

951301: Advanced Animation and Rendering Techniques

```
// set playback time to 2000
playbackOptions -min 1 -max 2000;
```

playbackOption จะทำงานกับจำนวนเฟรมในการเล่น โดยใช้ flag –min กำหนดเฟรมแรก และ flag –max ในการกำหนดเฟรมสุดท้าย ในที่นี้เราตั้งให้มีทั้งหมด 2000 เฟรม

หลังจาก execute คำสั่งให้ลอง play animation จะพบว่า Earth จะเคลื่อนที่เข้าหา Sun แต่เนื่องจากอัตราเร่งทำให้มันเลย Sun ไปแต่จะถูกดึงกลับมา จะเป็นเช่นนี้จนกว่าสุดท้าย Earth จะมาหยุดที่ Sun พอดี

จากนั้นให้ลองตั้งค่าสีให้กับ particles ทั้งสอง โดยให้ Sun มีสีแดงและ Earth มีสีฟ้า ดังตัวอย่าง

```
// add colour
addAttr -longName colorBlue -defaultValue 1
-attributeType double EarthShape;
addAttr -ln colorRed -dv 1 -at double SunShape;
```

เราสามารถทำได้ด้วยคำสั่ง addAttr (add attribute) ตามด้วย flag –longName (ln) เพื่อกำหนดค่าสีแบบชื่อเต็ม Maya ใช้ภาษาอังกฤษแบบ American English จึงสั่งเปลี่ยนสีด้วยคำสั่ง colorBlue เพื่อสร้างสีฟ้า และ colored เพื่อสร้างสีแดง โดยที่ flag –defaultValue (dv) เป็นการกำหนด shade สีให้กับสีที่เราเลือก ยิ่งมีค่าใกล้ 0 สีจะยิ่งมืด

เมื่อทำตามขั้นตอนแล้วจะพบว่า particles Earth และ Sun มีสีน้ำเงินและแดงตามลำดับ ขั้นต่อไปเราจะกำหนดขนาดให้กับ particles ทั้งสอง โดยที่ Sun ควรจะมีขนาดใหญ่กว่า Earth

```
// change particles display sizes
addAttr -ln radius -dv 0.5 EarthShape;
addAttr -ln radius -dv 1 SunShape;
```

เราสามารถกำหนดขนาดให้กับ particles ได้จาก addAttr ตามด้วย flag –longName radius เพื่อบอกว่าจะ modify ค่ารัศมี particle ตามด้วย flag –dv บอกขนาด ในที่นี้ให้ Earth มีขนาด = 0.5 และ Sun = 1.0

ทั้งหมดที่เราทำมาคือการสร้างความสัมพันธ์ (set goals) ให้กับ particles ทั้งสอง โดยที่เรายังไม่ได้กำหนดค่าเคลื่อนที่หลักใดๆให้กับ particles ทั้งสองเลย อย่างที่กล่าวไว้แล้วข้างต้นว่าทั้งสองค่านั้นแยกออกจากกันแต่จะส่งผลซึ่งกันและ ขั้นต่อไปเราจะลองมากำหนด velocity ให้กับ particles ทั้งสองดู แต่ก่อนที่เราจะทำ ให้กลับไปแก้ค่าแรงดึงดูดระหว่าง Sun กับ Earth ใหม่จาก 0.3 ไปเป็น 0.1 เพื่อให้เห็นผลของความสัมพันธ์ได้ชัดขึ้น

951301: Advanced Animation and Rendering Techniques

```
// make the second follows the first
goal -goal Sun -weight 0.1 Earth;
```

จากนั้นกำหนด velocity ให้กับ Earth และ Sun ด้วยคำสั่ง particle ดังตัวอย่าง

```
// add velocity
particle -edit -id 0 -attribute velocity
-vectorValue 10 0 0 Earth;
particle -e -id 0 -at velocity -vv 0 -0.1 0 Sun;
```

flag –attribute (-at) ตามด้วยชื่อ attribute เป็นการกำหนดว่า attribute ตัวไหนของ particle ที่เราต้องการ modify ในที่นี้คือค่า velocity จากนั้นตามด้วย flag –vectorValue (-vv) เพื่อบอกน้ำหนักกับทิศทางของแรงจากแนวแกน X, Y, Z ในที่นี้ Earth จะมีแรงเคลื่อนขึ้นตามแนวแกน X = 10 และ Sun จะเคลื่อนตามแนว –Y ด้วยแรง 0.1

ขั้นตอนสุดท้ายคือการทำให้วงโคจรมีความนุ่มนวลคล้ายวงโคจรของดวงดาวมากขึ้นขึ้น สามารถทำได้โดยการ smooth ค่า goal ดังนี้

```
// smoothen the orbit
setAttr Earth.goalSmoothness 4.0;
setAttr Sun.goalSmoothness 4.0;
```

เมื่อลอง execute คำสั่งดูจะพบว่า Sun ค่อยๆเคลื่อนที่ลงด้านล่างอย่างช้าๆ ในขณะที่ Earth ก็จะโคจรรอบ Sun โดยที่จะเคลื่อนที่ตาม Sun ลงด้านล่างไปด้วยเช่นกัน นี่คือตัวอย่างของการใช้ particle goals สร้างความสัมพันธ์เลียนแบบ โลกที่โคจรรอบพระอาทิตย์ ในขณะนี้เราจะพบว่าเรายังขาดพระจันทร์ที่ควรจะโคจรรอบโลกอยู่ ให้นักศึกษานำความรู้ที่ได้ เพิ่มเติมพระจันทร์ให้โคจรรอบโลกขึ้นมาด้วยดังภาพตัวอย่าง

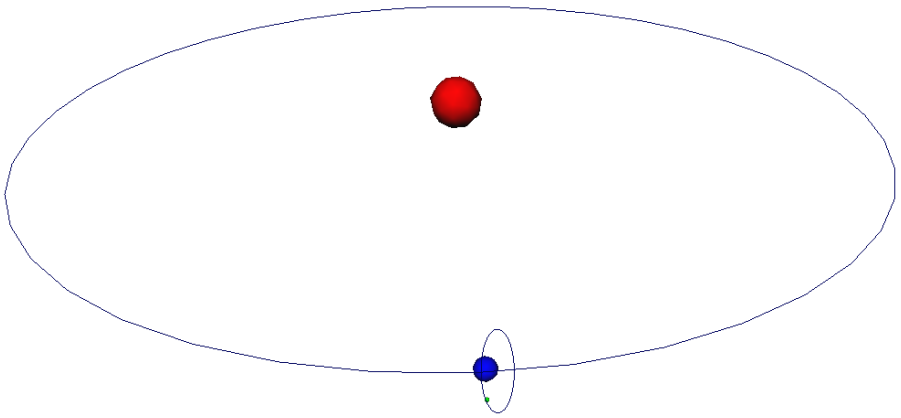

Fig 05-02: การจำลองวงโคจรของโลกรอบดวงอาทิตย์ และดวงจันทร์โคจรรอบโลก

951301: Advanced Animation and Rendering Techniques

คำสั่งทั้งหมดรวมพระจันทร์ในตอนท้ายมีดังนี้

```
// create 2 particles at different location
particle -p -5 0 5 -name Earth;
particle -p 5 0 -5 -name Sun;

// change render type
setAttr EarthShape.particleRenderType 4;
setAttr SunShape.prt 4;

// make the second follows the first
goal -goal Sun -weight 0.1 Earth;

// set playback time to 2000
playbackOptions -min 1 -max 2000;

// add colour
addAttr -longName colorBlue -defaultValue 1
-attributeType double EarthShape;
addAttr -ln colorRed -dv 1 -at double SunShape;
// change particles display sizes
addAttr -ln radius -dv 0.5 EarthShape;
addAttr -ln radius -dv 1 SunShape;


// add velocity
particle -edit -id 0 -attribute velocity
-vectorValue 10 0 0 Earth;
particle -e -id 0 -at velocity -vv 0 -0.1 0 Sun;

// smooth the orbit
setAttr Earth.goalSmoothness 4.0;
setAttr Sun.goalSmoothness 4.0;

// create moon orbit around the earth
particle -p -3 0 3 -name Moon;
setAttr MoonShape.prt 4;
goal -goal Earth -weight 0.275 Moon;
addAttr -ln colorGreen -dv 1 -at double MoonShape;
addAttr -ln radius -dv 0.1 MoonShape;
particle -e -id 0 -at velocity -vv 1 0 0 Moon;
setAttr Moon.goalSmoothness 4.0;
```


951301: Advanced Animation and Rendering Techniques

Solid Body Dynamics

การใช้งาน solid body dynamic นั้น เพื่อช่วยในขั้นตอนการทำแอนนิเมชันที่มีความซับซ้อนสูง เนื่องจากสามารถช่วยจัดการกับ particles ในการ collision ได้ ซึ่งเราจะเรียนรู้จากการประยุกต์ใช้ใน workshop 2 และ 3

Workshop 2

ในการทดลองนี้เราจะมาเรียนรู้เรื่องของ interaction ของ particles กับ surfaces ของวัตถุที่เราสร้าง เราสามารถกำหนด particle ให้กระทบและมีผล reaction กับพื้นผิวใดพื้นผิวหนึ่ง หรือมีผลกับหลายๆพื้นผิวในเวลาเดียวกันได้ โดยสามารถทำงานได้กับทั้งพื้นผิวแบบ NURBS และ polygon ใน workshop 2 นี้เราจะมาเริ่มด้วยตัวอย่างง่ายๆจากการจำลองของเหลวให้ไหลไปกระทบพื้นผิวหนึ่ง แล้วไหลต่อไปยังพื้นผิวที่สองดังตัวอย่างด้านล่างกัน

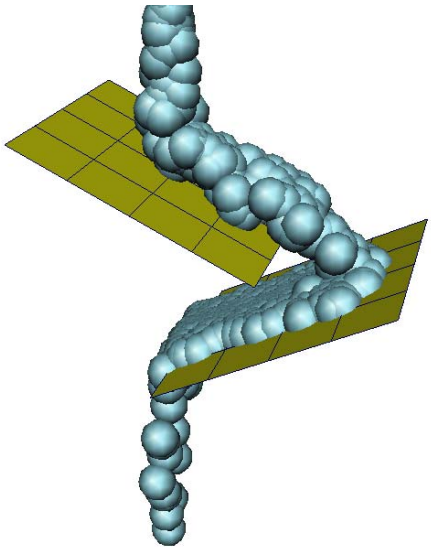

Fig 05-03: ตัวอย่างการจำลอง particles ให้มีพฤติกรรมเช่นเดียวกับของเหลวที่ไหลจากที่สูงลงสู่ที่ต่ำตามแรงโน้มถ่วงของโลก

ขั้นตอนแรกเรามาเริ่มจากการสร้างพื้นผิวขึ้นมาก่อน ในที่นี้เราจะใช้ polygon plane ในการทำงาน เราสามารถสร้าง plane ได้ด้วยคำสั่ง polyPlane ดังตัวอย่าง

```
// create plane
polyPlane -w 1 -h 1 -sx 4 -sy 4 -ax 0 1 0 -name Floor;
scale 6 6 6;
```

เนื่องจาก polyPlane ที่เราสร้างขึ้นมีขนาดเล็กเกินไป เราจึงปรับขนาดของมันให้ใหญ่ขึ้นด้วยคำสั่ง scale โดยให้มีขนาดใหญ่ขึ้น 6 เท่าในแนวแกน X, Y, Z โดยให้มีชื่อว่า Floor ดังตัวอย่างข้างบน

จากนั้นเราจะสั่งสร้างตัว emitter ขึ้นมาเพื่อผลิต particles ให้กับฉากของเรา โดยที่เราสามารถ flag เพื่อกำหนดค่าต่างๆให้กับ emitter ที่สร้างขึ้นมาได้ เช่น –position เพื่อกำหนดตำแหน่งของมัน –type เพื่อกำหนดประเภทของ emitter รวมถึงค่าการ generate ความเร็ว และความกระจายตัวของ particles ที่จะถูก generate ขึ้นมาได้อีกด้วย ดังตัวอย่าง

```
// create particles
string $eObject[] = `emitter -position 0 4 0 -type direction
-name Emit -rate 30 -speed 1 -spread 0.2 -dx 0 -dy -1.0 -dz
0`;
string $pObject[] = `particle`;
connectDynamic -em $eObject[0] $pObject[0];
```

เพื่อให้เห็นการทำงานของ particles ได้อย่างชัดเจน เราควรเพิ่มเวลา playback time เป็น 500 frames ซึ่งสามารถทำได้ดังนี้

```
playbackOptions -min 1 -max 500;
```

เมื่อลอง execute ทดสอบคำสั่งดูจะพบว่า emitter ชื่อ Emit ที่เราสร้าง ปล่อย particles ออกมามากมาย โดยที่ particles ทั้งหมดจะลอยออกมาทะลุพื้นผิวที่เราสร้างไว้ เนื่องจากเรายังไม่ได้กำหนดค่า collision ให้กับพื้นผิว

951301: Advanced Animation and Rendering Techniques

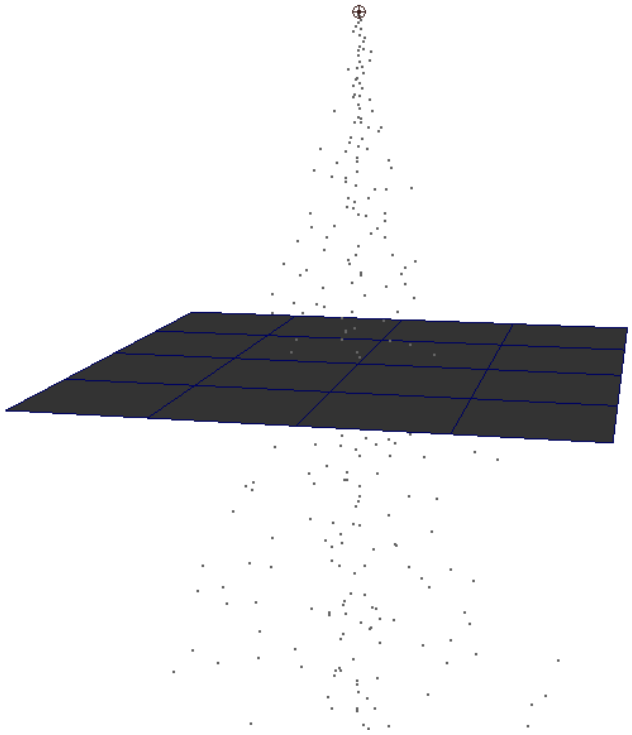

Fig 05-04: แสดงการที่ emitter ปล่อย particles ออกมาผ่านพื้นผิวที่ยังไม่มีการกำหนด collision

Particles ลอยทะลุพื้นผิวเสมือนไม่มีสิ่งใดกีดขวางอยู่ ขั้นตอนต่อไปเราจะมากำหนดการ collision ให้พื้นผิวเป็น solid body กัน ก่อนอื่นเราต้องสร้างค่าการกระทบของพื้นผิวที่เราต้องการขึ้นมาก่อนด้วยคำสั่ง collision ซึ่งสามารถกำหนดคุณสมบัติได้ด้วย flag –resilience, -friction และ –offset ตามด้วยชื่อพื้นผิวที่เราต้องการ ในที่นี้คือ Floor

```
// make the floor solid
collision -resilience 1.0 -friction 0.0 -offset 0.0 Floor;
```

เมื่อกำหนดให้ Floor เป็น solid body แล้ว ขั้นต่อไปคือการ connect พื้นผิวนี้เข้ากับ particle ที่ถูกสร้างขึ้นมา สามารถทำได้ดังนี้

```
connectDynamic -collisions Floor particle1;
```

จากนั้นให้ทดลอง execute คำสั่งจะพบว่า particles ทั้งหมดจะตกลงมากระทบพื้น แล้วสะท้อนย้อนกลับออกไปดังภาพตัวอย่างด้านล่าง ซึ่งเหมือนเป็นวัตถุที่ไร้น้ำหนัก ไม่ตรงกับคุณลักษณะของของเหลวที่เราต้องการสร้างขึ้น

951301: Advanced Animation and Rendering Techniques

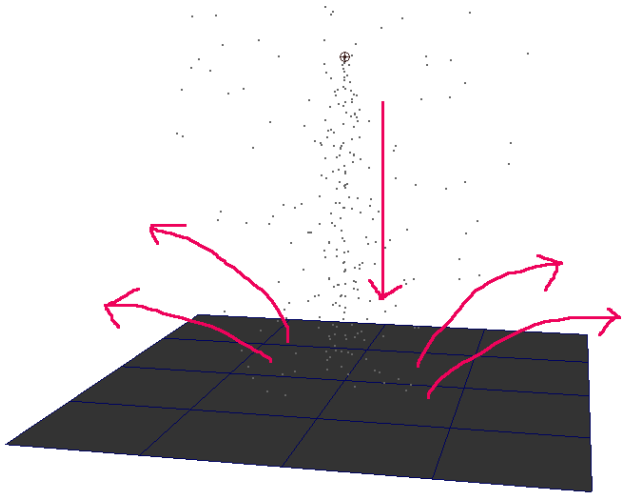

Fig 05-05: เมื่อเรากำหนดค่า collision ให้กับพื้นผิว particles ที่ตกกระทบพื้นผิวจะสะท้อนกลับ และไม่สามารถทะลุผ่านพื้นผิวไปได้

ขั้นต่อไปเราจะปรับค่าของ particles ให้เมื่อตกกระทบพื้นผิวแล้วไม่สะท้อนกลับไป ให้มีลักษณะเสมือนของหนักหรือของเหลวมากขึ้น ซึ่งสามารถทำได้โดยการปรับลดค่าของ resilience และเพิ่มค่า friction ขึ้น สามารถทำได้โดยเพิ่มคำสั่งด้านล่างเข้าไป

```
// make particles not to bounce
setAttr geoConnector1.resilience 0.0; // ปรับให้ไม่เด้ง
setAttr geoConnector1.friction 0.2; // ปรับให้กระจายออก
```

เมื่อทดลอง execute คำสั่งอีกครั้งหนึ่ง จะพบผลลัพธ์ที่ particles เมื่อตกกระทบพื้นผิวแล้วจะไหลกระจายออกไปทางด้านข้างตามแนวพื้นผิวตามที่เราต้องการ

951301: Advanced Animation and Rendering Techniques

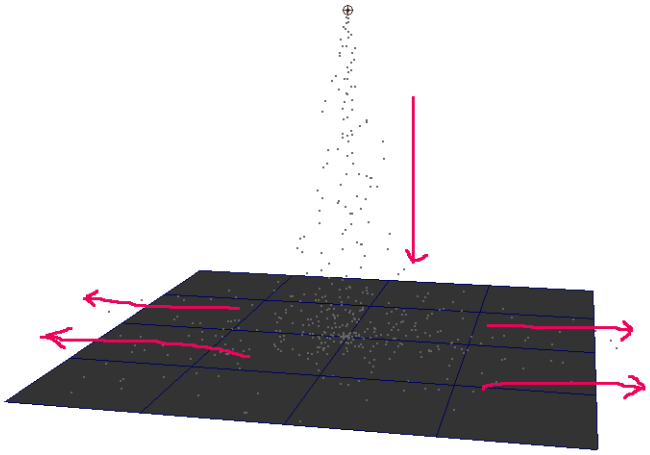

Fig 05-06: เมื่อเรากำหนดค่า resilience และ friction เป็น 0.0 และ 0.2 พฤติกรรมของ particles จะเปลี่ยนไปตามภาพตัวอย่าง

เมื่อได้คุณลักษณะของการ collision เป็นที่ต้องการแล้ว ขั้นต่อไปเราจะปรับพื้นผิวให้เอื้อต่อทิศทางการไหลของ particles โดยเราจะปรับระนาบของพื้นผิวให้มีความลาดชัน 26 องศา เราสามารถทำได้หลายวิธี ในที่นี้ขอใช้ตัวอย่างของคำสั่ง xfrom ในการปรับแต่ง

```
// rotate the floor
xform -rotation -26 0 0 Floor;
```

คำสั่ง xform จะใช้ในการปรับแต่งค่าการเคลื่อนย้ายวัตถุในแบบต่างๆ flag –rotation เป็นการกำหนดปรับแต่งค่าด้วยการหมุน ตามด้วยองศาที่ต้องการตามแนวแกน X, Y, Z จากนั้นตามด้วยชื่อวัตถุที่ต้องการ modify ในที่นี้เราสั่งให้หมุนตามแนวแกน X = 26 องศา ที่วัตถุชื่อว่า Floor

ให้ทดลอง execute คำสั่งเพื่อตรวจสอบว่า particles ไหลไปตามทิศทางที่เราต้องการหรือไม่ ถ้าทุกอย่างถูกต้อง เราจะเริ่มขั้นตอนต่อไปที่การสร้างระนาบที่สอง โดยวิธีที่ง่ายที่สุดคือการ duplicate ขึ้นมาจากระนาบที่เรามีอยู่แล้ว

```
// create a second floor
duplicate -name Floor1 Floor;
xform -translation 0 -3 -3 Floor1;
xform -rotation 26 0 0 Floor1;
```

จากคำสั่งด้านบน เราสั่ง duplicate วัตถุขึ้นมาใหม่โดยตั้งชื่อว่า Floor1 โดย duplicate ขึ้นมาจากวัตถุที่ชื่อ Floor แล้วเราใช้คำสั่ง xform Flag –translation เพื่อย้ายวัตถุลงไปและเคลื่อนออกไปทางขวา 3 units แล้ว xform flag –rotation ด้วยค่าตรงข้ามกับค่าของ Floor แรก เพื่อให้ระนาบทั้งสองเอียงเข้าหากันในลักษณะของการ mirror วัตถุ

951301: Advanced Animation and Rendering Techniques

เมื่อเสร็จแล้วทดลอง execute คำสั่งจะพบว่าเมื่อ particles ไหลผ่านระนาบแรกได้อย่างถูกต้อง แต่เมื่อตกลงมายังระนาบสองจะลอยทะลุพื้นผิวของระนาบสองออกไป ทั้งนี้เนื่องจากเรายังไม่ได้กำหนด solid body ให้กับระนาบสอง รวมถึงยังไม่ได้ connect ระนาบเข้ากับ particles นั่นเอง ขั้นตอนต่อไปเราจะมาทำทั้งสองขั้นตอนนี้กัน โดยใช้ค่าเดียวกับระนาบแรก

```
// create second geoConnector to new floor
collision -resilience 0 -friction 0.2 Floor1;
connectDynamic -collisions Floor1 particle1;
```

เมื่อเรากำหนด solid body ให้กับระนาบที่สองแล้วลอง execute คำสั่งดูจะพบปัญหาต่อไป นั่นคือเมื่อ particles ไหลจากระนาบแรกไปตกกระทบยังระนาบที่สองแล้ว แทนที่ particles จะไหลลงมาตามความลาดชันลงสู่พื้น มันกลับไหลย้อนขึ้นไปทางด้านบนของระนาบสองดังภาพตัวอย่าง

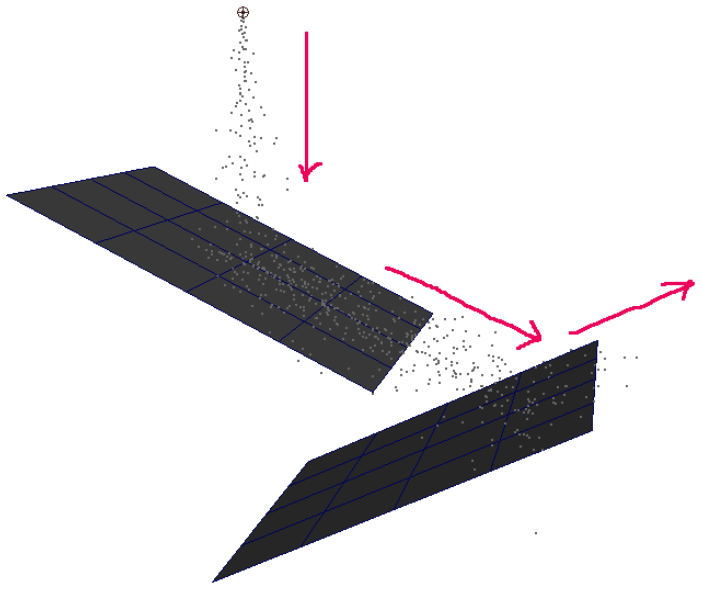

Fig 05-07: เมื่อ particles ตกกระทบพื้นผิวแบบ solid body แล้วจะสะท้อนไปยังพื้นผิวข้างเคียง แต่ยังมีพฤติกรรมที่ผิดธรรมชาติ

ทั้งนี้เนื่องจากเรายังไม่ได้กำหนด gravity field ให้กับ particles ของเรานั่นเอง เราสามารถสร้าง gravity filed ขึ้นมาได้ด้วยคำสั่ง gravity ตามด้วย flag –magnitude เพื่อกำหนดค่าแรงดึงดูด ในที่นี้กำหนดให้มีค่า = 1 จากนั้นอย่าลืม connect ตัว gravity นี้เข้ากับ particles ของเราด้วยคำสั่ง connectDynamic ตามด้วย flag –field เพื่อระบุตัว field ที่ต้องการเชื่อมด้วย ตามด้วย particles ที่ต้องการเชื่อมกับ field นั้นๆ ดังตัวอย่างถัดไป

951301: Advanced Animation and Rendering Techniques

```
// add gravity field to particle
gravity -magnitude 1 -attenuation 0.0;
connectDynamic -fields gravityField1 particle1;
```

เมื่อเราทดลอง execute คำสั่งจะพบว่า particles ไหลจากระนาบที่หนึ่งลงมายังระนาบที่สอง จากนั้นไหลลงสู่พื้นด้านล่างตามที่เราต้องการ แต่เนื่องจาก particles ที่แสดงอยู่มีคุณลักษณะไม่เหมาะสมกับของเหลวที่เราต้องเรา เราสามารถปรับแต่งได้จากการแสดงค่ารูปทรงของ particles ด้วยคำสั่ง particleRenderType โดยให้ตั้งค่าเป็น 4 แล้วปรับขนาดของ radius เป็น 0.2 ดังตัวอย่างหน้าถัดไป

```
// set particle shape to sphere
setAttr particle1.particleRenderType 4;
addAttr -longName radious -defaultValue 0.2 particleShape1;
```

คำสั่งทั้งหมดของ workshop 2 มีดังนี้

```
// create plane
polyPlane -w 1 -h 1 -sx 4 -sy 4 -ax 0 1 0 -name Floor;
scale 6 6 6;

// create particles
string $eObject[] = `emitter -position 0 4 0 -type direction
-name Emit -rate 30 -speed 1 -spread 0.2 -dx 0 -dy -1.0 -dz
0`;
string $pObject[] = `particle`;
connectDynamic -em $eObject[0] $pObject[0];

// set playback time to 500
playbackOptions -min 1 -max 500;

// make the floor solid
collision -resilience 1.0 -friction 0.0 -offset 0.0 Floor;
connectDynamic -collisions Floor particle1;

// make particles not to bounce
setAttr geoConnector1.resilience 0.0;
setAttr geoConnector1.friction 0.2;

// rotate the floor
xform -rotation -26 0 0 Floor;

// create a second floor
duplicate -name Floor1 Floor;
xform -translation 0 -3 -3 Floor1;
xform -rotation 26 0 0 Floor1;

// create second geoConnector to new floor
collision -resilience 0 -friction 0.2 Floor1;
```


951301: Advanced Animation and Rendering Techniques

```
connectDynamic -collisions Floor1 particle1;

// add gravity field to particle
gravity -magnitude 1 -attenuation 0.0;
connectDynamic -fields gravityField1 particle1;

// set particle shape to sphere
setAttr particle1.particleRenderType 4;
addAttr -longName radious -defaultValue 0.2 particleShape1;
```

Workshop 3

ในหัวข้อนี้เราจะนำความรู้ที่ได้เรียนมาจาก workshop 1 และ 2 มาใช้สร้างแอนนิเมชันที่มีการผสมผสานในการตั้งค่าความเคลื่อนไหวให้กับวัตถุ โดยรวมการใช้ particle การใช้ solid body dynamics และการ set keyframe แบบปกติเข้าด้วยกัน เพื่อให้นักศึกษาเข้าใจถึงความสัมพันธ์และสามารถนำไปประยุกต์ใช้ ช่วยในการสร้างแอนนิเมชันของตนเองต่อไป

ในแบบทดลองนี้เราจะสร้างถังขึ้นมาสามถัง แล้วเท particles ใส่ลงไปบนถังใบบนสุด เมื่อถังเต็มเราจะบังคับให้ถังเท particles ลงไปยังถังที่สองและสามตามลำดับ

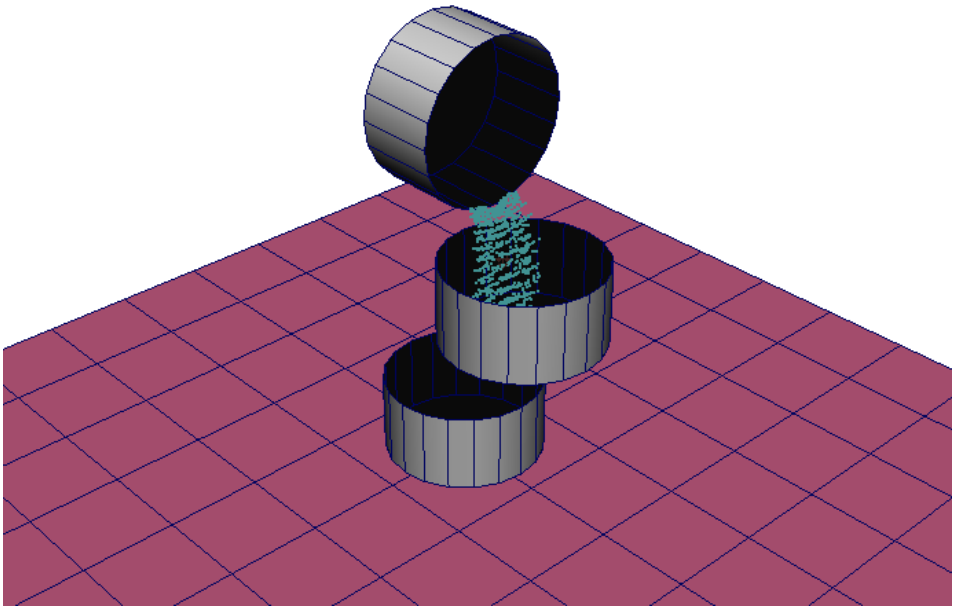

Fig 05-08: ภาพตัวอย่างการจำลองพฤติกรรมของ particles ให้เหมือนเมล็ดข้าว ที่ถูกเทจากถังหนึ่งไปยังอีกถังหนึ่ง

951301: Advanced Animation and Rendering Techniques

เราจะเริ่มจากการเตรียมสถานที่กันก่อน ให้สร้างพื้น polygon ขึ้นมา ด้วยคำสั่ง polyPlane ให้มีขนาด กว้าง = 12, ยาว = 12 ให้ตั้งชื่อว่า Floor

```
// create floor
polyPlane -w 12 -h 12 -name Floor;
```

จากนั้นให้สร้าง polygon cylinder ขึ้นมา โดยกำหนดให้มีรัศมี = 1, ความสูง = 1, subdivisions X = 20, subdivisions Y = 1, subdivisions Z = 0, subdivisions caps = 0 โดยให้ตั้งชื่อว่า BucketA

```
polyCylinder -r 1 -h 1 -sx 20 -sy 1 -sz 0 -ax 0 1 0 -rcp 0
-name BucketA;
```

จากนั้นให้ย้าย BucketA ขึ้นมา 0.51 ตามแนวแกน Y เพื่อให้มันลอยอยู่บน Floor แบบไม่สัมผัสกัน

```
move 0 0.51 0;
```

ขั้นตอนต่อไปเราจะ modify ให้กลายเป็นรูปถัง สามารถทำได้ด้วยการลบ face ด้านบนส่วนที่ cap รูปทรงอยู่ ในการ select เฉพาะ face ของวัตถุใดๆสามารถทำได้ด้วยคำสั่ง select และ flag –r ตามด้วย object name.f[หมายเลขของ face]; ในที่นี้คือ face ที่ 21 จากนั้นสามารถลบได้ด้วยคำสั่งลบธรรมดา doDelete ดังตัวอย่าง

```
select -r BucketA.f[21];
doDelete;
```

จากนั้นสร้างถังที่สองขึ้นมา วิธีที่ง่ายที่สุดคือการ duplicate ขึ้นมาจากถังอันแรกด้วยคำสั่ง duplicate ตามด้วย flag –name เพื่อตั้งชื่อถังใหม่ แล้วตามด้วยชื่อวัตถุต้นแบบ โดยให้ตั้งชื่อถังใหม่ว่า BucketB จากนั้นย้ายมันขึ้นมาวางเหนือ ว่า BucketA ดังตัวอย่าง

```
      duplicate -name BucketB BucketA;
      select -r BucketB;
      move 1 2.5 0;
```

จากนั้นสร้างถัง BucketC ขึ้นมาจากการ duplicate ถัง BucketA เช่นกัน แล้วย้ายให้ขึ้นมาอยู่เหนือ BucketA และ BucketB ดังภาพตัวอย่าง

```
duplicate -name BucketC BucketA;
select -r BucketC;
move 0 4.5 0;
```


951301: Advanced Animation and Rendering Techniques

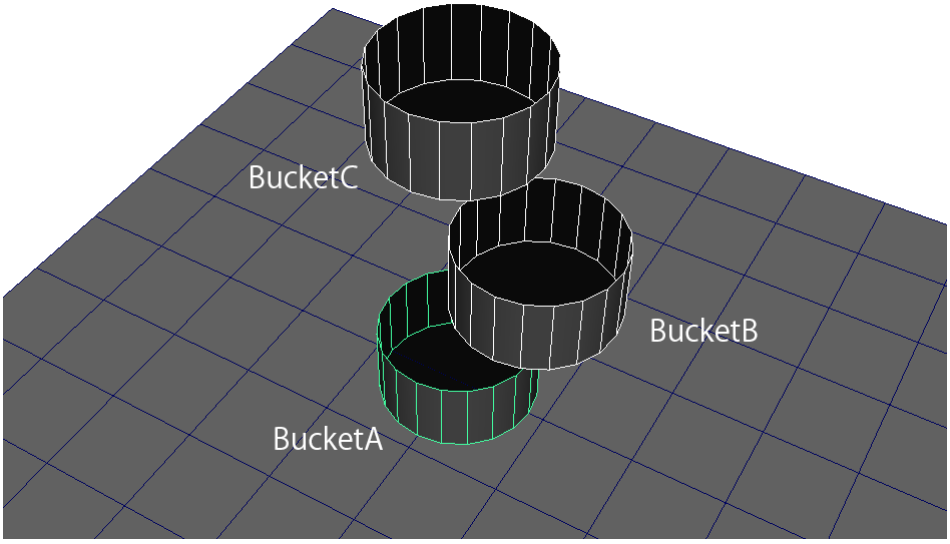

Fig 05-09: แสดงตำแหน่งของ Bucket-A, Bucket-B, และ Bucket-C

เมื่อฉากเราพร้อมแล้ว ขั้นตอนต่อไปเราจะมาสร้าง emitter กัน โดยกำหนดให้จุดกำเนิดอยู่ในตำแหน่ง 0 9 0 เพื่อให้ลอบอยู่ตรงกลางเหนือ BucketC พอดี ให้ตั้งชื่อว่า waterEmit โดยให้มีค่า properties ดังตัวอย่างด้านล่าง เมื่อเสร็จแล้วให้สร้าง particles ขึ้นมา connect กับ emitter ที่สร้าง

```
// create particles
string $eObject[] = `emitter -position 0 9 0 -type direction
-name waterEmit -rate 100 -speed 1 -spread 0.2 -dx 0 -dy -
1.0 -dz 0`;
string $pObject[] = `particle`;
connectDynamic -em $eObject[0] $pObject[0];
```

จัดการเพิ่ม playback time ให้เป็น 1200 เฟรม เพื่อให้สามารถเห็น dynamics ที่จะทำต่อไปได้อย่างครบถ้วน

```
      playbackOptions -min 1 -max 1200;
```

จากนั้นลอง execute คำสั่ง แล้ว play animation ดูจะพบว่า particles ถูกปล่อยออกมา อย่างไม่มีที่สิ้นสุด

951301: Advanced Animation and Rendering Techniques

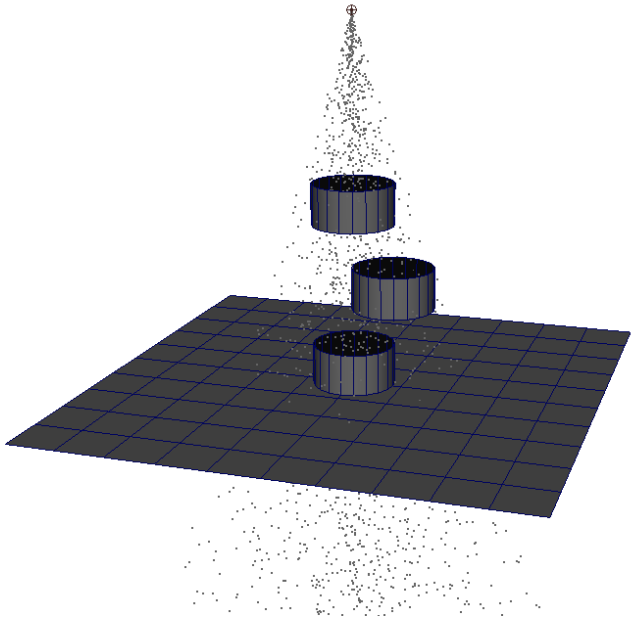

Fig 05-10: แสดง particles ที่ถูกปล่อยออกมาอย่างไม่มีที่สิ้นสุด ผ่านถังทั้งสาม

ในขั้นตอนนี้เราจะเริ่มจากการทำให้ถังใบบนสุด (BucketC) เป็น solid body ก่อน เช่นเดียวกับที่เราทำใน workshop 2 เราสามารถทำได้โดยการสร้าง collision ให้กับ BucketC แล้ว connect BucketC collision เข้ากับ particles ที่สร้างไว้ ในคราวนี้ให้ลองตั้ง particle render type เป็น 3 ดังตัวอย่าง

```
// set collision to BucketC
collision -resilience 0.3 -friction 0.0 -offset 0.0 BucketC;
connectDynamic -collisions BucketC particle1;
setAttr particle1.particleRenderType 3;
addAttr -longName radious -defaultValue 0.1 particleShape1;
```

ในการทำงานบางกรณีเราอาจต้องการกำหนดจำนวนของ particles ให้สอดคล้องกับความต้องการของเรา สามารถทำได้โดยการตั้งค่าให้กับ max count ในที่นี้ให้กำหนดจำนวน particles ไว้ที่ 1500 เพื่อให้ particles หยุดไหลก่อนที่เราจะทำการเคลื่อนย้ายถัง

```
// setting max count of particles to 1500
setAttr particleShape1.maxCount 1500;
```

ทดลอง execute คำสั่งแล้ว play animation ดูจะพบว่า particles จะถูกปล่อยมาจนครบ 1500 ภายในเฟรมที่ 400 โดนประมาณ จากนั้น emitter จะไม่ปล่อย particles ออกมาอีก แต่ปัญหาตอนนี้คือการฟุ้งกระจายของ particles เมื่อตกกระทบ BucketC เนื่องจากยังไม่มี gravity field

951301: Advanced Animation and Rendering Techniques

ขั้นต่อไปเราจะมาสร้าง gravity field แล้ว connect เข้ากับ particles เพื่อสร้างแรงดึงไม่ให้ particles ฟุ้งกระจายแบบไร้ทิศทางต่อไป สามารถทำได้ดังนี้

```
// adding gravity field
gravity -magnitude 8 -attenuation 0.3 -dx 0 -dy -1 -dz 0;
connectDynamic -fields gravityField1 particle1;
```

เมื่อเสร็จแล้วให้ทำงานสร้าง solid body ให้กับวัตถุที่เหลือทั้งหมด ประกอบด้วย BucketA, BucketB และ Floor โดยให้มีค่า collision เช่นเดียวกับ BucketC ทั้งหมด

```
// set collision to BucketA, B, and Floor
collision -resilience 0.3 -friction 0.0 -offset 0.0 BucketB;
collision -resilience 0.3 -friction 0.0 -offset 0.0 BucketA;
connectDynamic -collisions BucketB particle1;
connectDynamic -collisions BucketA particle1;
collision -resilience 0.3 -friction 0.0 Floor;
connectDynamic -collisions Floor particle1;
```

เมื่อเตรียมสถานการณ์ทุกอย่างเรียบร้อยแล้ว ขั้นตอนสุดท้ายคือเราจะมาสร้างความเคลื่อนไหวให้กับ BucketC เมื่อบรรจุ particles จนครบแล้วให้เท particles ทั้งหมดใส่ลงใน BucketB

ในขั้นตอนนี้เราไม่สามารถใช้คำสั่ง translations ได้ทันที เนื่องจากคำสั่งจะถูก execute ในเฟรมปัจจุบัน เฟรมเดียวทั้งหมด ดังนั้นก่อนที่เราจะสั่ง translate ได้ เราต้องบอกกับโปรแกรมก่อนว่าจะให้ทำงานที่เฟรมอะไร ซึ่งสามารถทำได้จากคำสั่ง currentTime ตามด้วยเฟรมที่ต้องการ ในที่นี้เราต้องการให้ BucketC เริ่มหมุนในเฟรมที่ 400 และหยุดหมุนในเฟรมที่ 500 โดยให้หมุนเป็นมุม -91 องศา ดังตัวอย่างด้านล่าง

```
// set keyframes to buckets
// Bucket C throw
currentTime 400 ;
select -r BucketC ;
setKeyframe {"BucketC"};
currentTime 500 ;
rotate -r -os -fo 0 0 -91;
setKeyframe -breakdown 0 -hierarchy none -controlPoints 0 -
shape 0 {"BucketC"};
```

จากนั้นลอง execute คำสั่งดูถ้าทำถูกต้องจะพบว่ามีแถบสีแดงแสดง keyframes อยู่ที่ timeline ด้านล่างสองจุด คือเฟรมที่ 400 และเฟรมที่ 500 และเมื่อ select ที่ BucketC จะพบว่าที่ channel box ค่า translations ต่างๆจะเปลี่ยนเป็นสีชมพู

ขั้นตอนต่อไปเราจะเคลื่อนถังกลับสู่ตำแหน่งเดิมเมื่อเท particles ออกเรียบร้อยแล้ว โดยให้เริ่มต้นเคลื่อนกลับที่เฟรม 600 แล้วเคลื่อนเสร็จที่เฟรม 700 ดังตัวอย่างด้านล่าง

951301: Advanced Animation and Rendering Techniques

```
currentTime 600 ;
select -r BucketC ;
setKeyframe {"BucketC"};
currentTime 700 ;
rotate -r -os -fo 0 0 91;
setKeyframe -breakdown 0 -hierarchy none -controlPoints 0 -
shape 0 {"BucketC"};
```

เมื่อเสร็จแล้วให้ลอง execute คำสั่ง สังเกตการณ์เคลื่อนที่ของ particles และ วัตถุทั้งหมดว่ามีความสอดคล้องกันตามที่เราต้องการหรือไม่ ถ้าถูกต้องแล้วให้ใช้วิธีเดียวกัน apply ให้กับ BucketB ดังตัวอย่างด้านล่าง

```
// Bucket B thrown
currentTime 700 ;
select -r BucketB ;
setKeyframe {"BucketB"};
currentTime 800 ;
rotate -r -os -fo 0 0 91;
setKeyframe -breakdown 0 -hierarchy none -controlPoints 0 -
shape 0 {"BucketB"};

// Bucket B unthrown
currentTime 900 ;
select -r BucketB ;
setKeyframe {"BucketB"};
currentTime 1000 ;
rotate -r -os -fo 0 0 -91;
setKeyframe -breakdown 0 -hierarchy none -controlPoints 0 -
shape 0 {"BucketB"};
```


951301: Advanced Animation and Rendering Techniques

คำสั่งทั้งหมดของ workshop 3 มีดังนี้

```
// create floor
polyPlane -w 12 -h 12 -name Floor;

// create bucket
polyCylinder -r 1 -h 1 -sx 20 -sy 1 -sz 0 -ax 0 1 0 -rcp 0
-name BucketA;
move 0 0.51 0;
select -r BucketA.f[21] ;
doDelete;

//create second and third buckets
duplicate -name BucketB BucketA;
select -r BucketB;
move 1 2.5 0;

duplicate -name BucketC BucketA;
select -r BucketC;
move 0 4.5 0;

// create particles
string $eObject[] = `emitter -position 0 9 0 -type direction
-name waterEmit -rate 100 -speed 1 -spread 0.2 -dx 0 -dy -
1.0 -dz 0`;
string $pObject[] = `particle`;
connectDynamic -em $eObject[0] $pObject[0];
playbackOptions -min 1 -max 1200;

// set collision to BucketC
collision -resilience 0.3 -friction 0.0 -offset 0.0 BucketC;
connectDynamic -collisions BucketC particle1;
setAttr particle1.particleRenderType 3;
addAttr -longName radious -defaultValue 0.1 particleShape1;

// setting max count of particles to 1500
setAttr particleShape1.maxCount 1500;

// adding gravity field
gravity -magnitude 8 -attenuation 0.3 -dx 0 -dy -1 -dz 0;
connectDynamic -fields gravityField1 particle1;

// set collision to BucketA, B, and Floor
collision -resilience 0.3 -friction 0.0 -offset 0.0 BucketB;
collision -resilience 0.3 -friction 0.0 -offset 0.0 BucketA;
connectDynamic -collisions BucketB particle1;
connectDynamic -collisions BucketA particle1;
collision -resilience 0.3 -friction 0.0 Floor;
connectDynamic -collisions Floor particle1;

// set keyframes to buckets
```


951301: Advanced Animation and Rendering Techniques

```
// Bucket C thrown
currentTime 400 ;
select -r BucketC ;
setKeyframe {"BucketC"};
currentTime 500 ;
rotate -r -os -fo 0 0 -91;
setKeyframe -breakdown 0 -hierarchy none -controlPoints 0 -
shape 0 {"BucketC"};

// Bucket C unthrown
currentTime 600 ;
select -r BucketC ;
setKeyframe {"BucketC"};
currentTime 700 ;
rotate -r -os -fo 0 0 91;
setKeyframe -breakdown 0 -hierarchy none -controlPoints 0 -
shape 0 {"BucketC"};
// Bucket B thrown
currentTime 700 ;
select -r BucketB ;
setKeyframe {"BucketB"};
currentTime 800 ;
rotate -r -os -fo 0 0 91;
setKeyframe -breakdown 0 -hierarchy none -controlPoints 0 -
shape 0 {"BucketB"};

// Bucket B unthrown
currentTime 900 ;
select -r BucketB ;
setKeyframe {"BucketB"};
currentTime 1000 ;
rotate -r -os -fo 0 0 -91;
setKeyframe -breakdown 0 -hierarchy none -controlPoints 0 -
shape 0 {"BucketB"};
```

Reference

- Wilkins, M. R. and Kazmier, C. (2005) *MEL Scripting for Maya Animations*, Morgan Kaufmann Publishers, Elsevier Inc.
- Galanakis, R. (2014) *Practical Maya Programming with Python*, Packt Publishing Ltd.
- Stripinis, D. (2003) *The MEL Companion: Maya Scripting 3D Artists*, Charles River Media, INC

951301: Advanced Animation and Rendering Techniques