418341: สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์
การบรรยายครั้งที่ 16

ประมุข ขันเงิน

pramook@gmail.com

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
- คุณสามารถดู documentation (ที่ยังไม่สมบูรณ์) ได้ที่ directory: doc/html

ข้อมูลพื้นฐาน

- เวกเตอร์ → Vector3
- จุด → Point3
- เมตริกซ์ → Matrix4x4
- การแปลง → Transform
- Texture Coordinate → Uv
- สี → Rgb และ Rgba

เวกเตอร์

```
struct Vector3
{
public:
    float x;
    float y;
    float z;

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

สิ่งที่คุณสามารถทำได้กับเวกเตอร์

- สร้างมันขึ้นมา
  - มี constructor เพื่ออำนวยความสะดวกหลายแบบ
    - `Vector3();`
      - สร้างให้ component ทุกตัวมีค่าเท่ากับ 0
    - `Vector3(float c);`
      - สร้างให้ component ทุกตัวมีค่าเท่ากับ c
    - `Vector3(float _x, float _y, float _z);`
      - กำหนดค่าให้ component ทุกตัว
- ตัวอย่าง

```
Vector3 a();          // a = (0,0,0)
Vector3 b(1);         // b = (1,1,1)
Vector3 c(1,2,3);     // c = (1,2,3)
```

สิ่งที่คุณสามารถทำได้กับเวกเตอร์

- บวกเวกเตอร์
  - โค้ด

```
Vector3 operator+(const Vector3 &v) const
{
    return Vector3(x + v.x, y + v.y, z + v.z);
}
```

  - ตัวอย่าง

```
Vector3 a(1);        // a = (1,1,1)
Vector3 b(1,2,3);    // b = (1,2,3)
Vector3 c = a+b;     // c = (2,3,4)
```

สิ่งที่คุณสามารถทำได้กับเวกเตอร์

- ลบเวกเตอร์
  - โค้ด

```
Vector3 operator-(const Vector3 &v) const
{
    return Vector3(x - v.x, y - v.y, z - v.z);
}
```

  - ตัวอย่าง

```
Vector3 a(1);       // a = (1,1,1)
Vector3 b(1,2,3);   // b = (1,2,3)
Vector3 c = a-b;    // c = (0,-1,-2)
```

สิ่งที่คุณสามารถทำได้กับเวกเตอร์

- คูณเวกเตอร์ด้วยสเกลาร์
  - โค้ด

```
Vector3 operator*(float f) const
{
   return Vector3(f*x, f*y, f*z);
}
```

  - ตัวอย่าง

```
Vector3 a(1,0,2);    // a = (1,0,2)
Vector3 b = a*1.5f; // b = (1.5,0,3)
```

สิ่งที่คุณสามารถทำได้กับเวกเตอร์

- หาความยาว
  - โค้ด

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

  - ตัวอย่าง

```
Vector3 a(3,4,0);        // a = (3,4,0)
float l = a.length();    // l = 5
```

สิ่งที่คุณสามารถทำได้กับเวกเตอร์

- คำนวณ dot product
  - โค้ด

```
float dot(const Vector3 &v1, const Vector3 &v2)
{
 return v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
}
```

  - ตัวอย่าง

```
Vector3 a(3,4,0);          // a = (3,4,0)
Vector3 b(1,-1,5);         // b = (1,-1,5)
float d = dot(a,b);        // d = -1
```

สิ่งที่คุณสามารถทำได้กับเวกเตอร์

- คำนวณ cross product
  - โค้ด

```
inline Vector3 cross(
    const Vector3 &v1, const Vector3 &v2)
{
    return Vector3(
        (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

```
Vector3 a(1,0,0);        // a = (1,0,0)
Vector3 b(0,1,0);        // b = (0,1,0)
Vector3 c = cross(a,b) // c = (0,0,1)
```

จุด

```
struct Point3
{
public:
    float x;
    float y;
    float z;

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

ข้อแตกต่างระหว่างจุดกับเวกเตอร์

- คุณไม่สามารถ
  - บวกจุดสองจุดเข้าด้วยกันได้
  - คูณจุดด้วยสเกลาร์ได้
  - หาความยาวของจุดได้
- แต่คุณสามารถ
  - ลบจุดสองจุดได้ (ผลลัพธ์ออกมาเป็นเวกเตอร์)
  - บวกจุดกับเวกเตอร์ได้ (ผลลัพธ์ออกมาเป็นจุด)
  - หาระยะห่างระหว่างจุดสองจุดได้

สิ่งที่คุณสามารถทำกับจุดได้

- สร้างมันขึ้นมา
  - มี constructor เพื่ออำนวยความสะดวกหลายแบบ
    - `Point3();`
      - สร้างให้ component ทุกตัวมีค่าเท่ากับ 0
    - `Point3(float c);`
      - สร้างให้ component ทุกตัวมีค่าเท่ากับ c
    - `Point3(float _x, float _y, float _z);`
      - กำหนดค่าให้ component ทุกตัว
- ตัวอย่าง

```
Point3 a();          // a = (0,0,0)
Point3 b(1);         // b = (1,1,1)
Point3 c(1,2,3);     // c = (1,2,3)
```

สิ่งที่คุณสามารถทำกับจุดได้

- ลบจุดหนึ่งออกจากอีกจุดหนึ่ง
  - ได้ผลเป็นเวกเตอร์จากตัวลบไปยังตัวตั้ง
  - โค้ด

```
Vector3 operator-(const Point3 &p) const
{
    return Vector3(x - p.x, y - p.y, z - p.z);
}
```

  - ตัวอย่าง

```
Point3  a(1,2,3); // a = (1,2,3)
Point3  b(4,5,6); // b = (4,5,6)
Vector3 c = b-a;  // c = (3,3,3)
```

สิ่งที่คุณสามารถทำกับจุดได้

- บวกจุดกับเวกเตอร์
  - ได้ผลเป็นจุดที่มีค่าเท่ากับการเลื่อนจุดตัวตั้งไปตามเวกเตอร์ที่ให้
  - โค้ด

```
Point3 operator+(const Vector3 &v) const
{
    return Point3(x + v.x, y + v.y, z + v.z);
}
```

  - ตัวอย่าง

```
Point3  a(1,2,3); // a = (1,2,3)
Vector3 v(1,1,1); // v = (1,1,1)
Point3  b = a+v   // b = (2,3,4)
```

สิ่งที่คุณสามารถทำกับจุดได้

- หาระยะห่างระหว่างจุด
  - โค้ด

```
float distance(const Point3 &p1, const Point3 &p2)
{
 return (p1 - p2).length();
}
```

  - ตัวอย่าง

```
Point3 a(1,2,3);           // a = (1,2,3)
Point3 b(1,2,4);           // b = (1,2,4)
float d = distance(a,b); // d = 1
```

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 Matrix4x4
{
private:
    float m[4][4];

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

สิ่งที่คุณสามารถทำได้กับเมตริกซ์

- สร้างมันขึ้นมา
  - มี constructor อำนวยความสะดวกหลายแบบ
- ตัวอย่าง

```
Matrix4x4 A;      // All elements are 0.
Matrix4x4 B(1);   // All elements are 1.
Matrix4x4 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);
```

สิ่งที่คุณสามารถทำได้กับเมตริกซ์

- บวกเมตริกซ์
- ลบเมตริกซ์
- คูณเมตริกซ์ด้วยสเกลาร์
- คูณเมตริกซ์กับเมตริกซ์

สิ่งที่คุณทำได้กับเมตริกซ์

- ตัวอย่าง

```
Matrix4x4 A(2,0,0,0,0,2,0,0,0,0,2,0,0,0,0,2);
Matrix4x4 B(0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0);
Matrix4x4 C = A+B;
// C = [[2,1,0,0],[0,2,1,0],[0,0,2,1],[1,0,0,2]]
Matrix4x4 D = A-B;
// D = [[2,-1,0,0],[0,2,-1,0],[0,0,2,-1],[-1,0,0,2]]
Matrix4x4 E = A*2;
// E = [[4,0,0,0],[0,4,0,0],[0,0,4,0],[0,0,0,4]]
Matrix4x4 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

```
Matrix4x4 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)."
Vector3   v(1,1,1);
Point3    p(1,1,1);
Vector3   u = A*v;   // u = (2,2,2)
Point3    q = A*p;   // v = (3,3,3)
```

สิ่งที่คุณทำได้กับเมตริกซ์

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

สิ่งที่คุณทำได้กับเมตริกซ์

```
Matrix4x4 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)."
Matrix4x4 B = transpose(A);
// B = [[2,0,0,0],[0,2,0,0],[0,0,2,0],[2,2,2,1]]
Matrix4x4 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 Matrix4x4 identity();`
    - สร้างเมตริกซ์เอกลักษณ์
  - `static Matrix4x4 translate(float x, float y, float z);`
    - สร้างเมตริกซ์ของการเลื่อนแกนขนาน
  - `static Matrix4x4 scale(float x, float y, float z);`
    - สร้างเมตริกซ์ของการย่อขยาย
  - `static Matrix4x4 rotate(float degrees, Vector3 axis);`
    - สร้างเมตริกซ์ของการหมุนในสามมิติ

สิ่งที่คุณทำได้กับเมตริกซ์

- `static Matrix4x4 look_at(`
  `Point3 eye, Point3 at, Vector3 up);`
  - สร้างเมตริกซ์ของการแปลง look-at
- `static Matrix4x4 orthographic_projection(`
  `float l, float r, float b, float t,`
  `float n=0, float f=1);`
  - สร้างเมตริกซ์ของ orthographic projection
- `static Matrix4x4 perspective_projection(`
  `float l, float r, float b,`
  `float t, float n, float f);`

  `static Matrix4x4 perspective_projection(`
  `float fovy, float aspect,`
  `float near, float far);`
  - สร้างการแปลงของ perspective projection

สิ่งที่คุณทำได้กับเมตริกซ์

- ตัวอย่าง

```
Matrix4x4 I = Matrix4x4::identity();
Matrix4x4 T = Matrix4x4::translation(1,2,3);
Matrix4x4 R = Matrix4x4::rotation(60,
  Vector3(0,0,1));
Matrix4x4 S = Matrix4xx::scale(2,2,2);
```

การบ้านครั้งต่อไป (1)

- ให้เขียนฟังก์ชัน static สำหรับสร้างเมตริกซ์เหล่านี้

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 Transform
{
public:
    Matrix4x4 m;
    Matrix4x4 mi;
    Matrix4x4 mit;

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

สิ่งที่คุณสามารถทำได้กับการแปลง

- สร้างมันขึ้นมา
  - มี constructor ให้ใช้หลายแบบ
    - `Transform();`
      - สร้างการแปลงเอกลักษณ์
    - `Transform(const Matrix4x4 &_m);`
      - กำหนดเมตริกซ์ให้
    - `Transform(const Matrix4x4 &_m,`
      `const Matrix4x4 &_mi);`
      - กำหนดเมตริกซ์และ inverse ของมันให้
    - `Transform(const Matrix4x4 &_m,`
      `const Matrix4x4 &_mi,`
      `const Matrix4x4 &_mit);`
      - กำหนดเมตริกซ์, inverse ของมัน, และ inverse transpose ของมันให้

สิ่งที่คุณสามารถทำได้กับการแปลง

- ตัวอย่าง

```
Transform T1 = Transform();
Transform T2 = Transform(Matrix4x4::translate(1,2,3));
Transform T3 = Transform(Matrix4x4::translate(1,2,3),
                         Matrix4x4::translate(-1,-2,-3));
Transform T4 = Transform(Matrix4x4::translate(1,2,3),
                   Matrix4x4::translate(-1,-2,-3),
                   transpose(Matrix4x4::translate(-1,-2,-3)));
```

สิ่งที่คุณสามารถทำได้กับการแปลง

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

```
Transform A = Transform(Matrix4x4::translate(1,2,3));
Transform B = Transform(Matrix4x4::scale(2,2,2));
Transform C = A * B;
// C = "scale by factor of 2" then "translate by
  (1,2,3)"
```

สิ่งที่คุณทำได้กับการแปลง

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

การบ้านครั้งต่อไป (2)

- เขียนฟังก์ชันเพื่อ
  - คูณ transform สองตัวเข้าด้วยกัน
  - สร้าง affine transform พื้นฐานต่างๆ

Texture Coordinate

```
struct Uv
{
public:
    float u;
    float v;

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

คุณสามารถทำอะไรได้กับ Texture Coordinate

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

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

สี

- มีสองแบบคือ
  - Rgb
  - Rgba
- เพื่อให้สไลด์สั้น เราจะกล่าวถึงเฉพาะสีแบบ Rgba เท่านั้น

สี Rgba

```
struct Rgba
{
public:
    float r;
    float g;
    float b;
    float a;

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

คุณสามารถทำอะไรกับสี Rgba ได้บ้าง

- สร้างมันขึ้นมา
  - มี constructor ให้ใช้หลายแบบ
    - Rgba()
      - กำหนดค่าเริ่มต้น r = g = b = a = 0
    - Rgba(float c)
      - กำหนดค่าเริ่มต้น r = g = b = a = c
    - Rgba(float _r, float _g, float _b)
      - กำหนดค่า r, g, b ตาม argument ที่ให้มาและกำหนด a = 1
    - Rgba(float _r, float _g, float _b, float _a)
      - กำหนดค่า r, g, b, และ a ตาม argument ที่ให้มา

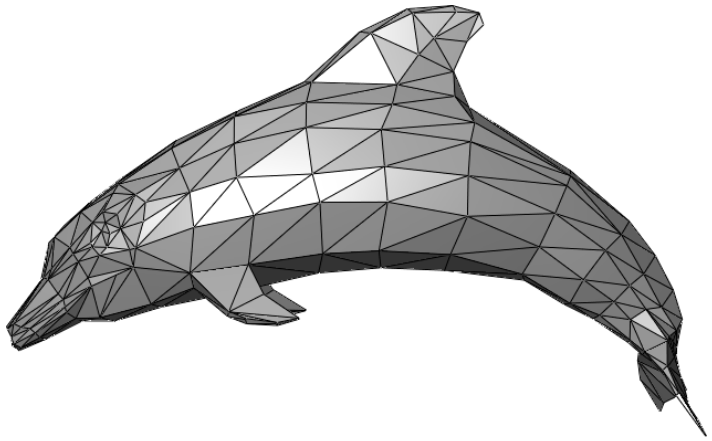
คุณสามารถทำอะไรกับสี Rgba ได้บ้าง

- เอามันมา
  - บวกกัน
  - ลบกัน
  - คูณกัน
- คูณ Rgba ด้วยสเกลาร์
- ตัวอย่าง

```
Rgba a(0.1, 0.2, 0.3, 0.4);
Rgba b(0.8, 0.7, 0.6, 0.5);
Rgba c = a + b; // c = (0.9, 0.9, 0.9, 0.9)
Rgba d = a - b; // d = (-0.7, -0.5, -0.3, -0.1)
Rgba e = a * b; // e = (0.08, 0.14, 0.18, 0.2)
Rgba f = a * 2; // f = (0.2, 0.4, 0.6, 0.8)
```

ข้อมูลรูปทรง

- รูปแบบการแทนรูปทรงที่ได้รับความนิยมมากที่สุด คือ polygon mesh (ตาข่ายของรูปหลายเหลี่ยม)
- Mesh ประกอบด้วย
  - จุดจำนวนหลายๆ จุด
  - เวกเตอร์ตั้งฉาก (normal) จำนวนหลายๆ เวกเตอร์
  - รูปหลายเหลี่ยม (polygon) ที่สร้างจากจุดข้างต้น แต่ละจุดมีข้อมูล normal กำกับอยู่ด้วย
  - ส่วนมากเราจะเรียกรูปหลายเหลี่ยมเหล่านี้ว่า “หน้า” (face)



ข้อมูลรูปทรง

- หน้า (face)
  - รูปหลายเหลี่ยมหนึ่งรูป
  - ต้องรู้ว่ามันมีมุมกี่มุม
  - ข้อตกลง: เก็บจุดยอดมุมที่เมื่อมองจากทางด้านนอกรูปหลายเหลี่ยมแล้วจะเรียงกันตามแนวทวนเข็มนาฬิกา
  - สำหรับจุดยอดมุมแต่ละจุดจะมี
    - เลขจำนวนเต็มตัวหนึ่งบอกว่าจุดยอดมุมนี้ตรงกับจุดไหน
    - เลขจำนวนเต็มตัวหนึ่งบอกว่าเวกเตอร์ตั้งฉากของจุดยอดมุมนี้

ข้อมูลรูปทรง

- เราจะสร้าง interface ชื่อ Mesh สำหรับเก็บข้อมูล mesh
- Interface นี้มีฟังก์ชันต่างๆ ดังต่อไปนี้
  - `int get_position_count() const`
    - หาว่ามีจุดกี่จุด
  - `int get_normal_count() const`
    - หาว่ามีเวกเตอร์ตั้งฉากกี่เวกเตอร์
  - `Point3 get_position(int position_index) const`
    - อ่านค่าจุดที่กำหนดด้วยดรรชนี position_index
  - `Vector3 get_normal(int normal_index) const`
    - อ่านค่าจุดที่กำหนดด้วยดรรชนี normal_index

ข้อมูลรูปทรง

- `int get_face_count() const`
  - หาว่ามีหน้าอยู่ทั้งหมดกี่หน้า
- `int get_face_vertex_count(int face_index) const`
  - หาว่าหน้าที่กำหนดด้วยดรรชนี face_index มีมุมกี่มุม
- `int get_face_vertex_position_index(`
  `int face_index, int vertex_index) const`
  - หาวดรรชนีของตำแหน่งของจุดมุมที่กำหนดด้วยดรรชนี vertex_index ของหน้าที่ถูกกำหนดด้วยดรรชนี face_index
- `int get_face_vertex_normal_index(`
  `int face_index, int vertex_index) const`
  - หาดรรชนีของ normal ของจุดมุมที่กำหนดด้วยดรรชนี vertex_index ของหน้าที่ถูกกำหนดด้วยดรรชนี face_index

ข้อมูลรูปทรง

- `Point3 get_face_vertex_position(`
  `int face_index, int vertex_index) const`
  - หาตำแหน่งของจุดมุมที่กำหนดด้วยดรรชนี vertex_index ของหน้าที่ถูกกำหนดด้วยดรรชนี face_index
- `Vector3 get_face_vertex_normal(`
  `int face_index, int vertex_index) const`
  - หา normal ของจุดมุมที่กำหนดด้วยดรรชนี vertex_index ของหน้าที่ถูกกำหนดด้วยดรรชนี face_index

ข้อมูลรูปทรง

- สังเกตว่า interface Mesh นี้มีเฉพาะเมธอดสำหรับอ่านข้อมูลที่เก็บอยู่ใน mesh เท่านั้น
  - นำไปใช้ในการแสดงผลได้อย่างเดียว
- เราจะสร้าง class ซึ่งเป็นลูกหลานของ Mesh ที่มีเมธอดสำหรับการเพิ่มข้อมูลลงไปด้วย
- ในที่นี้เราจะให้ชื่อ class นั้นว่า UniversalMesh (เนื่องจากมันแทน mesh ได้ทุกแบบ)

ข้อมูลรูปทรง

```
class UniversalMesh : public Mesh
{
public:
    virtual void append_position(const Point3 &position);
    virtual void append_normal(const Vector3 &normal);
    virtual void append_new_face();
    virtual void append_vertex_to_last_face(
        int position_index, int normal_index);

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

ข้อมูลรูปทรง

- `virtual void append_position(const Point3 &position);`
  - เพิ่มตำแหน่งต่อท้าย list ของตำแหน่งที่มีทั้งหมด
- `virtual void append_normal(const Vector3 &normal);`
  - เพิ่มเวกเตอร์ต่อท้าย list ของเวกเตอร์ที่มีทั้งหมด
- `virtual void append_new_face();`
  - เพิ่มหน้าใหม่ (เท่ากับเป็นการปิดหน้าเดิม)
- `virtual void append_vertex_to_last_face(int position_index, int normal_index);`
  - เพิ่มข้อมูลของจุดยอดมุมเข้าสู่หน้าสุดท้ายในตอนนี้
  - ข้อมูลของจุดยอดมุมคือดรรชนีของตำแหน่งและดรรชนีของ normal

ข้อมูลรูปทรง

- ตัวอย่างการใช้ UniversalMesh
  - เราจะสร้างลูกบาศก์ขนาด 2x2x2 ที่จุดศูนย์กลางอยู่ที่จุด (0,0,0)
  - อันดับแรกให้ประกาศมันขึ้นมาก่อน

```
UniversalMesh cube;
```

ข้อมูลรูปทรง

– หลังจากนั้นจึงใส่จุด

```
cube.append_position(Point3(-1,-1,-1));
cube.append_position(Point3(-1,-1, 1));
cube.append_position(Point3(-1, 1,-1));
cube.append_position(Point3(-1, 1, 1));
cube.append_position(Point3( 1,-1,-1));
cube.append_position(Point3( 1,-1, 1));
cube.append_position(Point3( 1, 1,-1));
cube.append_position(Point3( 1, 1, 1));
```

ข้อมูลรูปทรง

– แล้วจึงใส่ normal

```
cube.append_normal(Vector3( 0, 0, 1));
cube.append_normal(Vector3( 0, 0,-1));
cube.append_normal(Vector3( 1, 0, 0));
cube.append_normal(Vector3(-1, 0, 0));
cube.append_normal(Vector3( 0, 1, 0));
cube.append_normal(Vector3( 0,-1, 0));
```

ข้อมูล

- หลังจากนั้นจึงเพิ่มหน้าใหม่ที่ละหน้า แล้วใส่ข้อมูลจุดยอดมุมของหน้านั้นเข้าไปเรื่อยๆ

```
cube.append_new_face();
cube.append_vertex_to_last_face(1,0);
cube.append_vertex_to_last_face(5,0);
cube.append_vertex_to_last_face(7,0);
cube.append_vertex_to_last_face(3,0);
```

การบ้านครั้งต่อไป (3)

- ให้ implement UniversalMesh

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

การบ้านครั้งต่อไป (4)

- เขียน code เพื่ออ่านไฟล์ .obj