418382 สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์
การบรรยายครั้งที่ 6

ประมุข ขันเงิน

pramook@gmail.com

MOUSE INPUT

การติดต่อกับเมาส์ด้วย GLUT

- ดักการกดปุ่ม
  - glutMouseFunc(…)
- ดักการเคลื่อนที่ของ mouse
  - glutMotionFunc(…)
  - glutPassiveMotionFunc(…)

glutMouseFunc

- glutMouseFunc(void (*f)(int, int, int, int))
  - ฟังก์ชัน f จะถูกเรียกทุกๆ ครั้งที่มีการ (1) กดปุ่มเมาส์ (2) ปล่อยปุ่ม mouse
- Argument ของ f มีดังต่อไปนี้
  - ตัวแรกแสดงปุ่มที่กด มีได้สามค่า
    - GLUT_LEFT_BUTTON
    - GLUT_RIGHT_BUTTON
    - GLUT_MIDDLE_BUTTON
  - ตัวที่สองแสดงว่าปุ่มถูกกดหรือถูกปล่อย มีได้สองค่า
    - GLUT_DOWN แสดงว่าปุ่มถูกกด
    - GLUT_UP แสดงว่าปุ่มถูกปล่อย
  - ตัวที่สามและตัวที่สี่คือตำแหน่ง x และ y ใน screen space ของเมาส์

ตัวอย่าง

```
void mouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON)
            printf("Left mouse button");
    else if (button == GLUT_RIGHT_BUTTON)
            printf("Right mouse button");
    else if (button == GLUT_MIDDLE_BUTTON)
            printf("Middle mouse button");
    if (state == GLUT_DOWN)
            printf(" is clicked");
    else if (state == GLUT_UP)
            printf(" is released");
    printf(" at (%d, %d)\n", x, y);
}
```

ตัวอย่าง (ต่อ)

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);

            :
            :

    glutMouseFunc(mouse);

    glutMainLoop();
    return 0;
}
```

โปรแกรมคลิกแล้วเกิดจุดบนหน้าจอ

- เวลาคลิกเมาส์ปุ่มซ้ายแล้วจะเกิดจุดสีเขียวบนหน้าจอ
- เก็บจุดที่คลิกเอาไว้ใน vector ชื่อ points

```
struct Point {
    float x, y;
    Point(float _x, float _y) {
        x = _x;
        y = _y;
    }
};

vector<Point> points;
```

โปรแกรมคลิกแล้วเกิดจุดบนหน้าจอ (ต่อ)

- เวลาคลิกเมาส์ปุ่มซ้ายให้เพิ่มจุดเข้าใน points

```
void mouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        if (x < 0 || x > screenWidth) return;
        if (y < 0 || y > screenHeight) return;
        float xx = 2*(x * 1.0f / screenWidth) - 1;
        float yy = 2*((screenHeight - y) * 1.0f / screenHeight) - 1;
        points.push_back(Point(xx,yy));
        glutPostRedisplay();
    }
}
```

โปรแกรมคลิกแล้วเกิดจุดบนหน้าจอ (ต่อ)

- เวลาวาดรูปก็ให้วาดจุดทั้งหมดที่เก็บไว้

```
void display()
{
    glClearColor(0,0,0,0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(0,1,0);
    glPointSize(3.0f);
    glBegin(GL_POINTS);
    for(int i=0;i<(int)points.size();i++)
        glVertex2f(points[i].x, points[i].y);
    glEnd();
    glutSwapBuffers();
}
```

glutMotionFunc

- glutMotionFunc(void (*f)(int, int))
  - ถูกเรียกหลักจากผู้ใช้กดปุ่มเมาส์แล้ว เมาส์เปลี่ยนตำแหน่ง
  - Argument ของ f คือพิกัด x และ y ของ mouse ใน screen space

ตัวอย่าง

```
void motion(int x, int y)
{
    printf("Mouse is at position (%d, %d).\n", x, y);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
            :
            :
            :
    glutMotionFunc(motion);
    glutMainLoop();
    return 0;
}
```

โปรแกรมลูกบอลวิ่งตามเมาส์เมื่อคลิกค้าง

- มีตัวแปร bool ชื่อ follow ไว้เก็บว่าตอนนี้ลูกบอลจะวิ่งตามเมาส์หรือเปล่า
- เก็บจุดศูนย์กลางของลูกบอลไว้ในตัวแปร cx และ cy
- เก็บตำแหน่งของเมาส์ไว้ที่ mx และ my

โปรแกรมลูกบอลวิ่งตามเมาส์เมื่อคลิกค้าง (ต่อ)

```
void animate(int id) {
    if (follow) {
        float vx, vy;
        float dx = mx - cx;
        float dy = my - cy;
        float l = sqrtf(dx*dx + dy*dy);
        if (l < 0.000001f) { vx = 0;  vy = 0; }
        if (l < 0.05f)  {  vx = dx;  vy = dy;  }
        else { vx = 0.05f*dx / l; vy = 0.05f*dy / l;  }
        cx += vx; cy += vy;
        if (cx < -1) cx = -1;  if (cx > 1) cx = 1;
        if (cy < -1) cy = -1;  if (cy > 1) cy = 1;
    }
    glutPostRedisplay();
    glutTimerFunc(INTERVAL, animate, 0);
}
```

หาการขจัดและขนาดของการขจัด

โปรแกรมลูกบอลวิ่งตามเมาส์เมื่อคลิกค้าง (ต่อ)

```
void animate(int id) {
    if (follow) {
        float vx, vy;
        float dx = mx - cx;
        float dy = my - cy;
        float l = sqrtf(dx*dx + dy*dy);
        if (l < 0.000001f) { vx = 0; vy = 0; }
        if (l < 0.05f) { vx = dx; vy = dy; }
        else { vx = 0.05f*dx / l; vy = 0.05f*dy / l; }
        cx += vx; cy += vy;
        if (cx < -1) cx = -1;  if (cx > 1) cx = 1;
        if (cy < -1) cy = -1;  if (cy > 1) cy = 1;
    }
    glutPostRedisplay();
    glutTimerFunc(INTERVAL, animate, 0);
}
```

ถ้าลูกบอลอยู่ที่เดียวกับเมาส์แล้วก็ไม่ต้องให้มันเคลื่อนที่

โปรแกรมลูกบอลวิ่งตามเมาส์เมื่อคลิกค้าง (ต่อ)

```
void animate(int id) {
    if (follow) {
        float vx, vy;
        float dx = mx - cx;
        float dy = my - cy;
        float l = sqrtf(dx*dx + dy*dy);
        if (l < 0.000001f) { vx = 0;  vy = 0; }
        if (l < 0.05f) { vx = dx;  vy = dy; }
        else { vx = 0.05f*dx / l; vy = 0.05f*dy / l;  }
        cx += vx; cy += vy;
        if (cx < -1) cx = -1;  if (cx > 1) cx = 1;
        if (cy < -1) cy = -1;  if (cy > 1) cy = 1;
    }
    glutPostRedisplay();
    glutTimerFunc(INTERVAL, animate, 0);
}
```

ถ้าลูกบอลอยู่ใกล้มากก็ให้ความเร็วของลูกบอลมีค่าเท่ากับการขจัด (ซึ่งมีค่าน้อยมาก)

โปรแกรมลูกบอลวิ่งตามเมาส์เมื่อคลิกค้าง (ต่อ)

```
void animate(int id) {
    if (follow) {
        float vx, vy;
        float dx = mx - cx;
        float dy = my - cy;
        float l = sqrtf(dx*dx + dy*dy);
        if (l < 0.000001f) { vx = 0;  vy = 0; }
        if (l < 0.05f)  {  vx = dx;  vy = dy;  }
        else { vx = 0.05f*dx / l; vy = 0.05f*dy / l;  }
        cx += vx; cy += vy;
        if (cx < -1) cx = -1;  if (cx > 1) cx = 1;
        if (cy < -1) cy = -1;  if (cy > 1) cy = 1;
    }
    glutPostRedisplay();
    glutTimerFunc(INTERVAL, animate, 0);
}
```

มีเช่นนั้นให้ลูกบอลมีอัตราเร็ว 0.05 หน่วยและมีความเร็วพุ่งไปยังตำแหน่งของเมาส์

โปรแกรมลูกบอลวิ่งตามเมาส์เมื่อคลิกค้าง (ต่อ)

```
void animate(int id) {
    if (follow) {
        float vx, vy;
        float dx = mx - cx;
        float dy = my - cy;
        float l = sqrtf(dx*dx + dy*dy);
        if (l < 0.000001f) { vx = 0;  vy = 0; }
        if (l < 0.05f)  {  vx = dx;  vy = dy;  }
        else { vx = 0.05f*dx / l; vy = 0.05f*dy / l;  }
        cx += vx; cy += vy;
        if (cx < -1) cx = -1;  if (cx > 1) cx = 1;
        if (cy < -1) cy = -1;  if (cy > 1) cy = 1;
    }
    glutPostRedisplay();
    glutTimerFunc(INTERVAL, animate, 0);
}
```

ทำให้ลูกบอลเคลื่อนที่

โปรแกรมลูกบอลวิ่งตามเมาส์เมื่อคลิกค้าง (ต่อ)

```
void animate(int id) {
    if (follow) {
        float vx, vy;
        float dx = mx - cx;
        float dy = my - cy;
        float l = sqrtf(dx*dx + dy*dy);
        if (l < 0.000001f) { vx = 0;  vy = 0; }
        if (l < 0.05f)  {  vx = dx;  vy = dy;  }
        else { vx = 0.05f*dx / l; vy = 0.05f*dy / l;  }
        cx += vx; cy += vy;
        if (cx < -1) cx = -1;  if (cx > 1) cx = 1;
        if (cy < -1) cy = -1;  if (cy > 1) cy = 1;
    }
    glutPostRedisplay();
    glutTimerFunc(INTERVAL, animate, 0);
}
```

กันลูกบอลตกขอบ

โปรแกรมลูกบอลวิ่งตามเมาส์เมื่อคลิกค้าง (ต่อ)

- เมื่อผู้ใช้คลิก ให้เปลี่ยน follow เป็น true
- เมื่อผู้ใช้ปล่อย mouse ให้เปลี่ยน follow เป็น false

```
void mouse(int button, int state, int x, int y) {
    if (state == GLUT_DOWN) {
        follow = true;
        setTarget(x,y);
        glutPostRedisplay();
    }
    else
        follow = false;
}
```

โปรแกรมลูกบอลวิ่งตามเมาส์เมื่อคลิกค้าง (ต่อ)

- setTarget(int x, int y) มีไว้เซต mx และ my ให้ตรงกับตำแหน่งของ mouse

```
void setTarget(int x, int y)
{
    if (x < 0 || x > screenWidth) return;
    if (y < 0 || y > screenHeight) return;
    mx = 2*(x * 1.0f / screenWidth) - 1;
    my = 2*((screenHeight - y) * 1.0f / screenHeight) - 1;
}
```

โปรแกรมลูกบอลวิ่งตามเมาส์เมื่อคลิกค้าง (ต่อ)

- ฟังก์ชันที่ให้ glutMotionFunc มีหน้าที่เรียก setTarget เพื่อเซต mx และ my เมื่อเมาส์เคลื่อนที่

```
void motion(int x, int y)
{
   setTarget(x,y);
}
```

glutPassiveMotionFunc

- glutPassiveMotionFunc(void (*f)(int, int))
  - ถูกเรียกเมื่อเมาส์เคลื่อนที่ แม้ไม่ได้การกดปุ่มเมาส์ก็ตาม
  - Argument ทั้งสองของฟังก์ชัน f คือพิกัด x และ y ใน screen space ของเมาส์

ตัวอย่าง

```
void passive_motion(int x, int y)
{
    printf("Mouse is at position (%d, %d).\n", x, y);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
            :
            :
            :
    glutPassiveMotionFunc(passive_motion);
    glutMainLoop();
    return 0;
}
```

โปรแกรมลูกบอลวิ่งตามเมาส์

- เหมือนกับโปรแกรมที่แล้ว แต่ไม่มีตัวแปร follow และใช้ glutPassiveMotionFunc แทน

```
void passive_motion(int x, int y)
{
   setTarget(x,y);
}
```

โปรแกรมลูกบอลวิ่งตามเมาส์ (ต่อ)

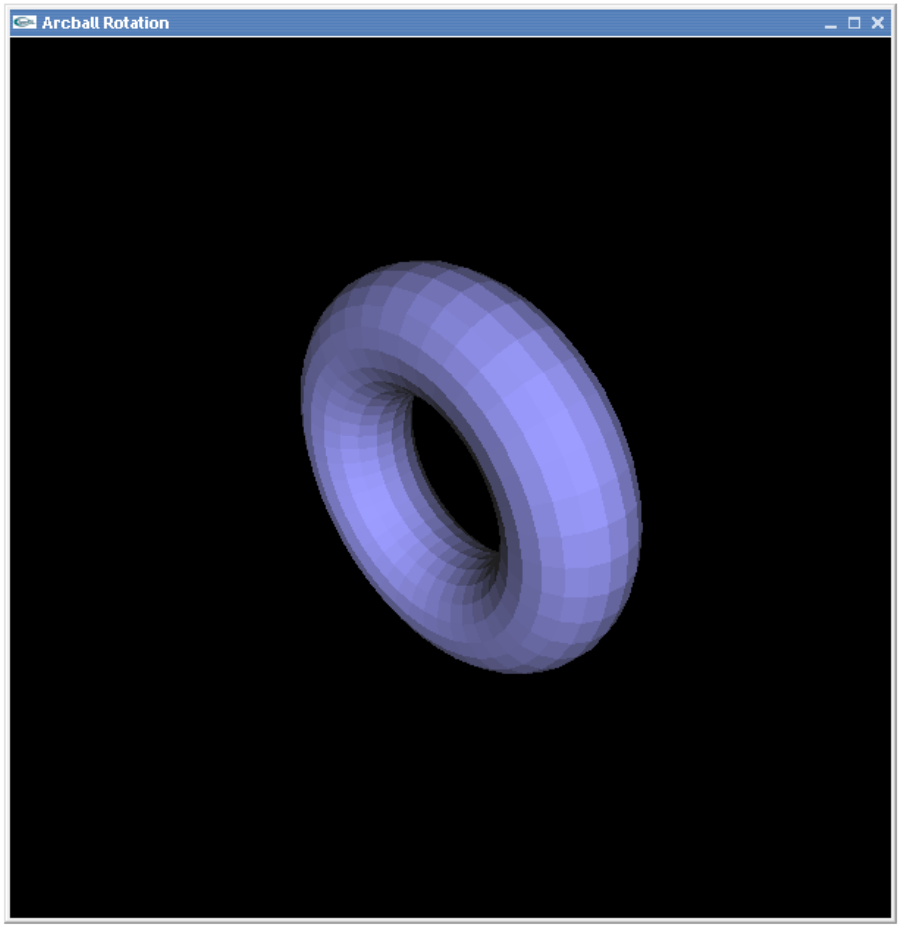
```
void animate(int id) {
    float vx, vy;
    float dx = mx - cx;
    float dy = my - cy;
    float l = sqrtf(dx*dx + dy*dy);
    if (l < 0.000001f) { vx = 0; vy = 0; }
    if (l < 0.05f) { vx = dx; vy = dy; }
    else { vx = 0.05f*dx / l; vy = 0.05f*dy / l; }
    cx += vx; cy += vy;
    if (cx < -1) cx = -1; if (cx > 1) cx = 1;
    if (cy < -1) cy = -1; if (cy > 1) cy = 1;
    glutPostRedisplay();
    glutTimerFunc(INTERVAL, animate, 0);
}
```

ARCBALL ROTATION

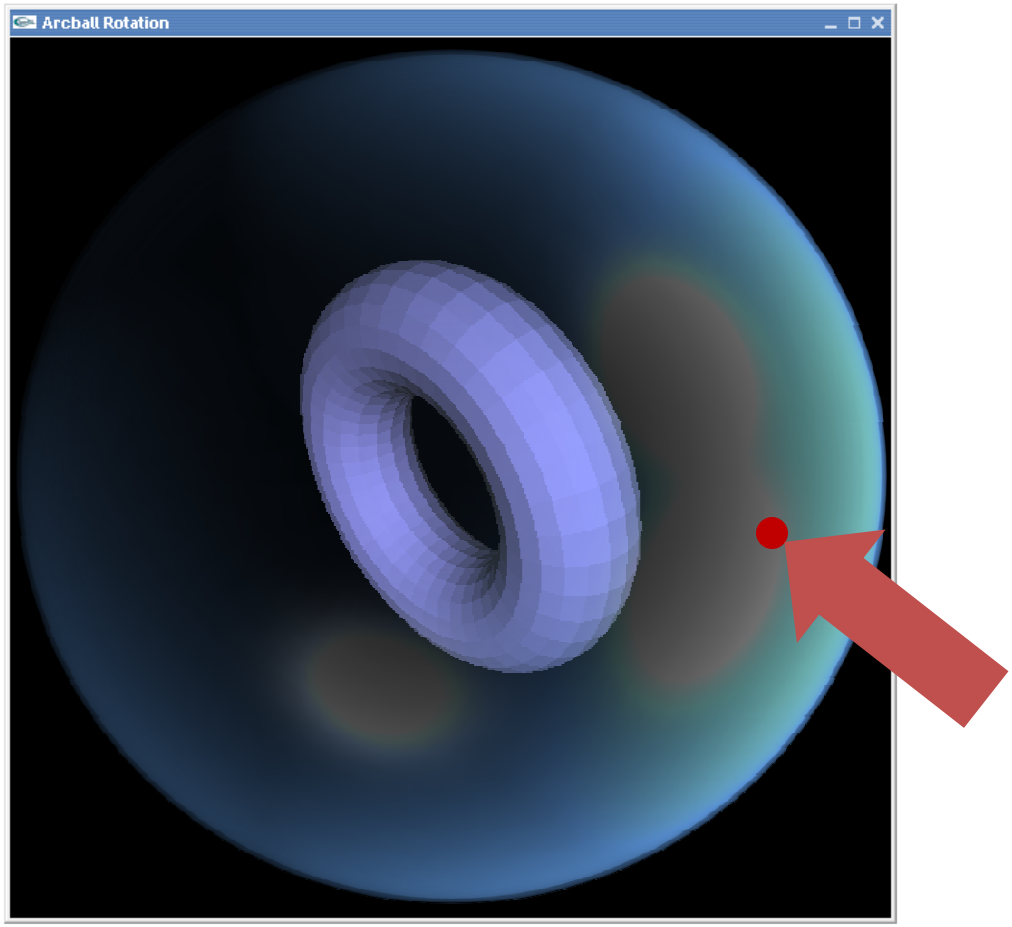
Arcball Rotation

- ทำให้ผู้ใช้สามารถ “หมุน” โมเดล อย่างเป็นธรรมชาติ
- คิดว่ามีทรงกลมอยู่ตรงกลางฉาก
- ผู้ใช้คลิกจุดบนทรงกลม แล้วลากจุดบนทรงกลมนั้น ทำให้เกิดการหมุน

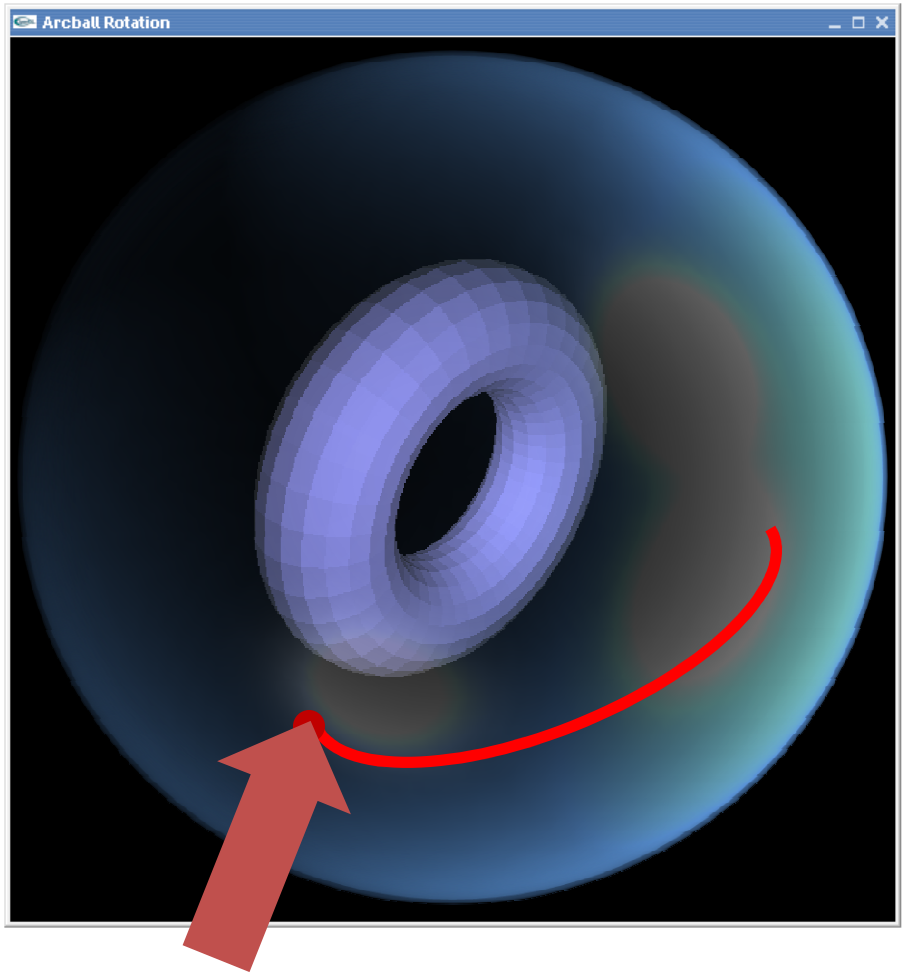
Arcball Rotation



Arcball Rotation (ต่อ)



Arcball Rotation (ต่อ)



Arcball Rotation (ต่อ)

- เวลาแสดงผล เก็บ rotation matrix เอาไว้
- rotation matrix เป็น matrix ขนาด 4x4
- เก็บได้ด้วย array ของจำนวน 16 ตัว
- OpenGL เก็บ matrix โดยเรียงตัวเลขแบบ column major
- กล่าวคือ สมมติเราใช้ double m[16] เก็บมัน ตัวเลขจะเรียงแบบนี้

$$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{pmatrix}$$

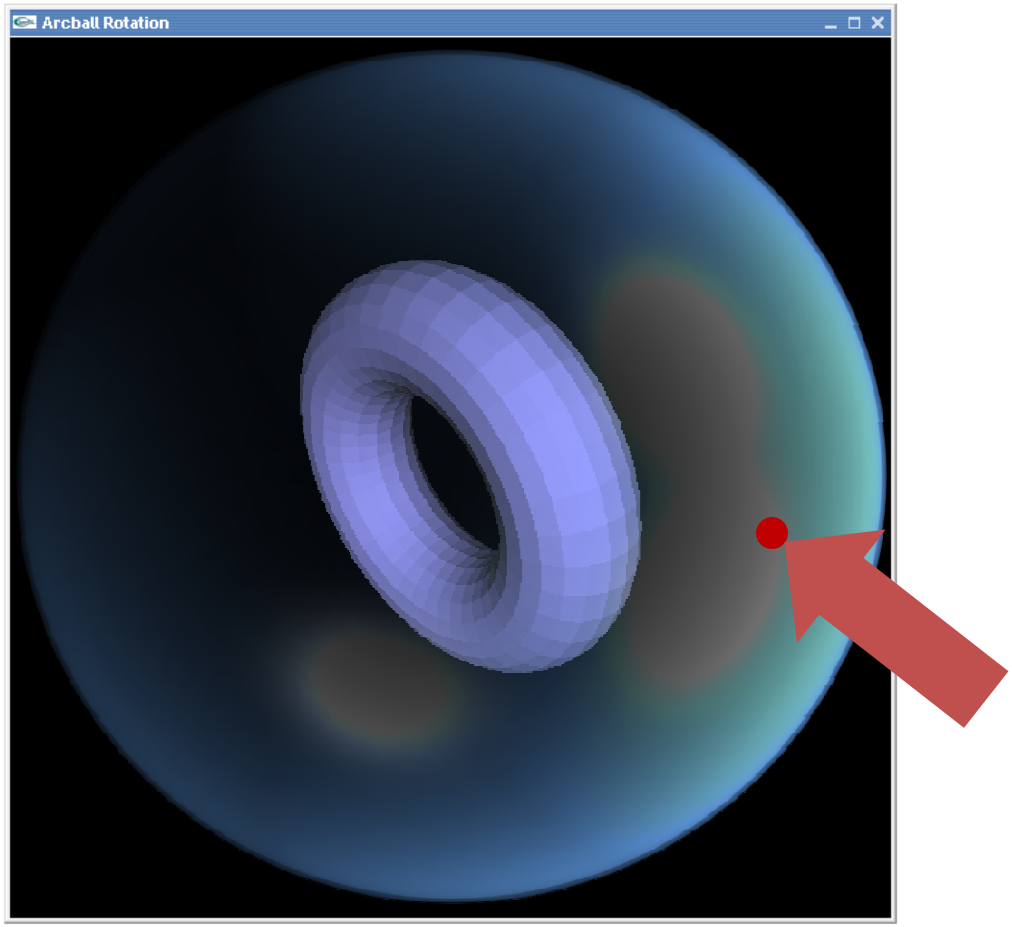
Arcball Rotation (ต่อ)

- เราเก็บ rotation matrix ปัจจุบันไว้ใน
  double currentRotation[16];
- เวลาวาดโมเดลก็นำ currentRotation ไปคูณกับ modelview matrix ก่อนด้วยคำสั่ง glMultMatrixd
- การคูณนี้ทำหลังจากคูณ modelview matrix ด้วย view matrix แล้ว

Arcball Rotation (ต่อ)

```
void display()
{
    glClearColor(0,0,0,0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0,0,5,0,0,0,0,1,0);

    glMultMatrixd(currentRotation);
    drawSomething();

    glutSwapBuffers();
}
```

ถ้าผู้ใช้คลิกตรงนี้ มันตรงกับจุดไหนของทรงกลม?



ถ้าผู้ใช้คลิกตรงนี้ มันตรงกับจุดไหนของทรงกลม? (ต่อ)

- เวลาคลิก เรารู้ screen space coordinate (x,y) ของเมาส์
- เพื่อความง่าย เรากำหนด object space ของทรงกลม
  - ทรงกลมมีรัศมี 1
  - ตรงกลางหน้าจอคือจุด (0,0,0)
  - ขอบซ้ายของหน้าต่าง x = -1, ขอบขวา x = 1
  - ขอบล่างของหน้าต่าง y = -1, ขอบบน y = 1
- ดังนั้นได้ว่า ถ้า x_s และ y_s คือพิกัดใน object space ของ sphere

$$x_s = 2\frac{x}{w} - 1$$

$$y_s = 2\frac{h - y}{h} - 1$$

ถ้าผู้ใช้คลิกตรงนี้ มันตรงกับจุดไหนของทรงกลม? (ต่อ)

- แล้ว z_s เท่ากับเท่าไหร่?
- สมการวงกลม $x_s^2 + y_s^2 + z_s^2 = 1$ ดังนั้น $z_s = \sqrt{1 - x_s^2 + y_s^2}$
- แต่ว่า $x_s^2 + y_s^2$ อาจมีค่ามากกว่า 1 ได้!
  - ในกรณีนี้ให้เราให้ $z_s = 0$
  - แล้วเซต

$$x_s = \frac{x_s}{\sqrt{x_s^2 + y_s^2}}$$

$$y_s = \frac{y_s}{\sqrt{x_s^2 + y_s^2}}$$

ถ้าผู้ใช้คลิกตรงนี้ มันตรงกับจุดไหนของทรงกลม? (ต่อ)

```
void onSphere(double p[], int x, int y)
{
    double xs = 2*(x * 1.0 / screenWidth) - 1;
    double ys = 2*((screenHeight - y) * 1.0 / screenHeight) - 1;
    double l = xs*xs + ys*ys;
    if (l > 1)
    {
        xs /= sqrt(l);
        ys /= sqrt(l);
        l = 1;
    }
    double zs = sqrt(1 - l*l);

    p[0] = xs; p[1] = ys; p[2] = zs;
}
```

ถ้าผู้ใช้คลิกตรงนี้ มันตรงกับจุดไหนของทรงกลม? (ต่อ)

```
void onSphere(double p[], int x, int y)
{
    double xs = 2*(x * 1.0 / screenWidth) - 1;
    double ys = 2*((screenHeight - y) * 1.0 / screenHeight) - 1;
    double l = xs*xs + ys*ys;
    if (l > 1)
    {
        xs /= sqrt(l);
        ys /= sqrt(l);
        l = 1;
    }
    double zs = sqrt(1 - l*l);

    p[0] = xs; p[1] = ys; p[2] = zs;
}
```

คำนวณ x_s, y_s

ถ้าผู้ใช้คลิกตรงนี้ มันตรงกับจุดไหนของทรงกลม? (ต่อ)

```
void onSphere(double p[], int x, int y)
{
    double xs = 2*(x * 1.0 / screenWidth) - 1;
    double ys = 2*((screenHeight - y) * 1.0 / screenHeight) - 1;
    double l = xs*xs + ys*ys;
    if (l > 1)
    {
        xs /= sqrt(l);
        ys /= sqrt(l);
        l = 1;
    }
    double zs = sqrt(1 - l*l);

    p[0] = xs; p[1] = ys; p[2] = zs;
}
```

คำนวณ $x_s^2 + y_s^2$

ถ้าผู้ใช้คลิกตรงนี้ มันตรงกับจุดไหนของทรงกลม? (ต่อ)

```
void onSphere(double p[], int x, int y)
{
    double xs = 2*(x * 1.0 / screenWidth) - 1;
    double ys = 2*((screenHeight - y) * 1.0 / screenHeight) - 1;
    double l = xs*xs + ys*ys;
    if (l > 1)
    {
        xs /= sqrt(l);
        ys /= sqrt(l);
        l = 1;
    }
    double zs = sqrt(1 - l*l);

    p[0] = xs; p[1] = ys; p[2] = zs;
}
```

ถ้า $x_s^2 + y_s^2 > 1$ ให้เซต

$$x_s = \frac{x_s}{\sqrt{x_s^2 + y_s^2}}$$

$$y_s = \frac{y_s}{\sqrt{x_s^2 + y_s^2}}$$

ถ้าผู้ใช้คลิกตรงนี้ มันตรงกับจุดไหนของทรงกลม? (ต่อ)

```
void onSphere(double p[], int x, int y)
{
    double xs = 2*(x * 1.0 / screenWidth) - 1;
    double ys = 2*((screenHeight - y) * 1.0 / screenHeight) - 1;
    double l = xs*xs + ys*ys;
    if (l > 1)
    {
        xs /= sqrt(l);
        ys /= sqrt(l);
        l = 1;
    }
    double zs = sqrt(1 - l*l);

    p[0] = xs; p[1] = ys; p[2] = zs;
}
```

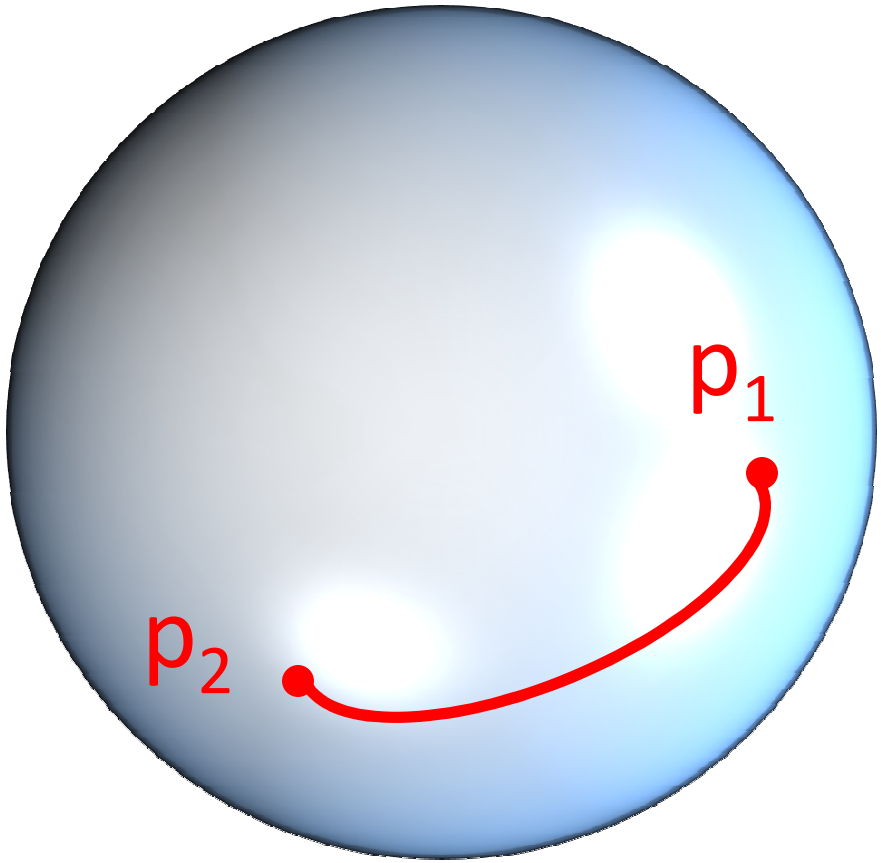
คำนวณ $z_s = \sqrt{1 - x_s^2 + y_s^2}$

ถ้าผู้ใช้คลิกตรงนี้ มันตรงกับจุดไหนของทรงกลม? (ต่อ)

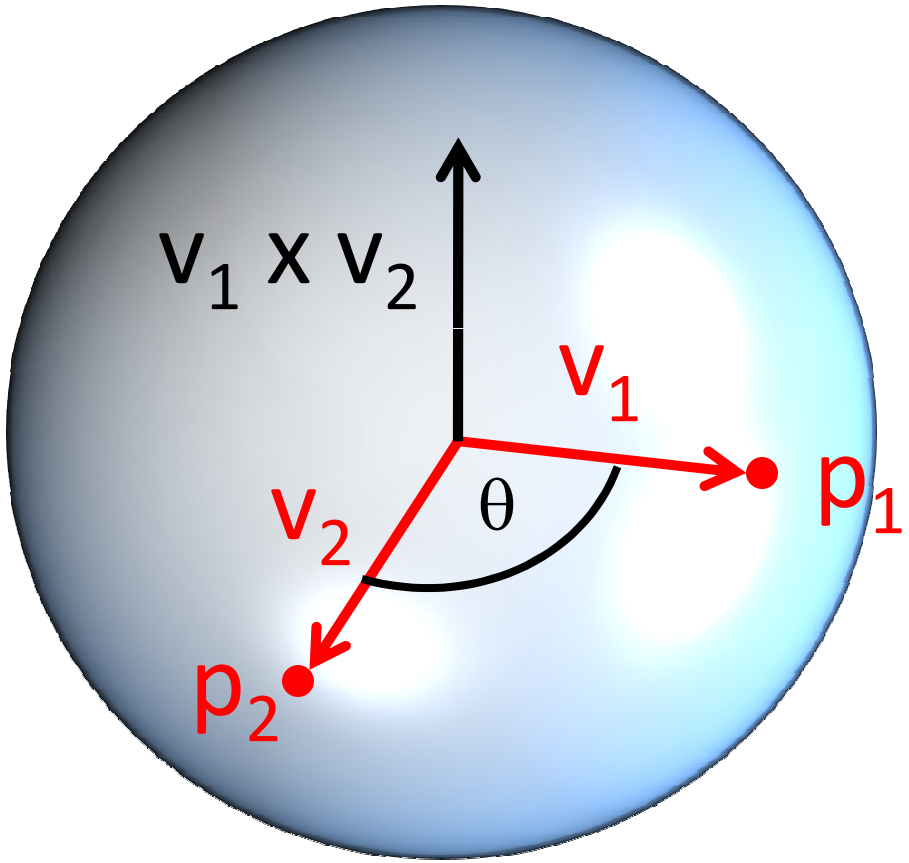
```
void onSphere(double p[], int x, int y)
{
    double xs = 2*(x * 1.0 / screenWidth) - 1;
    double ys = 2*((screenHeight - y) * 1.0 / screenHeight) - 1;
    double l = xs*xs + ys*ys;
    if (l > 1)
    {
        xs /= sqrt(l);
        ys /= sqrt(l);
        l = 1;
    }
    double zs = sqrt(1 - l*l);

    p[0] = xs; p[1] = ys; p[2] = zs;
}
```

คืนค่ากลับไปใน array ที่ให้มา

ถ้าผู้ใช้คลิกเมาส์จุด p_1 แล้วลากไปจุด p_2 แล้ว
มันตรงกับการหมุนแกนอะไร? ด้วยมุมเท่าไหร่?



ถ้าผู้ใช้คลิกเมาส์จุด p_1 แล้วลากไปจุด p_2 แล้ว
มันตรงกับการหมุนแกนอะไร? ด้วยมุมเท่าไหร่? (ต่อ)



ถ้าผู้ใช้คลิกเมาส์จุด p_1 แล้วลากไปจุด p_2 แล้วมันตรงกับการหมุนแกนอะไร? ด้วยมุมเท่าไหร่?

- ให้ v_1 เป็นเวกเตอร์ที่ลากจากจุดศูนย์กลางไปยัง p_1
  ให้ v_2 เป็นเวกเตอร์ที่ลากจากจุดศูนย์กลางไปยัง p_2
- แล้วการหมุนจะเป็นการหมุนรอบแกน $v_1 \times v_2$
  ด้วยมุม $\theta = \arccos(v_1 \cdot v_2)$

ฟังก์ชัน arcball

- void arcball(int x, int y)
  - ทำหน้าที่เปลี่ยน rotation matrix ที่เก็บใน currentRotation
  - เงื่อนไข
    - x และ y เป็น screen coordinate ของตำแหน่งเมาส์ใหม่
    - ตำแหน่งเมาส์เดิมเก็บอยู่ที่ startX, startY
    - ตัวแปร startRotation เก็บ rotation matrix ในตอนที่เมาส์อยู่ที่ตำแหน่ง (startX, startY)

ฟังก์ชัน arcball (ต่อ)

```
void arcball(int x, int y)
{
	if (x == startX && y == startY)
		return;

	double  v1[3], v2[3];
	onSphere(v1, startX, startY);
	onSphere(v2, x, y);

	double axis[3];
	cross(axis, v1, v2);
	normalize(axis);
	double cosTheta = dot(v1, v2);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	glRotated(acos(cosTheta) / M_PI * 180, axis[0], axis[1], axis[2]);
	glMultMatrixd(startRotation);
	glGetDoublev(GL_MODELVIEW_MATRIX, currentRotation);
}
```

ฟังก์ชัน arcball (ต่อ)

```
void arcball(int x, int y)
{
    if (x == startX && y == startY)
        return;

    double v1[3], v2[3];
    onSphere(v1, startX, startY);
    onSphere(v2, x, y);

    double axis[3];
    cross(axis, v1, v2);
    normalize(axis);
    double cosTheta = dot(v1, v2);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotated(acos(cosTheta) / M_PI * 180, axis[0], axis[1], axis[2]);
    glMultMatrixd(startRotation);
    glGetDoublev(GL_MODELVIEW_MATRIX, currentRotation);
}
```

ถ้าเมาส์ไม่เลื่อนก็ไม่ต้องทำอะไร

ฟังก์ชัน arcball (ต่อ)

```
void arcball(int x, int y)
{
    if (x == startX && y == startY)
        return;

    double v1[3], v2[3];
    onSphere(v1, startX, startY);
    onSphere(v2, x, y);

    double axis[3];
    cross(axis, v1, v2);
     normalize(axis);
    double cosTheta = dot(v1, v2);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotated(acos(cosTheta) / M_PI * 180, axis[0], axis[1], axis[2]);
    glMultMatrixd(startRotation);
    glGetDoublev(GL_MODELVIEW_MATRIX, currentRotation);
}
```

คำนวณ v_1 และ v_2

ฟังก์ชัน arcball (ต่อ)

```
void arcball(int x, int y)
{
	if (x == startX && y == startY)
		return;

	double v1[3], v2[3];
	onSphere(v1, startX, startY);
	onSphere(v2, x, y);

	double axis[3];
	cross(axis, v1, v2);
	normalize(axis);
	double cosTheta = dot(v1, v2);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	glRotated(acos(cosTheta) / M_PI * 180, axis[0], axis[1], axis[2]);
	glMultMatrixd(startRotation);
	glGetDoublev(GL_MODELVIEW_MATRIX, currentRotation);
}
```

คำนวณแกน $V_1 \times V_2$ และทำให้มันเป็นเวกเตอร์หนึ่งหน่วย

ฟังก์ชัน arcball (ต่อ)

```
void arcball(int x, int y)
{
	if (x == startX && y == startY)
		return;

	double v1[3], v2[3];
	onSphere(v1, startX, startY);
	onSphere(v2, x, y);

	double axis[3];
	cross(axis, v1, v2);
	 normalize(axis);
	double cosTheta = dot(v1, v2);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	glRotated(acos(cosTheta) / M_PI * 180, axis[0], axis[1], axis[2]);
	glMultMatrixd(startRotation);
	glGetDoublev(GL_MODELVIEW_MATRIX, currentRotation);
}
```

คำนวณแกน cos θ

ฟังก์ชัน arcball (ต่อ)

```
void arcball(int x, int y)
{
	if (x == startX && y == startY)
		return;

	double v1[3], v2[3];
	onSphere(v1, startX, startY);
	onSphere(v2, x, y);

	double axis[3];
	cross(axis, v1, v2);
	 normalize(axis);
	double cosTheta = dot(v1, v2);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	glRotated(acos(cosTheta) / M_PI * 180, axis[0], axis[1], axis[2]);
	glMultMatrixd(startRotation);
	glGetDoublev(GL_MODELVIEW_MATRIX, currentRotation);
}
```

คูณ matrix ใหม่เข้ากับ matrix เดิม

ฟังก์ชัน arcball (ต่อ)

```
void arcball(int x, int y)
{
	if (x == startX && y == startY)
		return;

	double v1[3], v2[3];
	onSphere(v1, startX, startY);
	onSphere(v2, x, y);

	double axis[3];
	cross(axis, v1, v2);
	 normalize(axis);
	double cosTheta = dot(v1, v2);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	glRotated(acos(cosTheta) / M_PI * 180, axis[0], axis[1], axis[2]);
	glMultMatrixd(startRotation);
	glGetDoublev(GL_MODELVIEW_MATRIX, currentRotation);
}
```

แล้วเอาค่าใหม่ใส่คืนไปที่

currentRotation

Arcball Rotation (ต่อ)

- เวลาคลิกเมาส์
  - เซตค่า startX และ startY
  - ก็อปค่าของ currentRotation มาใส่ startRotation
  - บันทึกว่ามีการคลิกเมาส์อยู่

Arcball Rotation (ต่อ)

```
void mouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        for(int i=0;i<16;i++)
            startRotation[i] = currentRotation[i];
        startX = x;
        startY = y;
        clicked = true;
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_UP)
        clicked = false;
}
```

Arcball Rotation (ต่อ)

- เวลาเมาส์เลื่อน
  - เรียก arcball

```
void motion(int x, int y)
{
    if (clicked)
    {
        arcball(x,y);
        glutPostRedisplay();
    }
}
```

การเขียนโปรแกรมเชิงออปเจกต์กับ
การจัดการข้อมูลทางคอมพิวเตอร์กราฟิกส์

ข้อมูลทางคอมพิวเตอร์กราฟิกส์

- มีอยู่หลายชนิดด้วยกัน
  - จุด
  - เวกเตอร์
  - เมตริกซ์
  - การแปลง
  - Texture coordinate
  - สี
  - รูปภาพ
  - รูปทรง
  - Texture mapping
  - Material
  - Material mapping
  - Scene graph
  - มุมกล้อง
  - ฉาก
  - ฯลฯ

การเขียนโปรแกรมเพื่อจัดการข้อมูลเหล่านี้

- เราจะแทนข้อมูลทุกแบบที่ว่าข้างต้นแต่ละอย่างด้วย object
- ข้อมูลแต่ละประเภทจะมี class ของมันเอง
- โค้ดของวันนี้ให้ดูได้ในโค้ดตัวอย่าง
- กรุณาดูด้วยเพราะคุณจะต้องใช้โค้ดเหล่านี้ทำการบ้าน

ข้อมูลพื้นฐาน

- เวกเตอร์, จุด, สี → Float3
- Texture Coordinate → Float2
- เมตริกซ์ → Float4x4
- การแปลง →Xform

เวกเตอร์

```
struct Float3
{
  union
  {
    float d[3];
    struct { float x, y, z; };
    struct { float r, g, b; };
    struct { float s, t, u; };
    struct { float alpha, beta, gamma; };
  };

  // Other parts of the struct omitted.
}
```

สิ่งที่คุณสามารถทำได้กับ Float3

- สร้างมันขึ้นมา
  - มี constructor เพื่ออำนวยความสะดวกหลายแบบ
    - `Float3();`
      - สร้างให้ component ทุกตัวมีค่าเท่ากับ 0
    - `Float3(float c);`
      - สร้างให้ component ทุกตัวมีค่าเท่ากับ c
    - `Float3(float _x, float _y, float _z);`
      - กำหนดค่าให้ component ทุกตัว
- ตัวอย่าง

```
Float3 a();          // a = (0,0,0)
Float3 b(1);         // b = (1,1,1)
Float3 c(1,2,3);     // c = (1,2,3)
```

สิ่งที่คุณสามารถทำได้กับ Float3

- บวกมัน
  - โค้ด

```
Float3 operator + (const Float3 &x) const
{
  return Float3(d[0] + x[0],
    d[1] + x[1], d[2] + x[2]);
}
```

- ตัวอย่าง

```
Float3 a(1);        // a = (1,1,1)
Float3 b(1,2,3);    // b = (1,2,3)
Float3 c = a+b;     // c = (2,3,4)
```

สิ่งที่คุณสามารถทำได้กับ Float3

- ลบมัน
  - โค้ด

```
Float3 operator - (const Float3 &x) const
{
  return Float3(d[0] - x[0],
    d[1] - x[1], d[2] - x[2]);
}
```

  - ตัวอย่าง

```
Float3 a(1);        // a = (1,1,1)
Float3 b(1,2,3);    // b = (1,2,3)
Float3 c = a-b;     // c = (0,-1,-2)
```

สิ่งที่คุณสามารถทำได้กับ Float3

- คูณมันด้วยสเกลาร์
  - โค้ด

```
Float3 operator * (float c) const
{
  return Float3(d[0]*c, d[1]*c, d[2]*c);
}
```

  - ตัวอย่าง

```
Float3 a(1,0,2);    // a = (1,0,2)
Float3 b = a*1.5f; // b = (1.5,0,3)
```

สิ่งที่คุณทำได้กับ Float3

- คูณมัน (ทีละมิติ)
  - โค้ด

```
Float3 operator * (const Float3 &x) const
{
   return Float3(d[0] * x[0],
     d[1] * x[1], d[2] * x[2]);
}
```

  - ตัวอย่าง

```
Float3 a(2,2,2);  // a = (2,2,2)
Float3 b(1,2,3);  // b = (1,2,3)
Float3 c = a*b;   // c = (2,4,6)
```

สิ่งที่คุณสามารถทำได้กับ Float3

- หาความยาว
  - โค้ด

```
float length() const
{
    return sqrtf(x*x + y*y + z*z);
}
```

  - ตัวอย่าง

```
Float3 a(3,4,0);        // a = (3,4,0)
float l = a.length();   // l = 5
```

สิ่งที่คุณสามารถทำได้กับ Float3

- คำนวณ dot product
  - โค้ด

```
float dot(const Float3 &v1, const Float3 &v2)
{
 return v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
}
```

  - ตัวอย่าง

```
Float3 a(3,4,0);          // a = (3,4,0)
Float3 b(1,-1,5);         // b = (1,-1,5)
float d = dot(a,b);       // d = -1
```

สิ่งที่คุณสามารถทำได้กับ Float3

- คำนวณ cross product
  - โค้ด

```
inline Float3 cross(
    const Float3 &v1, const Float3 &v2)
{
    return Float3(
        (v1.y * v2.z) - (v1.z * v2.y),
        (v1.z * v2.x) - (v1.x * v2.z),
        (v1.x * v2.y) - (v1.y * v2.x));
}
```

  - ตัวอย่าง

```
Float3 a(1,0,0);        // a = (1,0,0)
Float3 b(0,1,0);        // b = (0,1,0)
Float3 c = cross(a,b) // c = (0,0,1)
```

สิ่งที่คุณสามารถทำได้กับ Float3

- ทำให้เป็นเวกเตอร์หนึ่งหน่วย
  - โค้ด

```
inline Float3 normalize(const Float3 &v)
{
  return v / v.length();
}
```

  - ตัวอย่าง

```
Float3 a(1,1,1);          // a = (1,1,1)
Float3 b = normalize(a)
     // b = (0.717,0.717,0.717)
```

เมตริกซ์

- เราสนใจเฉพาะเมตริกซ์ขนาด 4x4 เท่านั้น
  - เนื่องจากการแปลงที่เราสนใจทั้งหมดสามารถแทนได้ด้วยเมตริกซ์ 4x4
    - Affine transformation
    - Look-at transformation
    - Perspective projection
- เราเก็บเมตริกซ์ในอะเรย์ 2 มิติ m ขนาด 4x4
- เราเรียงสมาชิกในเมตริกซ์ตามลำดับ row-major
  - m[0][0] คือ สมาชิกในแถวแรก คอลัมน์แรก
  - m[0][1] คือ สมาชิกในแถวแรก คอลัมน์ที่สอง
  - m[1][2] คือ สมาชิกในแถวที่สอง คอลัมน์ที่สาม
  - m[3][3] คือ สมาชิกในแถวที่สี่ คอลัมน์ที่สี่
- การเรียงสมาชิกแบบนี้แตกต่างกับของ OpenGL ที่เรียงแบบ column-major

เมตริกซ์

```
struct Float4x4
{
private:
    float m[4][4];

    // Other parts of the struct omitted.
};
```

สิ่งที่คุณสามารถทำได้กับเมตริกซ์

- สร้างมันขึ้นมา
  - มี constructor อำนวยความสะดวกหลายแบบ
- ตัวอย่าง

```
Float4x4 A;      // All elements are 0.
Float4x4 B(1);   // All elements are 1.
Float4x4 C(1,2,3,4,
           5,6,7,8,
           9,1,2,3,
           4,5,6,7);
float m = {{1,0,0,0},
           {0,1,0,0},
           {0,0,1,0},
           {0,0,0,1}};
Matrix4x4 D(m);
```

สิ่งที่คุณทำได้กับเมตริกซ์

- เอาข้อมูลของมันใส่อะเรย์แบบ column-major
  - โค้ด

```
inline void fill_column_major_array(float *result) const
{
  FOR(col, 4)
    FOR(row, 4)
      result[4*col + row] = m[row][col];
}
```

  - ตัวอย่าง

```
int a[16];
Float4x4 M(1,0,0,1,0,1,0,2,0,0,1,3,0,0,0,1);
M.fill_column_major_array(a);
```

สิ่งที่คุณสามารถทำได้กับเมตริกซ์

- บวกเมตริกซ์
- ลบเมตริกซ์
- คูณเมตริกซ์ด้วยสเกลาร์
- คูณเมตริกซ์กับเมตริกซ์

สิ่งที่คุณทำได้กับเมตริกซ์

- ตัวอย่าง

```
Float4x4 A(2,0,0,0,0,2,0,0,0,0,2,0,0,0,0,2);
Float4x4 B(0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0);
Float4x4 C = A+B;
// C = [[2,1,0,0],[0,2,1,0],[0,0,2,1],[1,0,0,2]]
Float4x4 D = A-B;
// D = [[2,-1,0,0],[0,2,-1,0],[0,0,2,-1],[-1,0,0,2]]
Float4x4 E = A*2;
// E = [[4,0,0,0],[0,4,0,0],[0,0,4,0],[0,0,0,4]]
Float4x4 F = A*B;
// F = [[0,2,0,0],[0,0,2,0],[0,0,0,2],[2,0,0,0]]
```

สิ่งที่คุณทำได้กับเมตริกซ์

- คูณเมตริกซ์กับจุด
- คูณเมตริกซ์กับเวกเตอร์
- สังเกต
  - เมตริกซ์มีขนาด 4x4 แต่จุดและเวกเตอร์เป็นเมตริกซ์ขนาด 3x1
    - ฉะนั้นโดยธรรมชาติแล้วมันคูณกันไม่ได้
  - แต่เราจะคูณเมตริกซ์ด้วย homogeneous coordinate ของจุดและเวกเตอร์ ซึ่งเป็นเมตริกซ์ขนาด 4x1
  - ระวัง: HC ของจุดและเวกเตอร์นั้นต่างกัน
    - HC ของจุด (x,y,z) คือ (x,y,z,1)
    - HC ของเวกเตอร์ (x,y,z) คือ (x,y,z,0)

สิ่งที่คุณสามารถทำได้กับเมตริกซ์

```
Float4x4 A(2,0,0,1,0,2,0,1,0,0,2,1,0,0,0,1);
// A is "scale by factor of 2" then "translate by
   (1,1,1)."
Float3   v(1,1,1);
Float3   p(1,1,1);
Float3   u = A.multiply_vector(v);   // u = (2,2,2)
Float3   q = A.multiply_point(p);    // v = (3,3,3)
```

สิ่งที่คุณทำได้กับเมตริกซ์

- คำนวณ transpose ของมัน
- คำนวณ inverse ของมัน
- คำนวณ determinant ของมัน

สิ่งที่คุณทำได้กับเมตริกซ์

```
Float4x4 A(2,0,0,2,0,2,0,2,0,0,2,2,0,0,0,1);
// A is "scale by factor of 2" then "translate by
   (2,2,2)."
Float4x4 B = transpose(A);
// B = [[2,0,0,0],[0,2,0,0],[0,0,2,0],[2,2,2,1]]
Float4x4 C = inverse(A);
// C = [[0.5,0,0,1],[0,0.5,0,1],[0,0,0.5,1],[0,0,0,1]]
float d = det(A)
// d = 8
```

สิ่งที่คุณทำได้กับเมตริกซ์

- สร้าง matrix ของการแปลงที่สำคัญๆ
  - การย่อขยาย
  - การหมุน
  - การเลื่อนแกนขนาด
  - Look at transformation
  - Orthogonal projection
  - Perspective projection

สิ่งที่คุณทำได้กับเมตริกซ์

- ฟังก์ชันสำหรับสร้างการแปลงเหล่านี้เขียนเป็น static method เอาไว้
  - `static Float4x4 identity();`
    - สร้างเมตริกซ์เอกลักษณ์
  - `static Float4x4 translate(float x, float y, float z);`
    - สร้างเมตริกซ์ของการเลื่อนแกนขนาน
  - `static Float4x4 scale(float x, float y, float z);`
    - สร้างเมตริกซ์ของการย่อขยาย
  - `static Float4x4 rotate(float degrees, Vector3 axis);`
    - สร้างเมตริกซ์ของการหมุนในสามมิติ
  - `static Float4x4 look_at(`
    `Float3 eye, Float3 at, Float3 up);`
    - สร้างเมตริกซ์ของการแปลง look-at

สิ่งที่คุณทำได้กับเมตริกซ์

- ตัวอย่าง

```
Float4x4 I = Float4x4::identity();
Float4x4 T = Float4x4::translation(1,2,3);
Float4x4 R = Float4x4::rotation(60,
  Float3(0,0,1));
Float4x4 S = Float4x4::scale(2,2,2);
```

การแปลง

- แทนการแปลงแบบ affine เท่านั้น
  - ไม่รวม perspetive projection ซึ่งไม่ใช่การแปลง affine
- เราสามารถแทนการแปลงด้วยเมตริกซ์
- แต่เราต้องการให้คลาสของการแปลงสามารถทำงานได้มากกว่าเมตริกซ์เฉยๆ
  - เราต้องการคำนวณการแปลงผันกลับได้อย่างรวดเร็ว
  - เราต้องการคำนวณ inverse transpose ของการเมตริกซ์ของการแปลงได้อย่างรวดเร็วด้วย
- ฉะนั้นสำหรับการแปลงหนึ่งๆ เราจะเก็บ
  - เมตริกซ์ของมัน และ
  - เมตริกซ์ของ inverse ของมัน และ
  - inverse transpose ของเมตริกซ์ของมัน

การแปลง

```
struct Xform
{
public:
    Float4x4 m;
    Float4x4 mi;
    Float4x4 mit;

    // Other parts of the class omitted.
};
```

สิ่งที่คุณสามารถทำได้กับการแปลง

- สร้างมันขึ้นมา
  - มี constructor ให้ใช้หลายแบบ
    - `Xform();`
      - สร้างการแปลงเอกลักษณ์
    - `Xform(const Float4x4 &_m);`
      - กำหนดเมตริกซ์ให้
    - ```
      Xform(const Float4x4 &_m,
            const Float4x4 &_mi);
      ```
      - กำหนดเมตริกซ์และ inverse ของมันให้
    - ```
      Xform(const Float4x4 &_m,
            const Float4x4 &_mi,
            const Float4x4 &_mit);
      ```
      - กำหนดเมตริกซ์, inverse ของมัน, และ inverse transpose ของมันให้

สิ่งที่คุณสามารถทำได้กับการแปลง

- ตัวอย่าง

```
Xform T1 = Xform();
Xform T2 = Xform(Float4x4::translate(1,2,3));
Xform T3 = Xform(Float4x4::translate(1,2,3),
                 Float4x4::translate(-1,-2,-3));
Xform T4 = Xform(Float4x4::translate(1,2,3),
                 Float4x4::translate(-1,-2,-3),
                 transpose(Float4x4::translate(-1,-2,-3)));
```

สิ่งที่คุณสามารถทำได้กับการแปลง

- คูณมันเข้าด้วยกัน
  - พูดอีกอย่างหนึ่งคือทำการ compose มัน
  - ถ้า A และ B เป็นการแปลงแล้ว A*B คือการแปลงที่ทำ B ก่อนแล้วค่อยทำ A
- ตัวอย่าง

```
Xform A = Xform(Float4x4::translate(1,2,3));
Xform B = Xform(Float4x4::scale(2,2,2));
Xform C = A * B;
// C = "scale by factor of 2" then "translate by
  (1,2,3)"
```

สิ่งที่คุณทำได้กับการแปลง

- สร้างการแปลง affine พื้นฐานต่างๆ
  - `static Xform identity();`
  - `static Xform translate(`
    `float x, float y, float z);`
  - `static Xform scale(`
    `float x, float y, float z);`
  - `static Xform rotate(`
    `float degrees, Vector3 axis);`

Float2

```
struct Float2
{
  union
  {
    float d[2];
    struct { float x, y; };
    struct { float s, t; };
    struct { float u, v; };
  };

  // Other parts of the class omitted.
};
```

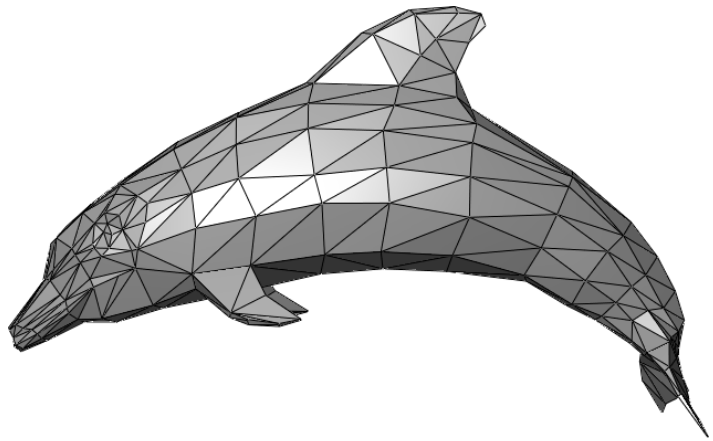
คุณสามารถทำอะไรได้กับ Float2

- สร้างมันขึ้นมา
  - มี constructor ให้ใช้อยู่หลายแบบ
    - `Float2();`
      - กำหนดค่าเริ่มต้น u = v = 0
    - `Float2(float c);`
      - กำหนดค่าเริ่มต้น u = v = c
    - `Float2(float _u, float _v);`
      - กำหนดค่าเริ่มต้น u = _u, v = _v
  - ตัวอย่าง

```
Float2 tc1 = Float2();          // tc1 = (0,0)
Float2 tc2 = Float2(0.5);       // tc2 = (0.5,0.5)
Float2 tc3 = Float2(0.1,0.2);   // tc3 = (0.1,0.2)
```

ข้อมูลรูปทรง

- รูปแบบการแทนรูปทรงที่ได้รับความนิยมมากที่สุด คือ polygon mesh (ตาข่ายของรูปหลายเหลี่ยม)
- Mesh ประกอบด้วย
  - จุดจำนวนหลายๆ จุด
  - เวกเตอร์ตั้งฉาก (normal) จำนวนหลายๆ เวกเตอร์
  - Texture coordinate หลายๆ ตัว
  - รูปหลายเหลี่ยม (polygon) ที่สร้างจากจุดข้างต้น แต่ละจุดมีข้อมูล normal และ texture coordinate กำกับอยู่ด้วย
  - เพื่อความง่าย เราจะเก็บเฉพาะรูปสามเหลี่ยมเท่านั้น



Mesh

```
class Mesh
{
public:
    Mesh();
    virtual ~Mesh();

public:
    std::vector<Float3> positions;
    std::vector<Float3> normals;
    std::vector<Float2> tex_coords;
    std::vector<MeshTriangle> triangles;
};
```

MeshTriangle

- เก็บสามเหลี่ยมใน mesh อันหนึ่ง
- ประกอบด้วย
  - int p[3]
    - อะเรย์สำหรับเก็บเลขที่ของตำแหน่งในสามมิติของ vertex ทั้งสาม
  - int n[3]
    - อะเรย์สำหรับเก็บเลขที่ของ normal ของ vertex ทั้งสาม
  - int t[3]
    - อะเรย์สำหรับเก็บเลขที่ของ texture coordinate ของ vertex ทั้งสาม

ไฟล์ .obj

- รูปแบบไฟล์ .obj เป็นรูปแบบไฟล์ที่ใช้แทน mesh ที่ใช้กันค่อนข้างแพร่หลาย
  - รูปแบบง่าย
  - เป็น plain text
- โปรแกรมทางด้านคอมพิวเตอร์กราฟิกส์ส่วนใหญ่ support ไฟล์ format นี้
  - 3DSMax, Maya, Blender, SoftImage XSI, ฯลฯ

ตัวอย่างไฟล์ .obj

```
# cube.obj
#

g cube

v  0.0  0.0  0.0
v  0.0  0.0  1.0
v  0.0  1.0  0.0
v  0.0  1.0  1.0
v  1.0  0.0  0.0
v  1.0  0.0  1.0
v  1.0  1.0  0.0
v  1.0  1.0  1.0

vn  0.0  0.0  1.0
vn  0.0  0.0 -1.0
vn  0.0  1.0  0.0
vn  0.0 -1.0  0.0
vn  1.0  0.0  0.0
vn -1.0  0.0  0.0

f  1//2  7//2  5//2
f  1//2  3//2  7//2
f  1//6  4//6  3//6
f  1//6  2//6  4//6
f  3//3  8//3  7//3
f  3//3  4//3  8//3
f  5//5  7//5  8//5
f  5//5  8//5  6//5
f  1//4  5//4  6//4
f  1//4  6//4  2//4
f  2//1  6//1  8//1
f  2//1  8//1  4//1
```

คำสั่งในไฟล์ .obj

- v x y z w
  - กำหนดตำแหน่งของจุดจุดหนึ่ง
  - จุดแรกที่กำหนดมีหมายเลข 1 จุดต่อไปมีหมายเลข 2 เช่นนี้ไปเรื่อยๆ
- vn i j k
  - กำหนด normal
  - normal แรกที่กำหนดมีหมายเลข 2 normal ตัวต่อไปมีหมายเลข 2 เช่นนี้ไปเรื่อยๆ

คำสั่งในไฟล์ .obj

- f v/vt/vn v/vt/vn v/vt/vn v/vt/vn
  - กำหนดหน้า
  - v/vt/vn จะมีอยู่กี่ตัวก็ได้ ขึ้นอยู่กับจำนวนมุม
  - v คือดรรชนีของตำแหน่ง (เริ่มจาก 1)
  - vt คือดรรชนีของ texture coordinate (เริ่มจาก 1)
    - แต่เราไม่สนใจตัวนี้
  - vn คือดรรชนีของ normal (เริ่มจาก 1)
  - ตัวอย่าง
    - f 1/1/1 2/2/2 3/3/3 4/4/4
  - สำหรับ mesh บาง mesh อาจไม่มีข้อมูล texture coordinate กรณีนี้เราสามารถเว้น vt ได้
    - f 1//1 2//2 3//3 4//4

ดู Demo

- การอ่านไฟล์ .obj