01418341 สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์
การบรรยายครั้งที่ 2

ประมุข ขันเงิน
pramook@gmail.com

CULLING

Culling

- รูปหลายเหลี่ยมมีสองหน้า: ด้านหน้า กับ ด้านหลัง
- เวลาวาดรูปทรงสามมิติ ส่วนมากด้านหน้าจะหันหน้ามาหาเรา
- ฉะนั้นโดยมากไม่มีความจำเป็นต้องวาดรูปเหลี่ยมที่หันหลังใส่เรา
  - ช่วยให้โปรแกรมเร็วขึ้น
  - ช่วยทำเอฟเฟกต์บางอย่าง เช่นมองเห็นภาพภายนอกจากภายในวัตถุ
- เราสามารถสั่งให้ OpenGL ไม่วาดรูปหลายเหลี่ยมที่หันหน้าที่เราไม่ต้องการเห็นมาให้ได้
- การเลือกไม่วาดรูปหลายเหลี่ยม เราเรียกว่า Culling

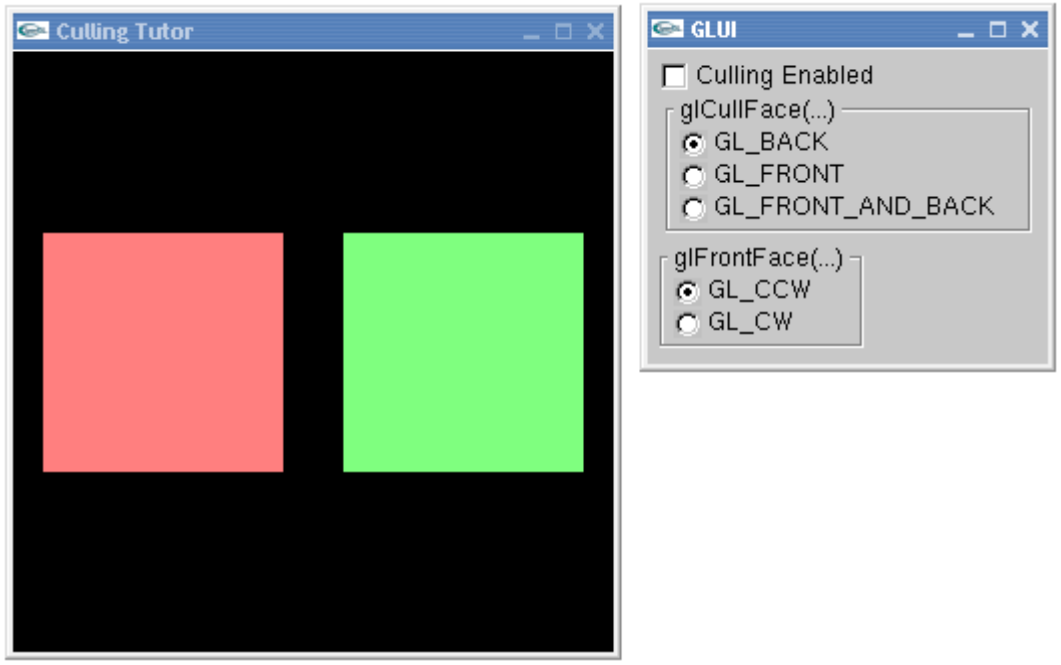
glCullFace

- glCullFace(GLenum mode)
  - กำหนดรูปแบบการ cull รูปหลายเหลี่ยม
  - mode มีค่าได้สามแบบ
    - GL_FRONT = ไม่วาดด้านหน้า
    - GL_BACK = ไม่วาดด้านหลัง
    - GL_FRONT_AND_BACK = ไม่วาดทั้งสองด้าน
  - ห้ามเรียกใน glBegin(…) และ glEnd()
  - ก่อนการทำ culling จะมีผล จะต้องสั่ง glEnable(GL_CULL_FACE) เสียก่อน
  - ยกเลิกการทำ culling (วาดหมดทุกด้าน) โดยการสั่ง glDisable(GL_CULL_FACE)

แล้วอะไรคือด้านหน้า อะไรคือด้านหลัง?

- โดยปกติ รูปหลายเหลี่ยมที่เรามองแล้วเห็นจุดมุมเรียงกันตามแนวทวนเข็มนาฬิกา ถือว่าหัน “ด้านหน้า” ให้เรา
- ถ้าเห็นจุดมุมเรียงกันตามเข็มนาฬิกา ถือว่าหัน “ด้านหลัง” ให้
- แต่เราสามารถเปลี่ยนได้ ด้วย glFrontFace
  - glFrontFace(GL_CCW) = ถือว่า “ทวนเข็ม” เป็นด้านหน้า
  - glFrontFace(GL_CW) = ถือว่า “ตามเข็ม” เป็นด้านหน้า

ดู demo


Culling Tutor
GLUI
Culling Enabled
glCullFace(...)
GL_BACK
GL_FRONT
GL_FRONT_AND_BACK
glFrontFace(...)
GL_CCW
GL_CW

ANIMATION WITH GLUT

สร้างภาพเคลื่อนไหวด้วย GLUT

- เราสามารถสร้างภาพเคลื่อนไหวโดยการวาดภาพใหม่เป็นระยะๆ
- กล่าวคือ ทุกช่วงเวลาจำกัด เราจะต้องมีการเรียกฟังก์ชันที่เราให้ไปใน glutDisplayFunc ใหม่
  - ถ้าต้องการความเร็วประมาณ 30 เฟรม/วินาที (ลื่นไหลพอใช้ได้) ต้องมีการเรียกฟังก์ชันนี้ทุกๆ 33 มิลลิวินาที
  - ถ้าต้องการความเร็วประมาณ 60 เฟรม/วินาที (ลื่นไหลมากๆ) ต้องมีการเรียกฟังก์ชันนี้ทุกๆ 17 มิลลิวินาที

แล้วจะเรียกฟังก์ชัน display อย่างไร?

- เรียกตรงๆ ความจริงก็ได้อยู่ แต่มันจะไปซ้ำซ้อนกับการเรียกใน glutMainLoop
- เราสามารถสั่งให้ glutMainLoop เรียกฟังก์ชัน display ได้ด้วยคำสั่ง glutPostRedisplay()
- glutPostRedisplay()
  - บอก GLUT ว่าเราต้องการให้มันเรียกฟังก์ชันที่ให้ไปกับ glutDisplayFunc ในลูปครั้งต่อไป
  - ถ้าเรียก glutPostRedisplay หลายครั้งก่อนลูปต่อไป ฟังก์ชัน display จะถูกเรียกเพียงครั้งเดียว

แล้วจะเรียก glutPostRedisplay ทุกๆ X มิลลิวินาทีได้อย่างไร?

- glutTimerFunc(unsigned int msecs,
  void (*func)(int value), value)
  - เรียกฟังก์ชัน func ที่ให้มาเป็น argument ที่สองหลังจากเวลาผ่านไปได้ msecs มิลลิวินาที
  - func จะได้รับ value ที่ให้มาเป็น argument ที่ 3 เป็น argument
  - จะเรียก func เพียงครั้งเดียวเท่านั้น
  - ดังนั้นถ้าต้องการให้เรียกเป็นคาบๆ ใน func จะต้องมีการเรียก glutTimerFunc ให้ตัวเองใหม่

ตัวอย่าง 1

```
void printInt(int value)
{
   printf("Got an integer: %d\n", value);
}

int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
   glutCreateWindow("glutTimerFunc Example 01");
   glutDisplayFunc(display);
   glutTimerFunc(2000, printInt, 123);
   glutTimerFunc(3000, printInt, 456);
   glutTimerFunc(4000, printInt, 789);
   glutMainLoop();
}
```

ตัวอย่าง 2

```
void printAndReschedule(int interval)
{
   printf("I'm printing this every %d milliseconds\n", interval);
   glutTimerFunc(interval, printAndReschedule, interval);
}

int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
   glutCreateWindow("glutTimerFunc Example 01");
   glutDisplayFunc(display);
   glutTimerFunc(0, printAndReschedule, 500);
   glutTimerFunc(0, printAndReschedule, 1000);
   glutTimerFunc(0, printAndReschedule, 2000);
   glutMainLoop();
}
```

ANIMATION &
MVC DESIGN PATTERN

หลักการทำ animation แบบ Model-View-Controller

- เราแบ่งโปรแกรมของเราออกเป็นสามส่วน
  - Model = ส่วนที่จัดการกับข้อมูล ในที่นี้คือโมเดลสามมิติหรือสองมิติที่เราต้องการแสดง
  - View = ส่วนที่นำโมเดลไปแสดง
  - Controller = ส่วนที่ทำการเปลี่ยนแปลงโมเดล ซึ่งอาจเป็นผลมาจากเวลาที่ผ่านไป หรือการสั่งงานของผู้ใช้
- มีการแบ่งหน้าที่กันชัดเจน
  - Controller ทำหน้าที่เปลี่ยนแปลงโมเดลอย่างเดียว ไม่มีหน้าที่แสดง
  - View ทำหน้าที่แสดงโมเดลอย่างเดียว ไม่มีเปลี่ยนแปลงโมเดลเอง

หลักการทำ animation แบบ Model-View-Controller (ต่อ)

- ในโปรแกรมกราฟฟิกส์ที่ใช้ GLUT ของเรา
  - Model = ส่วนที่เก็บข้อมูลต่างๆ
  - View = callback ของ glutDisplayFunc
  - Controller = callback function อื่นๆ
    - callback ของ glutTimerFunc
    - callback สำหรับเวลาผู้ใช้ใช้ keyboard หรือ mouse

Animation ลูกบอลเด้งกระทบกำแพง

- Model
  - ข้อมูลของลูกบอล
  - ตำแหน่งของจุดศูนย์กลางลูกบอล
  - ความเร็วของลูกบอล
  - รัศมี
  - สี
- ข้อมูลลูกบอลควรประกาศไว้เป็น global variable เพื่อที่จะได้ใช้ร่วมกันได้หลายฟังก์ชัน

```
struct Ball
{
  double x, y;
  double vx, vy;
  double radius;
  double r, g, b;
};

Ball ball;
```

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

- View
  - วาดลูกบอลเป็นวงกลมทึบด้วย OpenGL

```
void drawBall(Ball &b)
{
   glColor3d(b.r, b.g, b.b);
   glBegin(GL_TRIANGLE_FAN);
   glVertex2d(b.x, b.y);
   for(int i=0;i<CORNERS;i++)
   {
        double theta = 2*M_PI*i/CORNERS;
        double x = b.x + b.radius * cos(theta);
        double y = b.y + b.radius * sin(theta);
        glVertex2d(x,y);
   }
   glVertex2d(b.x + b.radius, b.y);
   glEnd();
}
```

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

- View (ต่อ)
  - สั่งให้วาดลูกบอลใหม่ทุกครั้งที่มีการเรียกฟังก์ชัน display

```
void display()
{
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   drawBall(ball);
   glFlush();
}

int main(int argc, char **argv)
{
   ...
   glutDisplayFunc(display);
   ...
}
```

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

- Controller
  - ทำให้ลูกบอลเปลี่ยนตำแหน่ง
  - เช็คว่าลูกบอลชนกำแพงหรือไม่
  - ถ้าชน ให้เปลี่ยนทิศทางการเคลื่อนที่ของมันด้วยการเปลี่ยนความเร็ว
- ฟังก์ชันที่ทำหน้าที่เป็น controller จะถูกเรียกเป็นช่วงๆ
  - ใช้ glutTimerFunc
  - ต้องลงทะเบียนตัวเองกับ glutTimerFunc ใหม่ทุกครั้งที่ถูกเรียก
  - ต้องเรียก glutPostRedisplay เพื่อให้ความเปลี่ยนแปลงถูกแสดงผล

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
void updateBall(Ball &b)
{
   b.x += b.vx;
   b.y += b.vy;
   if (b.x + b.radius > 1)
   {
        b.x = 1 - b.radius;
        b.vx = -fabs(b.vx);
   }
   if (b.x - b.radius < -1)
        /* handle collision with left wall */
   if (b.y + b.radius > 1)
        /* handle collision with top wall */
   if (b.y - b.radius < -1)
        /* handle collision with bottom wall */
}
```

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
void updateBall(Ball &b)
{
   b.x += b.vx;
   b.y += b.vy;
   if (b.x + b.radius > 1)
   {
        b.x = 1 - b.radius;
        b.vx = -fabs(b.vx);
   }
   if (b.x - b.radius < -1)
        /* handle collision with left wall */
   if (b.y + b.radius > 1)
        /* handle collision with top wall */
   if (b.y - b.radius < -1)
        /* handle collision with bottom wall */
}
```

เคลื่อนที่ลูกบอล

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
void updateBall(Ball &b)
{
   b.x += b.vx;
   b.y += b.vy;
   if (b.x + b.radius > 1)
   {
        b.x = 1 - b.radius;
        b.vx = -fabs(b.vx);
   }
   if (b.x - b.radius < -1)
        /* handle collision with left wall */
   if (b.y + b.radius > 1)
        /* handle collision with top wall */
   if (b.y - b.radius < -1)
        /* handle collision with bottom wall */
}
```

เช็คว่าชนกับกำแพงด้านขวาหรือไม่

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
void updateBall(Ball &b)
{
   b.x += b.vx;
   b.y += b.vy;
   if (b.x + b.radius > 1)
   {
        b.x = 1 - b.radius;
        b.vx = -fabs(b.vx);
   }
   if (b.x - b.radius < -1)
        /* handle collision with left wall */
   if (b.y + b.radius > 1)
        /* handle collision with top wall */
   if (b.y - b.radius < -1)
        /* handle collision with bottom wall */
}
```

ถ้าชนให้เปลี่ยนทิศทางให้วิ่งไปด้านซ้าย

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
void updateBall(Ball &b)
{
   b.x += b.vx;
   b.y += b.vy;
   if (b.x + b.radius > 1)
   {
        b.x = 1 - b.radius;
        b.vx = -fabs(b.vx);
   }
   if (b.x - b.radius < -1)
        /* handle collision with left wall */
   if (b.y + b.radius > 1)
        /* handle collision with top wall */
   if (b.y - b.radius < -1)
        /* handle collision with bottom wall */
}
```

โค้ดของด้านอื่นๆ
ก็คล้ายๆ กัน

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
void animate(int param)
{
   updateBall(ball);
   glutTimerFunc(INTERVAL, animate, 0);
   glutPostRedisplay();
}

int main(int argc, char **argv)
{
   ...
   glutTimerFunc(0, animate, 0);
   ...
}
```

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
void animate(int param)
{
   updateBall(ball);
   glutTimerFunc(INTERVAL, animate, 0);
   glutPostRedisplay();
}

int main(int argc, char **argv)
{
   ...
   glutTimerFunc(0, animate, 0);
   ...
}
```

เปลี่ยนโมเดลของลูกบอล (ทำให้มันเคลื่อนที่)

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
void animate(int param)
{
   updateBall(ball);
   glutTimerFunc(INTERVAL, animate, 0);
   glutPostRedisplay();
}

int main(int argc, char **argv)
{
   ...
   glutTimerFunc(0, animate, 0);
   ...
}
```

สั่งให้เรียกฟังก์ชันนี้ใหม่อีกครั้ง
หลังจากเวลาผ่านไป
INTERVAL มิลลิวินาที

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
void animate(int param)
{
   updateBall(ball);
   glutTimerFunc(INTERVAL, animate, 0);
   glutPostRedisplay();
}

int main(int argc, char **argv)
{
   ...
   glutTimerFunc(0, animate, 0);
   ...
}
```

สั่งให้วาดทั้งฉากใหม่

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
void animate(int param)
{
   updateBall(ball);
   glutTimerFunc(INTERVAL, animate, 0);
   glutPostRedisplay();
}

int main(int argc, char **argv)
{
   ...
   glutTimerFunc(0, animate, 0);
   ...
}
```

ลงทะเบียนให้เรียก animate
เป็นครั้งแรกหลัง glutMainLoop
เริ่มทำงาน (แล้วมันจะจัดการให้มัน
ถูกเรียกครั้งต่อๆ ไปเอง)

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
    glutInitWindowSize(600,600);
    glutCreateWindow("Bouncing Ball");
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutTimerFunc(0, animate, 0);

    ball.x = 0;
    ball.y = 0;
    ball.vx = 0.02;
    ball.vy = 0.03;
    ball.radius = 0.05;
    ball.r = 1.0;
    ball.g = 0.5;
    ball.b = 0.5;

    glutMainLoop();
}
```

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
    glutInitWindowSize(600,600);
    glutCreateWindow("Bouncing Ball");
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutTimerFunc(0, animate, 0);

    ball.x = 0;
    ball.y = 0;
    ball.vx = 0.02;
    ball.vy = 0.03;
    ball.radius = 0.05;
    ball.r = 1.0;
    ball.g = 0.5;
    ball.b = 0.5;

    glutMainLoop();
}
```

เริ่มต้น GLUT

สร้างวินโดว์

ลงทะเบียนฟังก์ชัน

ฯลฯ

Animation ลูกบอลเด้งกระทบกำแพง (ต่อ)

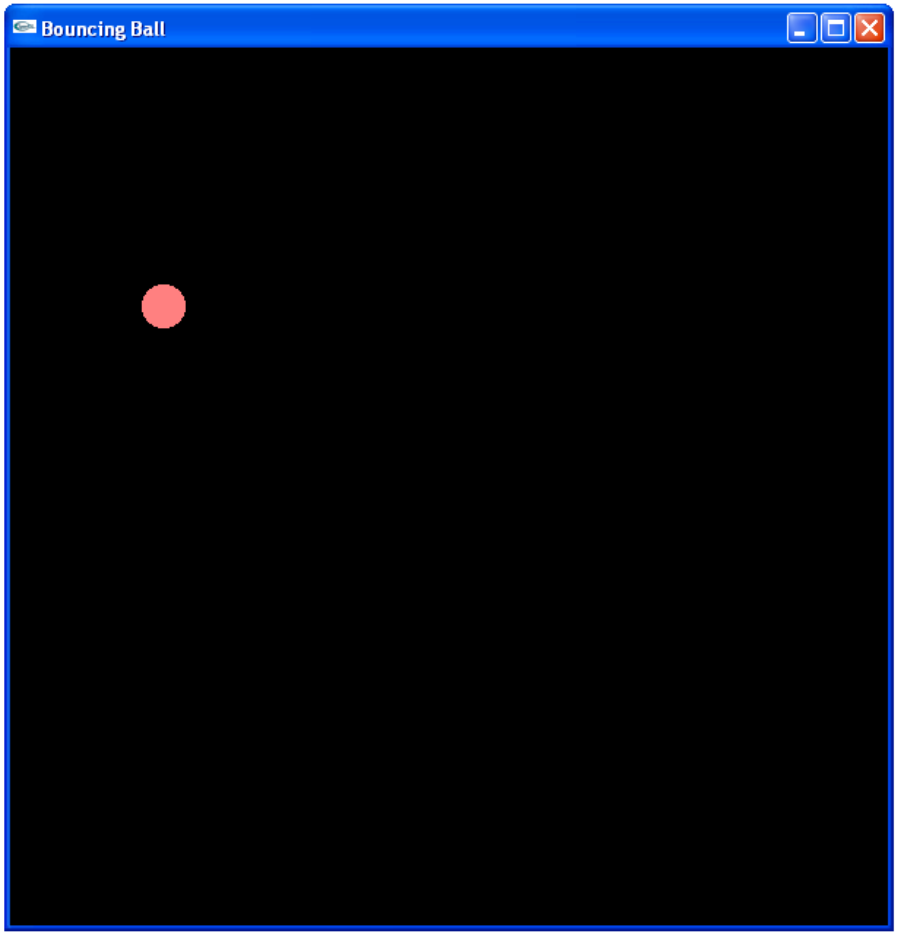
```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
    glutInitWindowSize(600,600);
    glutCreateWindow("Bouncing Ball");
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutTimerFunc(0, animate, 0);

    ball.x = 0;
    ball.y = 0;
    ball.vx = 0.02;
    ball.vy = 0.03;
    ball.radius = 0.05;
    ball.r = 1.0;
    ball.g = 0.5;
    ball.b = 0.5;

    glutMainLoop();
}
```

กำหนดค่าเริ่มต้นของลูกบอล

ฎ demo



DOUBLE BUFFERING

Flickering

- ทำไมภาพมันกระพริบๆ (ภาษาฝรั่งเรียกว่า flickering)
- จอภาพจะแสดงภาพใหม่เป็นช่วงๆ (ส่วนมากความถี่อยู่ที่ 60-72 เฮิร์ตซ์)
- เวลาจอภาพ มันจะไปเอาข้อมูลมาจาก framebuffer
- การวาดภาพของเรามีหลายขั้นตอน
  - ล้าง framebuffer ให้เป็นสีดำ
  - วาดลูกบอลทีละกลีบ
- เพราะฉะนั้นข้อมูลใน framebuffer บางเวลาจึงไม่ใช่ภาพที่สมบูรณ์
- เมื่อเวลาประจวบเหมาะ จอภาพจะเอาภาพที่ไม่สมบูรณ์ไปแสดง ทำให้ภาพกระพริบ

Double Buffering

- แก้ไขได้โดยการให้มี framebuffer สองอัน
- อันหนึ่งอยู่หลัง อันหนึ่งอยู่หน้า
- เวลาวาดให้วาดใส่ framebuffer ที่อยู่ข้างหลัง
- เวลาแสดงให้เอา framebuffer ที่อยู่ข้างหลังไปแสดง
- ภาพที่แสดงจึงเป็นภาพที่สมบูรณ์เสมอ
- ไม่เกิด flickering

การใช้ double buffering ใน GLUT

- ใช้ GLUT_DOUBLE แทน GLUT_SINGLE ใน glutInitDisplayMode
- ใช้ glutSwapBuffers() แทน glFlush()

Animation ลูกบอลเด้งกระทบกำแพง (แก้ใหม่)

```
void display()
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT);

  drawBall(ball);

  glFlush();
}
```

Animation ลูกบอลเด้งกระทบกำแพง (แก้ใหม่)

```
void display()
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT);

  drawBall(ball);

  glutSwapBuffers();
}
```

Animation ลูกบอลเด้งกระทบกำแพง (แก้ใหม่)

```
int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
   glutInitWindowSize(600,600);
   ...
}
```

Animation ลูกบอลเด้งกระทบกำแพง (แก้ใหม่)

```
int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
   glutInitWindowSize(600,600);
   ...
}
```

KEYBOARD INPUT

รับอินพุตจากแป้นพิมพ์

- glutKeyboardFunc(void (*func)(unsigned char key, int x, int y))
  - ให้ฟังก์ชันที่รับ unsigned char และ int สองตัว
  - key คือ ascii code ของปุ่มที่กด
  - x และ y คือตำแหน่งของ mouse

ตัวอย่าง 1

```
void printKey(unsigned char key, int x, int y)
{
    printf("You type key %d. The mouse is at (%d, %d)\n", key, x, y);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
    glutCreateWindow("glutKeyboardFunc Example");
    glutDisplayFunc(display);
    glutKeyboardFunc(printKey);
    glutMainLoop();
}
```

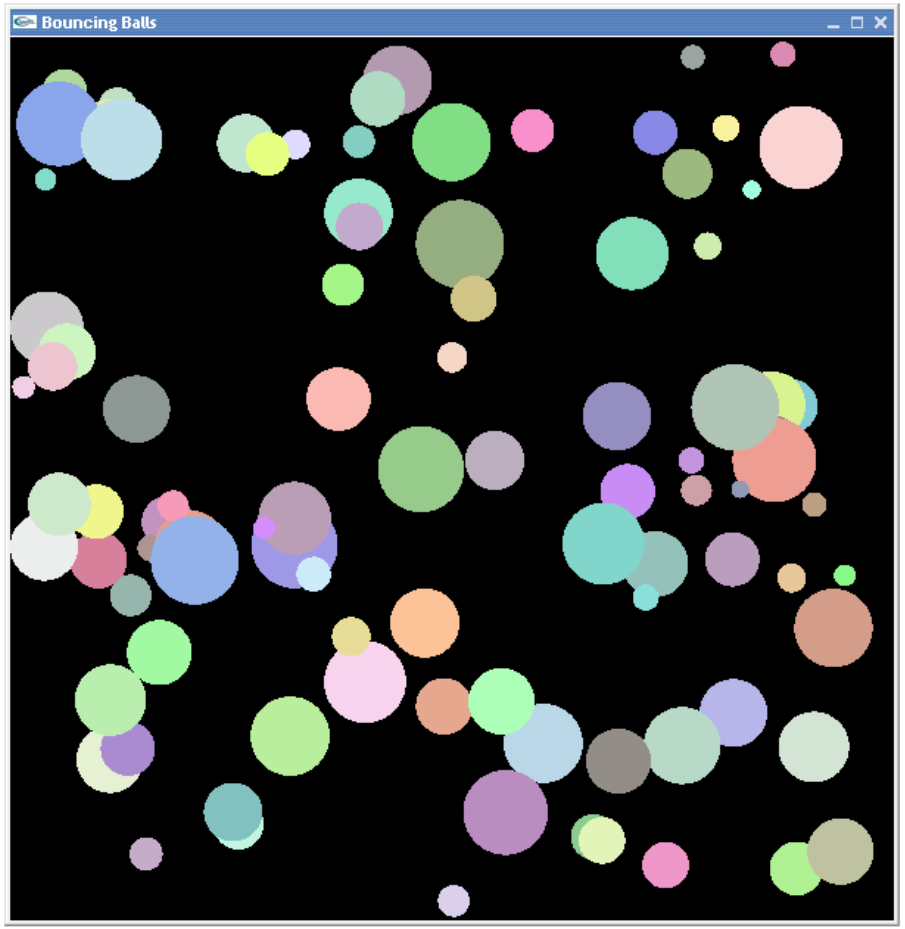
ตัวอย่าง 2

- โปรแกรมลูกบอลเด้งชนกำแพง เวอร์ชัน 2
- ตอนนี้มีลูกบอลได้หลายลูกแล้ว
- กด space bar แล้วมีลูกบอลเพิ่ม
- กด Esc เป็นการออกจากโปรแกรม

ตัวอย่าง 2

```
void keyboard(unsigned char key, int x, int y)
{
   if (key == ' ')
   {
       newRandomBall();
       glutPostRedisplay();
   }
   else if (key == 27)
       exit(0);
}
```

ดู demo



3D LINEAR ALGEBRA

เวกเตอร์สามมิติ

- ลำดับของจำนวนจริงสามตัว

$$(x, y, z)$$

- สัญลักษณ์: ตัวอักษรตัวพิมพ์เล็กหนา

$$\mathbf{u}, \mathbf{v}, \mathbf{w}$$

- เซตของเวกเตอร์สามมิติ

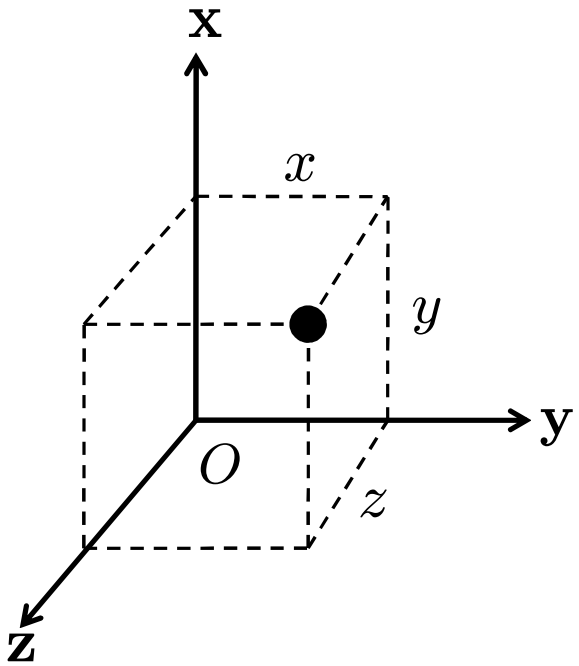
$$\mathbb{R}^3 = \{(x, y, z) : x, y, z \in \mathbb{R}\}$$

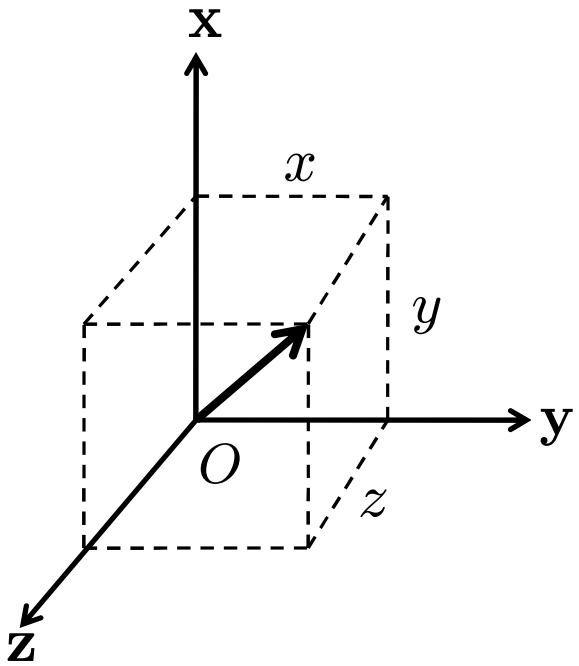
- เวกเตอร์พิเศษ

$$\mathbf{x} = (1, 0, 0),\ \mathbf{y} = (0, 1, 0),\ \mathbf{z} = (0, 0, 1),\ \mathbf{0} = (0, 0, 0)$$

เวกเตอร์สามมิติ (ต่อ)

- ความหมาย
  - จุดในสามมิติ
  - ทิศทางในสามมิติ





ปฏิบัติการบนเวกเตอร์สามมิติ

- กำหนดให้

$$\begin{aligned}\mathbf{u} &= (x_1, y_1, z_1)\\ \mathbf{v} &= (x_2, y_2, z_2)\end{aligned}$$

- การบวกและลบเวกเตอร์

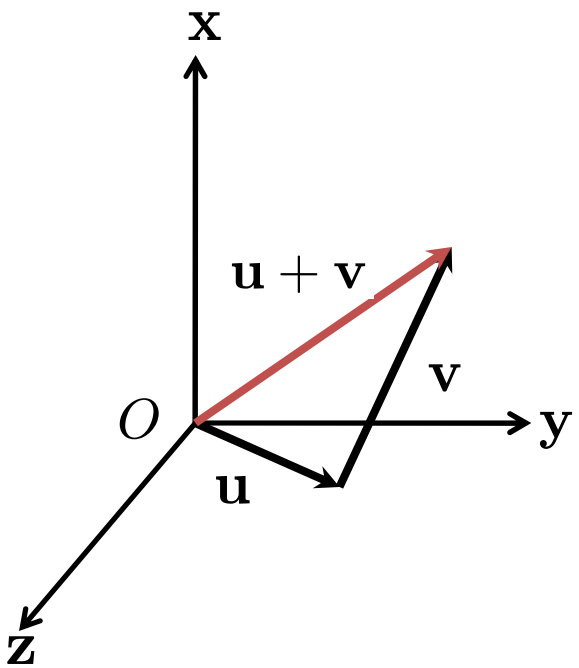
$$\mathbf{u} + \mathbf{v} = (x_1 + x_2,\ y_1 + y_2,\ z_1 + z_2)$$

- การคูณเวกเตอร์ด้วยสเกลาร์

$$c\mathbf{u} = (cx_1, cx_2, cx_3)$$

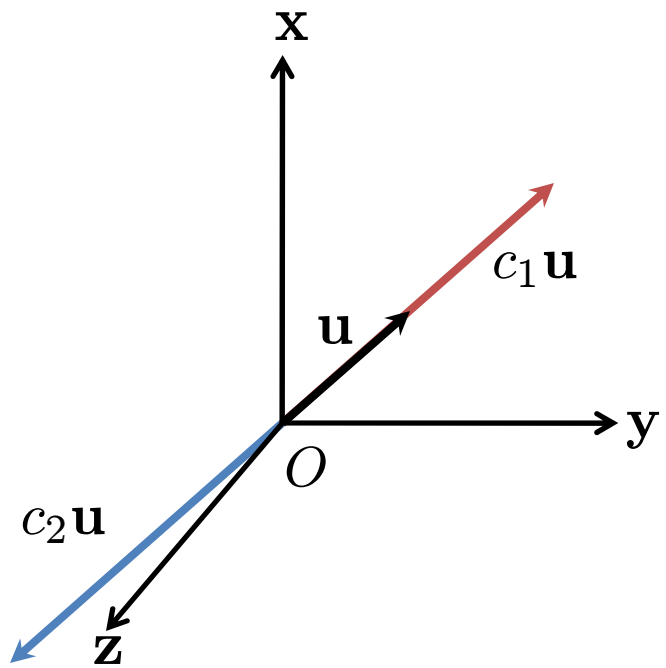
ปฏิบัติการบนเวกเตอร์สามมิติ (ต่อ)

- การบวก = เอาท้ายต่อหัว



ปฏิบัติการบนเวกเตอร์สามมิติ (ต่อ)

- การคูณด้วยสเกลาร์ = การยืด/หด



Linear Combination

- ให้ $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ เป็นเวกเตอร์ใดๆ

  และ $c_1, c_2, \ldots, c_n$ คือจำนวนจริงใดๆ

  เราเรียก

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \ldots + c_n\mathbf{v}_n$$

  ว่า linear combination ของ $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$

Linear Combination (ต่อ)

- เวกเตอร์สามมิติทุกเวกเตอร์เป็น linear combination ของ $\mathbf{x}$, $\mathbf{y}$, และ $\mathbf{z}$

$$(x, y, z) = x(1, 0, 0) + y(0, 1, 0) + z(0, 0, 1) = x\mathbf{x} + y\mathbf{y} + z\mathbf{z}$$

Span

- ถ้า $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ เป็นเวกเตอร์ใดๆ แล้ว

  เราเรียกเซต

  $$\{c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n : c_1, c_2, \ldots, c_n \in \mathbb{R}\}$$

  ว่า span ของ $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$

Span (ต่อ)

- Span ของ $\mathbf{x}$, $\mathbf{y}$, และ $\mathbf{z}$ มีค่าเท่ากับ $\mathbb{R}^3$
- Span ของ $\mathbf{x}$ มีค่าเท่ากับแกน x
- Span ของ $\mathbf{y}$ มีค่าเท่ากับแกน y
- Span ของ $\mathbf{z}$ มีค่าเท่ากับแกน z
- Span ของ $\mathbf{x}$ และ $\mathbf{y}$ มีค่าเท่ากับระนาบ xy
- Span ของ $\mathbf{x}$ และ $\mathbf{z}$ มีค่าเท่ากับระนาบ xz
- Span ของ $\mathbf{y}$ และ $\mathbf{z}$ มีค่าเท่ากับระนาบ yz

Linear Dependence

- เรากล่าวว่า $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$
  เป็นกลุ่มของเวกเตอร์ที่ linearly dependent
  ถ้ามีสเกลาร์ $c_1, c_2, \ldots, c_n$ ที่มีตัวใดตัวหนึ่งมีค่าไม่เท่ากับ 0
  ที่ทำให้

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \ldots + c_n\mathbf{v}_n = \mathbf{0}$$

Linear Independence

- ตรงข้ามกับ linear dependence
- เรากล่าวว่า $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$

  เป็นกลุ่มของเวกเตอร์ที่ linearly independent

  ถ้าค่า $c_1, c_2, \ldots, c_n$ ที่ทำให้

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \ldots + c_n\mathbf{v}_n = \mathbf{0}$$

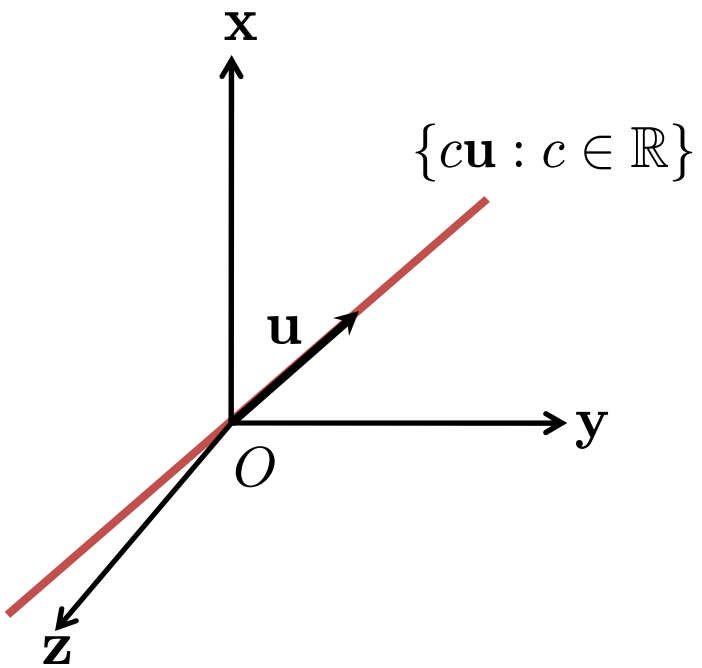
  คือ $c_1 = c_2 = \cdots = c_n = 0$ เท่านั้น

Linear Independence (ต่อ)

- **x**, **y**, และ **z** **---** linearly independent
- (1,2,0) และ (2,4,0) --- linearly dependent
  เพราะ 2*(1,2,0) – (2,4,0) = (0,0,0)
- (1,0,1), (2,3,0), (2,1.5,1) --- linearly dependent
  เพราะ (1,0,1) + 0.5*(2,3,0) – (2,1.5,1) = (0,0,0)
- (0,0,0) --- linearly dependent
  เพราะ c(0,0,0) = (0,0,0) สำหรับค่า c ใดๆ ที่ไม่เท่ากับ 0

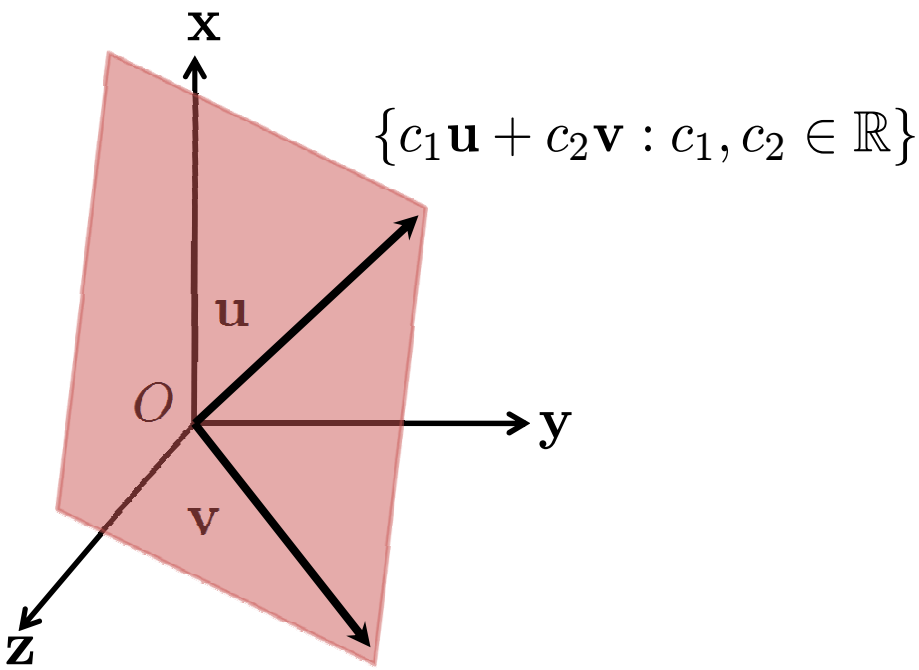
เส้นตรง

- ถ้า $\mathbf{u} \neq \mathbf{0}$ แล้ว span ของ $\mathbf{u}$ คือเส้นตรงที่ผ่านจุด O และ $\mathbf{u}$



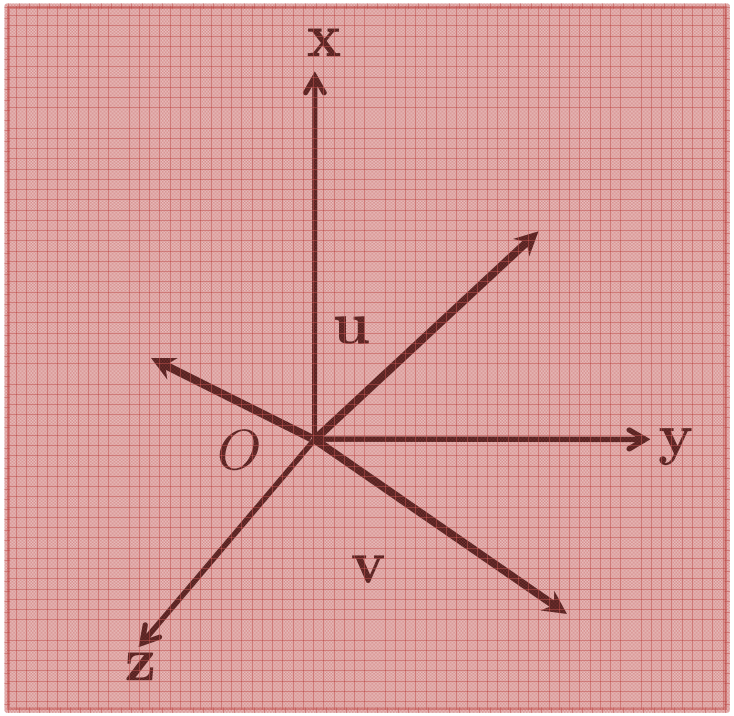
ระนาบ

- ถ้า $\mathbf{u}$ และ $\mathbf{v}$ เป็นเวกเตอร์ที่ linearly independent กันแล้ว span ของ $\mathbf{u}$ และ $\mathbf{v}$ คือระนาบที่ผ่านจุด O, $\mathbf{u}$, และ $\mathbf{v}$



ปริภูมิเวกเตอร์

- ถ้า **u, v,** และ **w** เป็นเวกเตอร์ที่ linearly independent กัน แล้ว span ของ **u, v,** และ **w** เซตของเวกเตอร์สามมิติทั้งหมด



$$\{c_1\mathbf{u} + c_2\mathbf{v} + c_3\mathbf{w} : c_1, c_2, c_3 \in \mathbb{R}\}$$

Basis

- ถ้า span ของ **u, v,** และ **w**

  มีค่าเท่ากับเซตของเวกเตอร์สามมิติทั้งหมด

  เราเรียก **u, v,** และ **w** ว่าเป็น basis ของปริภูมิเวกเตอร์สามมิติ

Basis (ต่อ)

- **x**, **y**, และ **z** เป็น basis ของปริภูมิเวกเตอร์สามมิติ
- (1,1,0), (0,1,1), และ (1,0,1) ก็เป็น basis
- แต่ (1,0,1), (2,3,0), (2,1.5,1) ไม่ใช่ เพราะมันไม่ linearly independent

ผลคูณสเกลาร์

- ผลคูณสเกลาร์ (dot product)

$$\mathbf{u} \cdot \mathbf{v} = x_1 x_2 + y_1 y_2 + z_1 z_2$$

- ขนาดเวกเตอร์

$$\|\mathbf{u}\| = \sqrt{x_1^2 + y_1^2 + z_1^2} = (\mathbf{u} \cdot \mathbf{u})^{\frac{1}{2}}$$

- สมบัติต่างๆ

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$$

$$\mathbf{u} \cdot (\mathbf{v} + \mathbf{w}) = \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{w}$$

$$\mathbf{u} \cdot (c\mathbf{v}) = c(\mathbf{u} \cdot \mathbf{v})$$

$$\|\mathbf{u} + \mathbf{v}\|^2 = \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2 + 2(\mathbf{u} \cdot \mathbf{v})$$

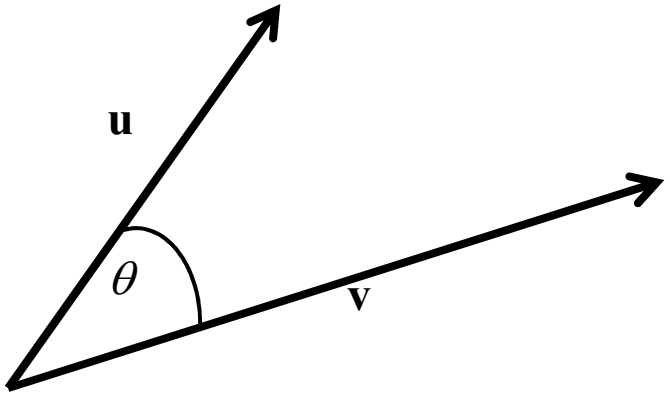
$$\|c\mathbf{u}\| = c\|\mathbf{u}\|$$

ผลคูณสเกลาร์ (ต่อ)

- สมบัติ

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos\theta$$

เมื่อ θ คือมุมระหว่าง $\mathbf{u}$ กับ $\mathbf{v}$



- $\mathbf{u}$ กับ $\mathbf{v}$ ตั้งฉากกันก็ต่อเมื่อ $\mathbf{u} \cdot \mathbf{v} = 0$

เวกเตอร์หนึ่งหน่วย

- เวกเตอร์ที่มีขนาดเท่ากับ 1
- $\|\mathbf{u}\| = 1$
- ถ้า $\mathbf{u}$ เป็นเวกเตอร์หนึ่งหน่วยแล้ว

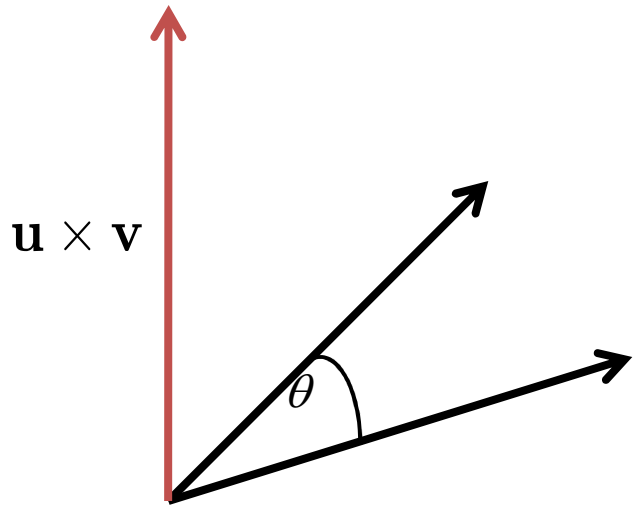
  $\mathbf{u} \cdot \mathbf{v} = \|\mathbf{v}\| \cos\theta$ คือความยาวของ $\mathbf{v}$ เมื่อแตกแรงไปในทิศของ $\mathbf{u}$

ผลคูณเวกเตอร์

- ผลคูณเวกเตอร์ (cross product)

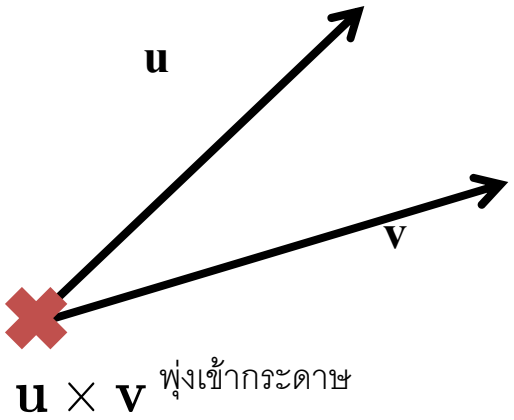
$$\begin{aligned}\mathbf{u}\times\mathbf{v} &= (y_1z_2 - y_2z_1,\ z_1x_2 - z_2x_1,\ x_1y_2 - x_2y_1)\\ &= (y_1z_2 - y_2z_1)\mathbf{x} + (z_1x_2 - z_2x_1)\mathbf{y} + (x_1y_2 - x_2y_1)\mathbf{z}\\ &= \begin{vmatrix}\mathbf{x} & \mathbf{y} & \mathbf{z}\\ x_1 & y_1 & z_1\\ x_2 & y_2 & z_2\end{vmatrix}\end{aligned}$$

- $\mathbf{u}\times\mathbf{v}$ ตั้งฉากกับทั้ง $\mathbf{u}$ และ $\mathbf{v}$
- $\|\mathbf{u}\times\mathbf{v}\| = \|\mathbf{u}\|\|\mathbf{v}\|\sin\theta$

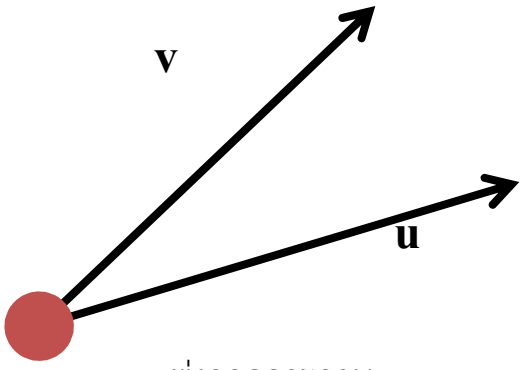


ผลคูณเวกเตอร์ (ต่อ)

- ทิศทางของ $\mathbf{u} \times \mathbf{v}$ คิดตาม**กฎมือขวา**
  - เอามือขวาชี้ไปตามทิศของ $\mathbf{u}$ ให้ฝ่ามือหันไปทาง $\mathbf{v}$
  - $\mathbf{u} \times \mathbf{v}$ ตั้งฉากกับระนาบที่นิยามโดย $\mathbf{u}$ กับ $\mathbf{v}$ พุ่งออกไปฝั่งที่นิ้วโป้งขวาอยู่



$\mathbf{u} \times \mathbf{v}$ พุ่งเข้ากระดาษ



$\mathbf{u} \times \mathbf{v}$ พุ่งออกกระดาษ

ผลคูณเวกเตอร์ (ต่อ)

- สมบัติต่างๆ

$$\mathbf{u} \times \mathbf{v} = -\mathbf{v} \times \mathbf{u}$$

$$\mathbf{u} \times (\mathbf{v} + \mathbf{w}) = \mathbf{u} \times \mathbf{v} + \mathbf{u} \times \mathbf{w}$$

$$\mathbf{u} \times (r\mathbf{v}) = (r\mathbf{u}) \times \mathbf{v} = r(\mathbf{u} \times \mathbf{v})$$

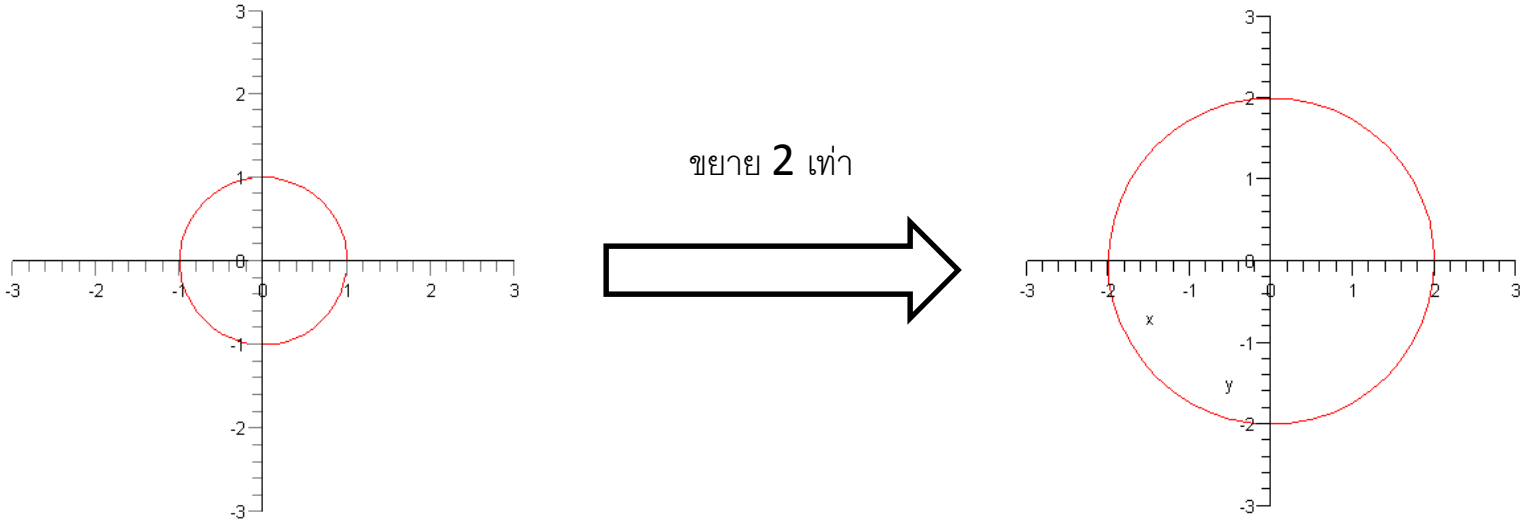
การแปลง

การแปลง (Transformations)

- ตัวอย่าง
  - “เลื่อนไปทางซ้าย 1 หน่วย”
  - “หมุนรอบแกน y 90 องศา”
  - “ขยายขนาดตามแกน z 2 เท่า”
- เอาไปใช้ที่ไหน?
  - Modeling
  - Animation
  - Rendering Pipeline

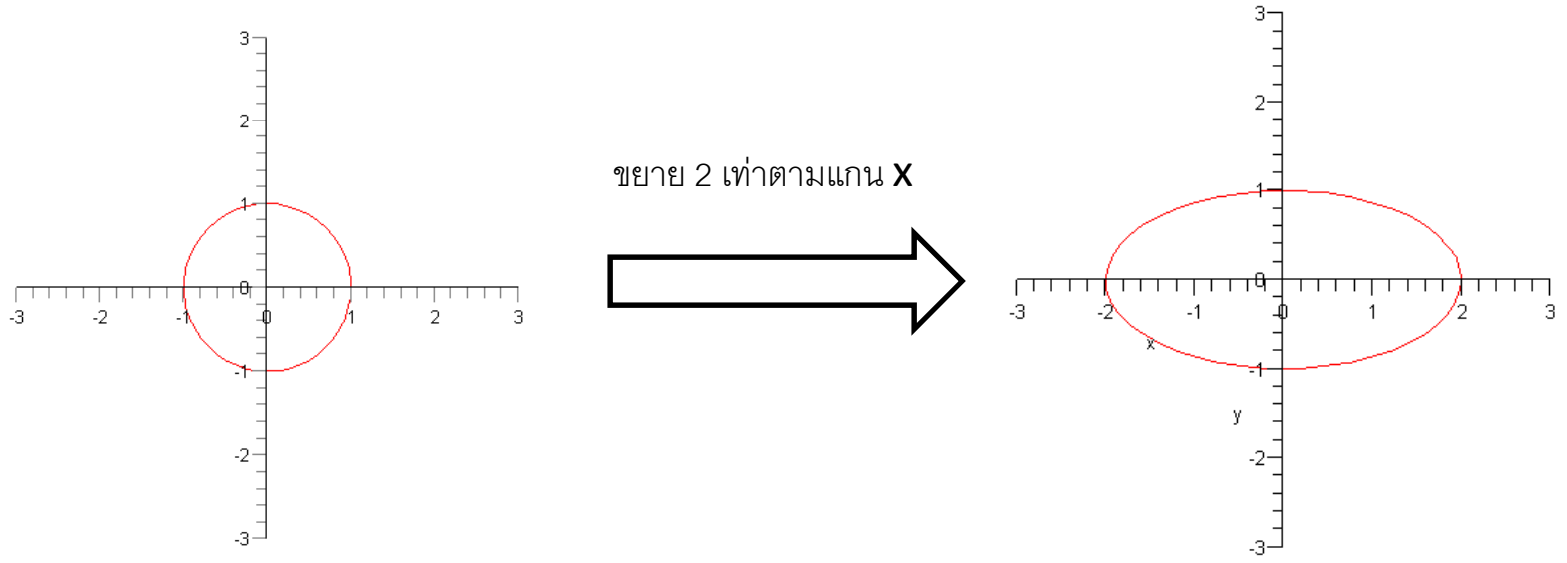
ตัวอย่าง

- สมมติเรารู้วิธีสร้างวงกลมรัศมี 1 หน่วย จุดศูนย์กลางอยู่ที่ (0,0)
- อยากได้วงกลมรัศมี 2 หน่วย จุดศูนย์กลางอยู่ที่เดิม



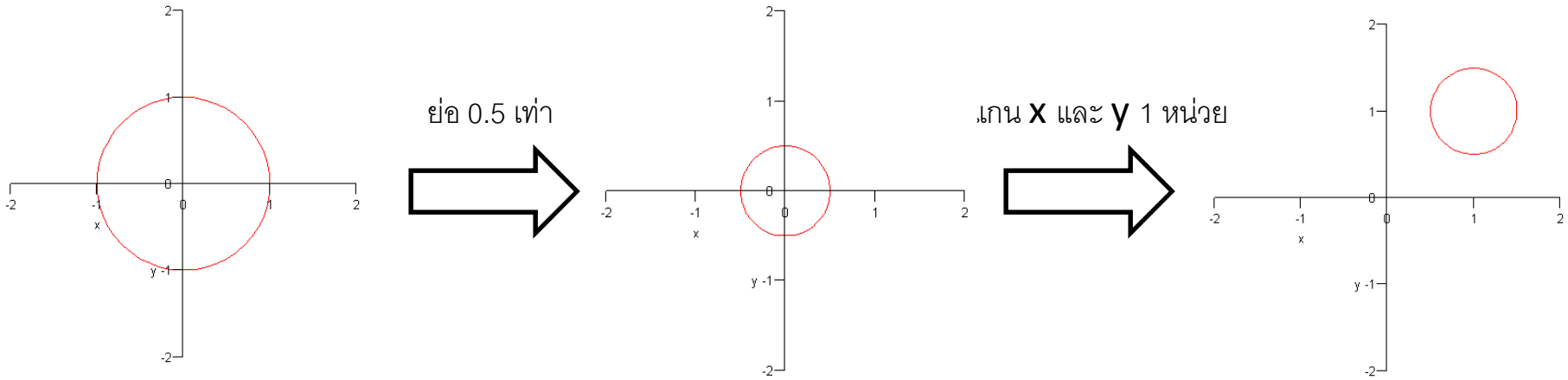
ตัวอย่าง

- อยากได้วงรีแกนเอกยาว 2 หน่วย แกนโทยาว 1 หน่วย



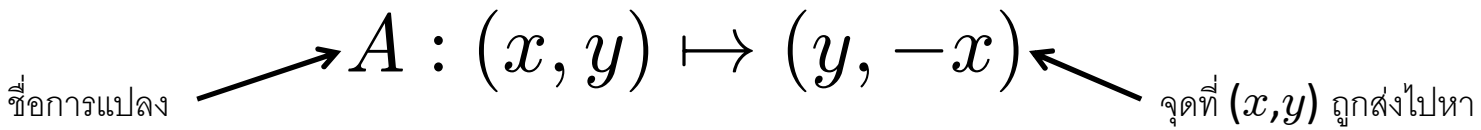
ตัวอย่าง

- อยากได้วงกลมรัศมี 0.5 หน่วย จุดศูนย์กลางอยู่ที่จุด (1,1)



การแปลงในสองมิติ

- **การแปลงในสองมิติ** คือ ฟังก์ชันที่ส่งเวกเตอร์ (หรือจุด) สองมิติไปยังเวกเตอร์สองมิติ
  - **สัญลักษณ์:** ตัวอักษรภาษาอังกฤษตัวใหญ่ A, B, C, D, ...
  - **เวลานิยาม:**



หรือจะเขียนแบบฟังก์ชันก็ได้

$$A((x, y)) = (y, -x)$$

ตัวอย่าง

$$B : (x, y) \mapsto (x + 2, y + 3)$$
$$C : (x, y) \mapsto (x + y, 0)$$
$$D : (x, y) \mapsto (1.5x, 3y)$$
$$E : (x, y) \mapsto (x, e^x)$$
$$F : (x, y) \mapsto C(D((x, y)) = (1.5x + 3y, 0)$$

การแปลงเอกลักษณ์

- **การแปลงเอกลักษณ์** (Identity Transformation) คือ การแปลงที่ส่งจุดทุกจุดไปหาตัวมันเอง

$$I : (x, y) \mapsto (x, y)$$

การแปลงเชิงเส้น

- เรากล่าวว่าการแปลง A เป็น**การแปลงเชิงเส้น (linear transformation)** ถ้ามันสอดคล้องกับสมบัติต่อไปนี้

1. $A(\mathbf{u}+\mathbf{v}) = A(\mathbf{u}) + A(\mathbf{v})$
2. $A(c\mathbf{u}) = cA(\mathbf{u})$

สำหรับเวกเตอร์ $\mathbf{u},\ \mathbf{v}$ ใดๆ ใน $\mathbb{R}^2$ และค่าคงที่ c ใดๆ

ตัวอย่าง

- การแปลงเอกลักษณ์ I เป็นการแปลงเชิงเส้น เพราะ

  1. $I(\mathbf{u}+\mathbf{v}) = \mathbf{u}+\mathbf{v} = I(\mathbf{u}) + I(\mathbf{v})$
  2. $I(c\mathbf{u}) = c\mathbf{u} = cI(\mathbf{u})$

- การแปลง $A : (x, y) \mapsto (y, -x)$ ก็เป็นการแปลงเชิงเส้น
- แต่การแปลง $B : (x, y) \mapsto (x + 2, y + 3)$ ไม่ใช่การแปลงเชิงเส้น เพราะ

$$B((2, 2)) = (4, 5) \neq (6, 8) = B((1, 1)) + B((1, 1))$$

ข้อสังเกต

- ถ้า A เป็น linear transformation แล้ว

$$A(\mathbf{0}) = \mathbf{0}$$

เพราะ $2A(\mathbf{0}) = A(2 \times \mathbf{0}) = A(\mathbf{0})$

การแปลงเชิงเส้นที่สำคัญ 2 ชนิด

- การย่อขยาย (Scaling)
- การหมุน (Rotation)

การย่อขยาย

- **การย่อขยาย (Scaling)** คือการแปลงที่อยู่ในรูป

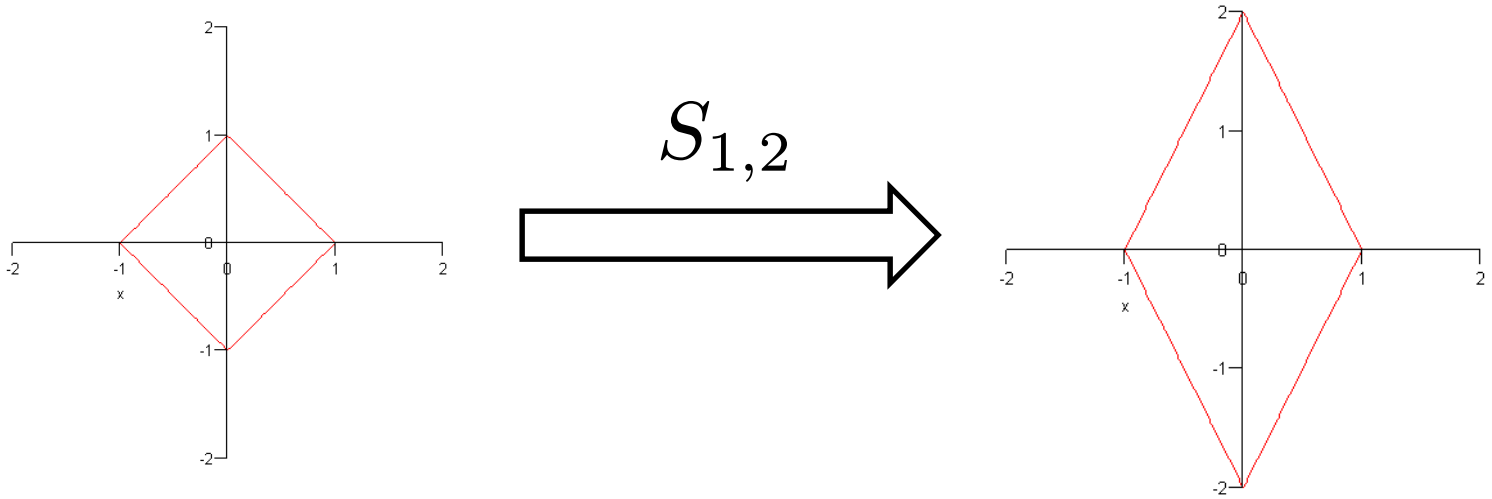
$$S_{\alpha,\beta} : (x, y) \mapsto (\alpha x, \beta y)$$

มีความหมายคือ

- ขยายในแนวแกน x เป็นจำนวน α เท่า
- ขยายในแนวแกน y เป็นจำนวน β เท่า

ตัวอย่าง

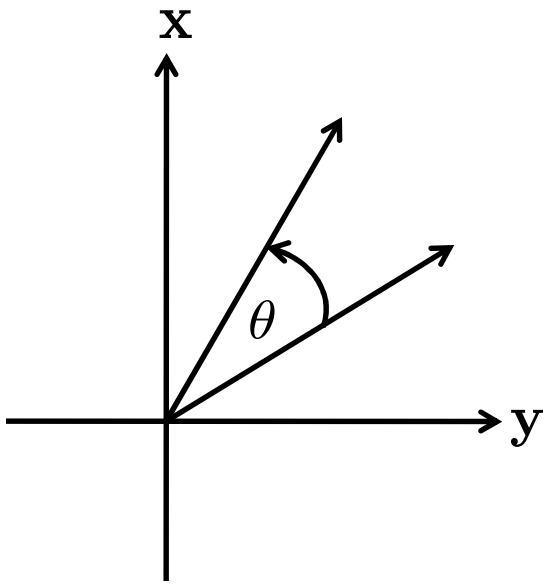
- เนื่องจาก $S_{1,1}((x,y)) = (1x, 1y) = (x,y)$
  ดังนั้น $S_{1,1} = I$
- $S_{2,2}((1,3)) = (2,6)$
- $S_{1,0.5}((3,10)) = (3,5)$
-



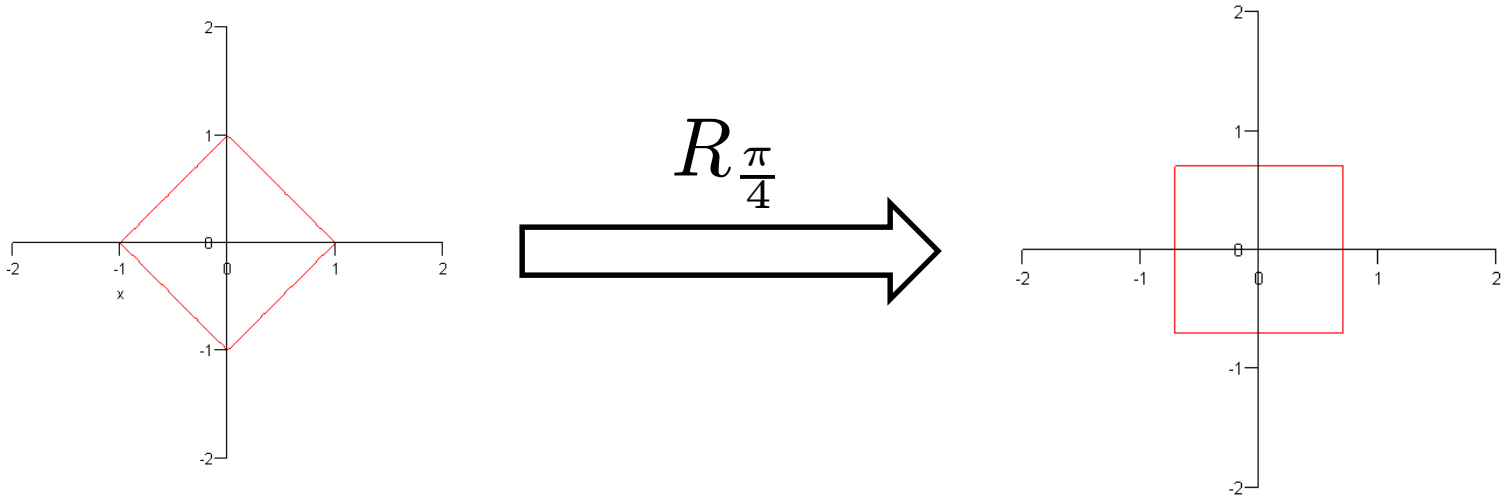
การหมุน

- การหมุน (rotation) ในที่นี้จะต้องกำหนดมุม θ และเราจะหมุนทวนเข็มนาฬิการอบจุด origin ไปเป็นมุม θ
- สัญลักษณ์ R_θ โดย

$$R_\theta : (x, y) \mapsto (x\cos\theta - y\sin\theta, y\cos\theta + x\sin\theta)$$



ตัวอย่าง

- $I = R_0$ (หมุน 0 เรเดียนเท่ากับไม่หมุนเลย)
- $R_{\frac{\pi}{6}}((1,0)) = \left(\cos\frac{\pi}{6}, \sin\frac{\pi}{6}\right) = \left(\frac{\sqrt{3}}{2}, \frac{1}{2}\right)$
- 

Linear Transformation และ Basis

- ให้ $\mathbf{x} = (1,0)$ และให้ $\mathbf{y} = (0,1)$

  เราได้ว่าสำหรับจุด (x,y) ใดๆ

$$(x,y) = x(1,0) + y(0,1) = x\mathbf{x} + y\mathbf{y}$$

- ถ้า A เป็น linear transformation เราจะได้ว่า

$$\begin{aligned} A((x,y)) &= A(x\mathbf{x} + y\mathbf{y}) \\ &= A(x\mathbf{x}) + A(y\mathbf{y}) \\ &= xA(\mathbf{x}) + yA(\mathbf{y}) \end{aligned}$$

Linear Transformation และ Basis

- กล่าวคือถ้าเรารู้ $A(\mathbf{x})$ และ $A(\mathbf{y})$ เราก็สามารถคำนวณ $A((x,y))$ สำหรับเวกเตอร์ (x,y) ใดๆ ได้ทั้งหมด
- พูดอีกแบบคือ linear transformation จะถูกนิยามด้วยค่าของมันที่ basis ของ vector space

การแทนการแปลงเชิงเส้นด้วยเมตริกซ์

- สมมติว่า $A(\mathbf{x}) = (a,b)$ และ $A(\mathbf{y}) = (c,d)$ จะได้ว่า

$$\begin{aligned} A((x,y)) &= x(a\mathbf{x} + b\mathbf{y}) + y(c\mathbf{x} + d\mathbf{y}) \\ &= (ax + cy)\mathbf{x} + (bx + dy)\mathbf{y} \\ &= (ax + cy, bx + dy) \end{aligned}$$

การแทนการแปลงเชิงเส้นด้วยเมตริกซ์

- ถ้าเขียนคู่ลำดับ (x,y) ด้วย column vector $\begin{bmatrix} x \\ y \end{bmatrix}$ จะได้ว่า

$$A\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} ax + cy \\ bx + dy \end{bmatrix}$$

$$= \begin{bmatrix} a & c \\ b & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- ฉะนั้นการแปลงเชิงเส้นสองมิติคือเมตริกซ์ 2x2

การแทนการแปลงเชิงเส้นด้วยเมตริกซ์

- สังเกต

$$A = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

$$\downarrow$$

$$A(\mathbf{x})$$

การแทนการแปลงเชิงเส้นด้วยเมตริกซ์

- สังเกต

$$A = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

$$\downarrow$$

$$A(\mathbf{y})$$

ตัวอย่าง

- I เป็นการแปลงเชิงเส้น และ $I(\mathbf{x}) = (1,0),\ I(\mathbf{y}) = (0,1)$
  ดังนั้น

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

เมตริกซ์เอกลักษณ์

- เนื่องจาก $S_{\alpha,\beta}(\mathbf{x}) = (\alpha, 0), S_{\alpha,\beta}(\mathbf{y}) = (0, \beta)$
  ดังนั้น

$$S_{\alpha,\beta} = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}$$

ตัวอย่าง

- เนื่องจาก

$$R_\theta(\mathbf{x}) = (\cos\theta, \sin\theta)$$
$$R_\theta(\mathbf{y}) = (-\sin\theta, \cos\theta)$$

ดังนั้น

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

การเลื่อนแกนขนาน

- **การเลื่อนแกนขนาน (translation)** คือ การแปลงที่อยู่ในรูป

$$T_{\mathbf{u}} : \mathbf{v} \mapsto \mathbf{v} + \mathbf{u}$$

มีความหมายคือ ถ้า $\mathbf{u} = (u_1, u_2)$ แล้ว
เราจะเลื่อนรูปไปตามแกน x เท่ากับ u_1 หน่วย
และเลื่อนรูปไปตามแกน y เท่ากับ u_2 หน่วย

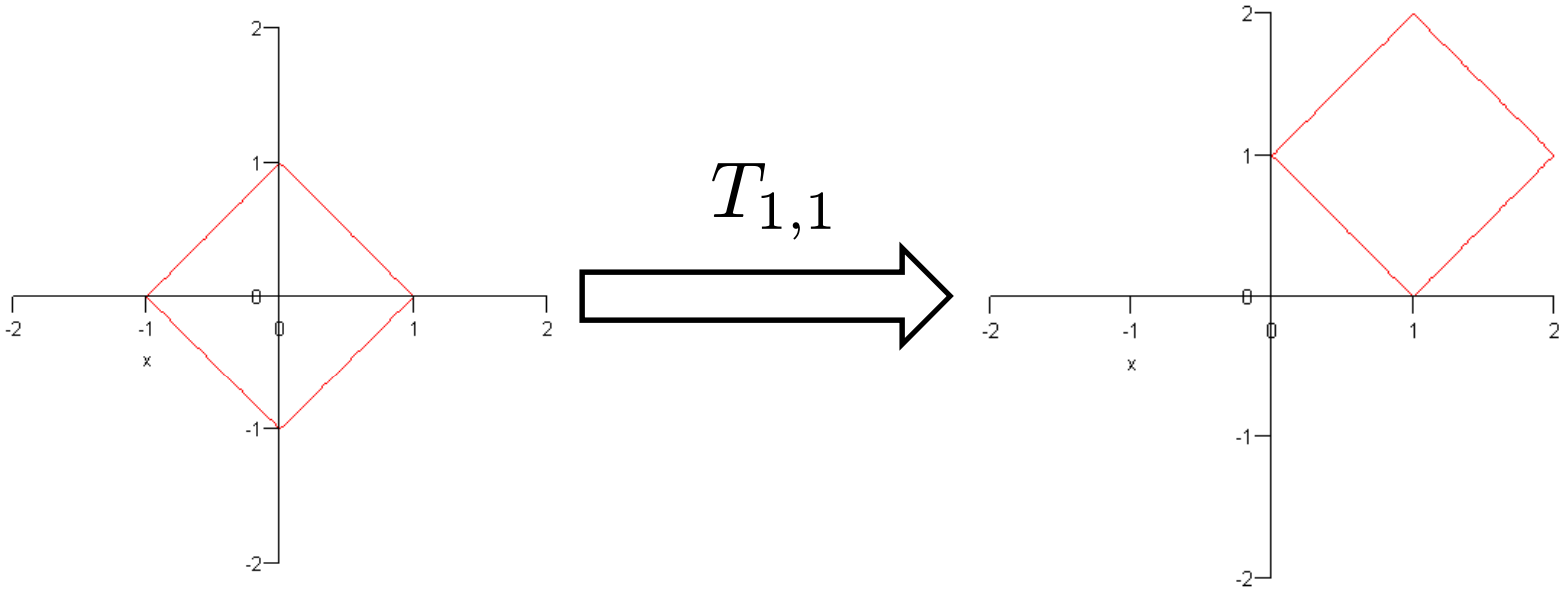
ตัวอย่าง

- I เป็นการเลื่อนแกนขนาน เพราะ $I(\mathbf{u}) = \mathbf{u} + \mathbf{0}$ ดังนั้น $I = T_{(0,0)}$ (เขียนง่ายๆ ว่า $T_{0,0}$)

- $T_{2,3}\ (0,0) = (2,3)$ ดังนั้น $T_{2,3}$ ไม่ใช่การแปลงเชิงเส้น

- กล่าวคือ ถ้า $\mathbf{u}$ ไม่ใช่ $\mathbf{0}$ แล้ว $T_{\mathbf{u}}$ ไม่เป็นการแปลงเชิงเส้น เนื่องจาก $T_{\mathbf{u}}(0) = \mathbf{u}$ ซึ่งมีค่าไม่เท่ากับ $\mathbf{0}$

ตัวอย่าง

- 

Composition

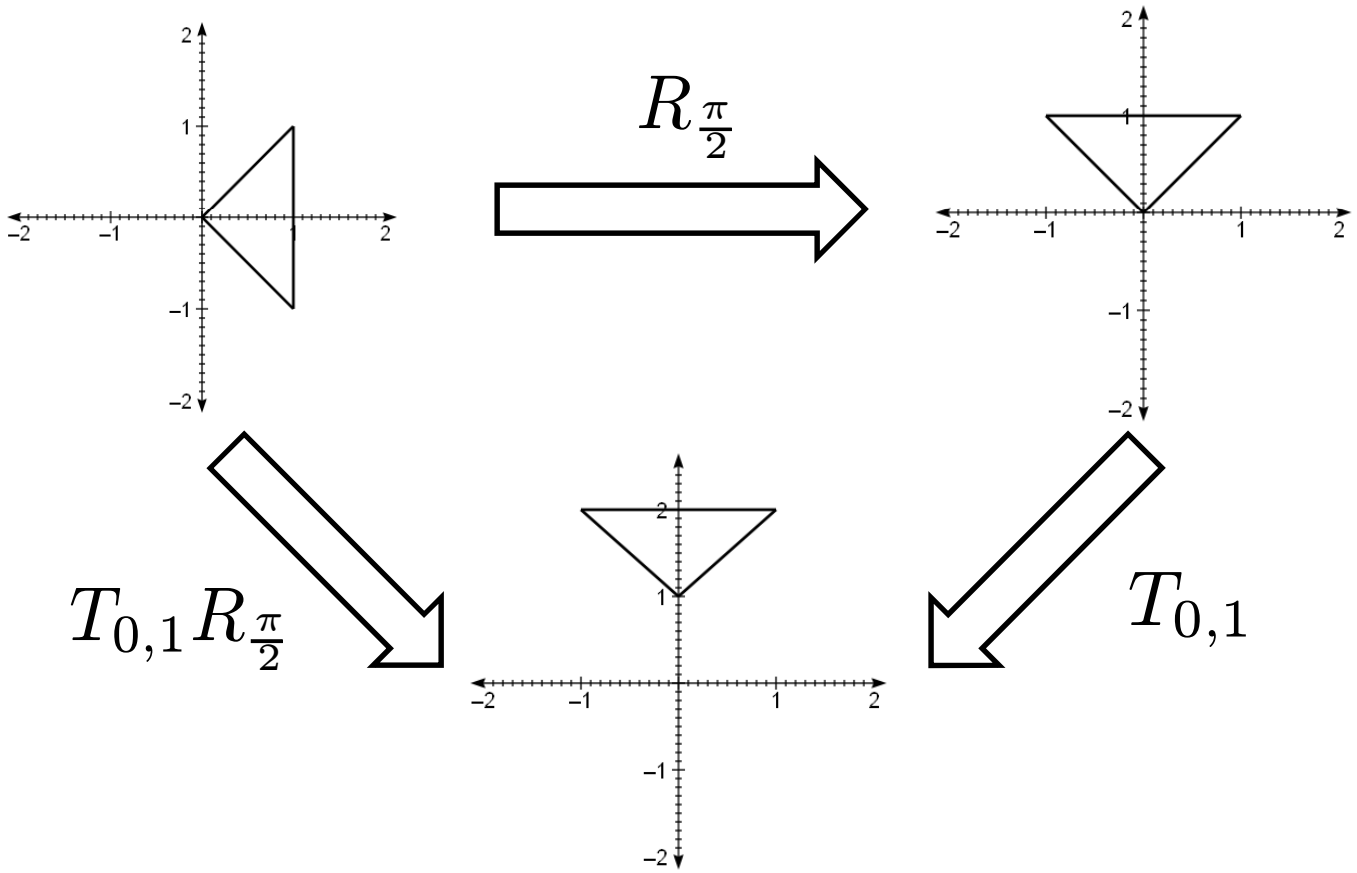
- **Composition** คือการนำเอาการแปลงสองอันมารวมให้เป็นอันเดียววิธีหนึ่ง
- ให้ A กับ B เป็นการแปลง composition ของมันคือการแปลง BA โดยที่

$$BA : (x, y) \mapsto B(A((x, y))$$

กล่าวคือเป็นการแปลงที่เกิดขึ้นจากการนำเวกเตอร์ข้อมูลเข้าไปแปลงด้วย A ก่อนแล้วจึงแปลงด้วย B

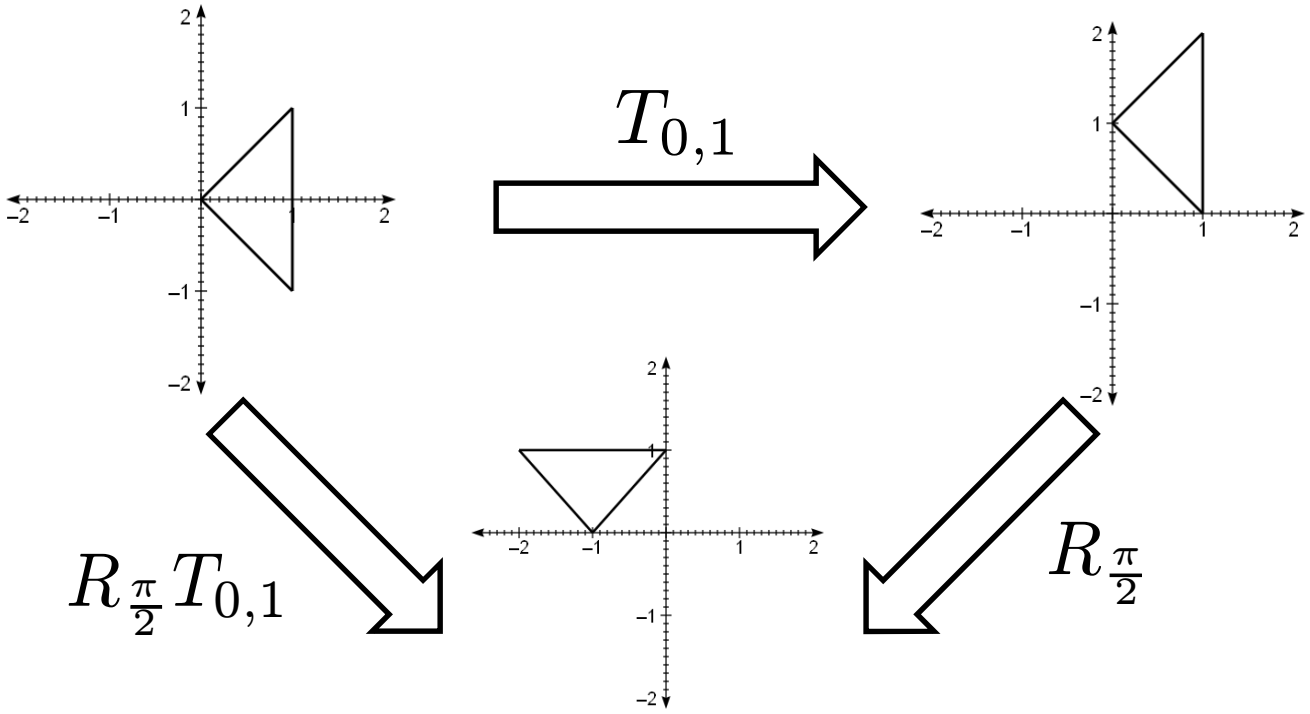
ตัวอย่าง

- $T_{0,1}R_{\frac{\pi}{2}}$ คือการหมุน 90 องศาแล้วเลื่อนทางแกน y หนึ่งหน่วย



ข้อสังเกต

- ถ้า A เป็นการแปลงใดๆ แล้ว $IA = AI = A$
- ระวัง! โดยทั่วไปแล้ว $AB \neq BA$
- ดู $R_{\frac{\pi}{2}} T_{0,1}$



การแปลงแอฟไฟน์

- เรากล่าวว่าการแปลง A เป็น**การแปลงแอฟไฟน์ (affine transformation)** ถ้า

$$A = T_{\mathbf{u}} B$$

โดยที่ $T_{\mathbf{u}}$ เป็นการเลื่อนแกนขนานและ B เป็นการแปลงเชิงเส้น

ตัวอย่าง

- I เป็นการแปลงแอฟไฟน์เพราะ $I = T_{0,0}I$
- $T_{0,1}R_{\frac{\pi}{2}}$ เป็นการแปลงแอฟไฟน์ (อย่างเห็นได้ชัด)
- $R_{\frac{\pi}{2}}T_{0,1}$ ก็เป็นการแปลงแอฟไฟน์เชิงเดียวกัน เพราะ

$$\begin{aligned} R_{\frac{\pi}{2}}T_{0,1}(\mathbf{v}) &= R_{\frac{\pi}{2}}(\mathbf{v} + (0,1)) \\ &= R_{\frac{\pi}{2}}(\mathbf{v}) + R_{\frac{\pi}{2}}((0,1)) \\ &= R_{\frac{\pi}{2}}(\mathbf{v}) + (-1,0) \\ &= T_{-1,0}R_{\frac{\pi}{2}}(\mathbf{v}) \end{aligned}$$

ข้อสังเกต

- ถ้า B เป็นการแปลงเชิงเส้นแล้ว $BT_{\mathbf{u}}$ จะเป็นการแปลงแอฟไฟน์เสมอ เนื่องจาก

$$BT_{\mathbf{u}}(\mathbf{v}) = B(\mathbf{v}+\mathbf{u}) = B(\mathbf{v}) + B(\mathbf{u}) = T_{B(\mathbf{u})}B(\mathbf{v})$$

กล่าวคือ

$$BT_{\mathbf{u}} = T_{B(\mathbf{u})}B$$

ข้อสังเกต

- เราสามารถพิสูจน์ได้ทำนองเดียวกันว่า ถ้าแต่ละตัวของการแปลง $A_1, A_2, \ldots, A_{n-1}, A_n$ เป็นการเลื่อนแกนขนานหรือการแปลงเชิงเส้นแล้ว การแปลง $A_n A_{n-1} \cdots A_2 A_1$จะเป็นการแปลงแอฟไฟน์
- ดังนั้นการแปลงแอฟไฟน์จึงเป็นกลุ่มของการแปลงที่รวม
  - การย่อขยาย
  - การหมุน
  - การเลื่อนแกนขนาน
  - การแปลงเชิงเส้น

  เอาไว้ทั้งหมด

Homogeneous Coordinates

- **Homogeneous coordinates** เป็นวิธีการแทนจุดสองมิติด้วยเวกเตอร์สามมิติแบบหนึ่ง โดยที่

  เวกเตอร์ (x,y,w) หมายถึงจุด $(x/w,y/w)$ ถ้า $w \neq 0$

- ตัวอย่าง
  - $(1,2,1)$ หมายถึงจุด $(1,2)$
  - $(2,4,2)$ หมายถึงจุด $(1,2)$ เช่นเดียวกัน
  - $(w,2w,w)$ ก็หมายถึงจุด $(1,2)$ สำหรับค่า w ใดๆ

การแทนการแปลงแอฟไฟน์ด้วยเมตริกซ์

- สมมติว่าเรามีการแปลงแอฟไฟน์ $A = T_{\mathbf{u}}B$ โดยที่

$$B = \begin{bmatrix} a & c \\ b & d \end{bmatrix}, \qquad \mathbf{u} = \begin{bmatrix} e \\ f \end{bmatrix}$$

จะได้ว่า

$$A\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} a & c \\ b & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

$$= \begin{bmatrix} ax + cy + e \\ bx + dy + f \end{bmatrix}$$

การแทนการแปลงแอฟไฟน์ด้วยเมตริกซ์

- ให้ $$N = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

  เมื่อเราคูณ N ด้วย $(x,y,1)$ ซึ่งเป็น homogeneous coordiate ของ (x,y) จะได้ว่า

$$N \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + cy + e \\ bx + dy + f \\ 1 \end{bmatrix}$$

homogeneous coordinate ของ $A((x, y))$

การแทนการแปลงแอฟไฟน์ด้วยเมตริกซ์

- ฉะนั้น affine transform คือเมตริกซ์ 3x3 ที่แถวล่างเท่ากับ $(0,0,1)$

การแทนการแปลงแอฟไฟน์ด้วยเมตริกซ์

- สังเกต

$$A = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

$$A(\mathbf{x}) - \mathbf{u} = B(\mathbf{x})$$

การแทนการแปลงแอฟไฟน์ด้วยเมตริกซ์

- สังเกต

$$A = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

$$A(\mathbf{y}) - \mathbf{u} = B(\mathbf{y})$$

การแทนการแปลงแอฟไฟน์ด้วยเมตริกซ์

- สังเกต

$$A = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{u} = A(\mathbf{0})$$

ตัวอย่าง

- $T_{0,1}R_{\frac{\pi}{2}}((0,0)) = (0,1)$
- $T_{0,1}R_{\frac{\pi}{2}}((1,0)) - (0,1) = (0,2) - (0,1) = (0,1)$
- $T_{0,1}R_{\frac{\pi}{2}}((0,1)) - (0,1) = (-1,1) - (0,1) = (-1,0)$
- ดังนั้น

$$T_{0,1}R_{\frac{\pi}{2}} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

เมตริกซ์ของการแปลงแอฟไฟน์ที่สำคัญ

$$S_{\alpha,\beta} = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_{\mathbf{u}} = \begin{bmatrix} 1 & 0 & u_1 \\ 0 & 1 & u_2 \\ 0 & 0 & 1 \end{bmatrix} \qquad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Composition และเมตริกซ์

- Composition คือการคูณเมตริกซ์
- ตัวอย่าง

$$T_{0,1}R_{\frac{\pi}{2}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

การแปลงผันกลับ

- **การแปลงผันกลับ (inverse)** ของการแปลง A คือการแปลง A^{-1} ที่ทำให้

$$AA^{-1} = A^{-1}A = I$$

- การแปลงบางตัวไม่มี inverse เช่น $A : (x, y) \mapsto (x, 0)$
- การแปลงแอฟไฟน์ A จะมี inverse ก็ต่อเมื่อเมตริกซ์ของ A มี inverse พูดอีกนัยหนึ่งคือ $\det(A) \neq 0$

การแปลงผันกลับของการแปลงแอฟไฟน์ที่สำคัญ

$$\left(S_{\alpha,\beta}\right)^{-1} = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{\alpha} & 0 & 0 \\ 0 & \frac{1}{\beta} & 0 \\ 0 & 0 & 1 \end{bmatrix} = S_{\frac{1}{\alpha},\frac{1}{\beta}}$$

$$\left(T_{\mathbf{u}}\right)^{-1} = \begin{bmatrix} 1 & 0 & u_1 \\ 0 & 1 & u_2 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & -u_1 \\ 0 & 1 & -u_2 \\ 0 & 0 & 1 \end{bmatrix} = T_{-\mathbf{u}}$$

การแปลงผันกลับของการแปลงแอฟไฟน์ที่สำคัญ

$$
\begin{aligned}
\left(R_\theta\right)^{-1} &= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \\
&= \begin{bmatrix} -\cos\theta & -\sin\theta & 0 \\ \sin\theta & -\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = R_{-\theta}
\end{aligned}
$$