การเขียน Ray Tracer อย่างง่าย

ประมุข ขันเงิน

อัลกอริทึม

RGB Trace(Ray *r)*

{

Find closest point that r intersects an object.

return radiance from the intersection point

}

หาจุดตัดที่ใกล้สุด

- เราสามารถหาจุดตัดที่ใกล้ที่สุดได้หลายวิธี
- แต่ตอนนี้เราจะใช้วิธีโง่ๆ
- เอา ray ไปหาจุดตัดกับวัตถุทุกวัตถุแล้วหาจุดที่ใกล้ที่สุด

หาจุดตัดที่ใกล้สุด (ต่อ)

- เวลาเขียนโปรแกรม เราสามารถกำหนดให้ฟังก์ชันที่ทำ intersection test คืนค่า t ที่ทำให้

$$r.\mathbf{o} + t(r.\mathbf{d})$$

  เป็นจุดตัดของ ray กับวัตถุนั้น
- เราแค่เก็บค่า t ที่น้อยที่สุด และวัตถุเจ้าของค่า t นั้นเอาไว้

หาจุดตัดที่ใกล้สุด (ต่อ)

RGB TRACE(Ray r)

{

$(t, o) \leftarrow$ FIND-CLOSEST-INTERSECTION(r)

return radiance from the intersection point

}

หาจุดตัดที่ใกล้สุด (ต่อ)

```
FIND-CLOSEST-INTERSECTION(Ray r)
{
    t_closest ← ∞
    foreach object o in the scene {
        t ← o.INTERSECT(r)
        if t < t_closest {
            t_closest ← t
            o_closest ← o
        }
    }
    return (t_closest, o_closest)
}
```

หาข้อมูลเพิ่มเติมเกี่ยวกับจุดตัด

- เวลาจะให้สีเราต้องมีข้อมูลเกี่ยวกับจุดตัดเพิ่มเติม
- สำหรับ ray tracer ง่ายๆ ตัวนี้ เราจะหา
  - ตำแหน่งของจุดตัด $\mathbf{p}$
  - เวกเตอร์ตั้งฉากกับพื้นผิว $\mathbf{n}$
- ข้อมูลอื่นๆ ที่จำเป็น เช่น
  - texture coordinate
  - tangent vector

  แต่เรายังไม่สนใจมันตอนนี้

หาข้อมูลเพิ่มเติมเกี่ยวกับจุดตัด (ต่อ)

- เก็บข้อมูลเหล่านี้ไว้ในโครงสร้างข้อมูลประเภท DiffGeom (ย่อมาจาก Differential Geometry)

```
class DiffGeom
{
    Vector p
    Vector n
}
```

- เราจะใส่ข้อมูลอย่างอื่นเพิ่มในโครงสร้างข้อมูลนี้ในอนาคต

หาข้อมูลเพิ่มเติมเกี่ยวกับจุดตัด (ต่อ)

- นอกจากนี้ยังกำหนดให้วัตถุแต่ละตัวต้อง implement ฟังก์ชัน

$$\text{DiffGeom } \textsc{Get-Differential-Geometry}(\text{Ray } r, \text{ float } t)$$

- ฟังก์ชันนี้รับ ray และเวลาที่ ray นั้นตัดกับวัตถุ แล้วคืน DiffGeom ที่จุดตัดออกมา
- เพื่อความง่าย เราจะคิดว่าเวลาที่ให้มาเป็นเวลาที่ถูกต้องแล้ว (จะได้ไม่ต้องเช็คซ้ำอีก)

หาข้อมูลเพิ่มเติมเกี่ยวกับจุดตัด (ต่อ)

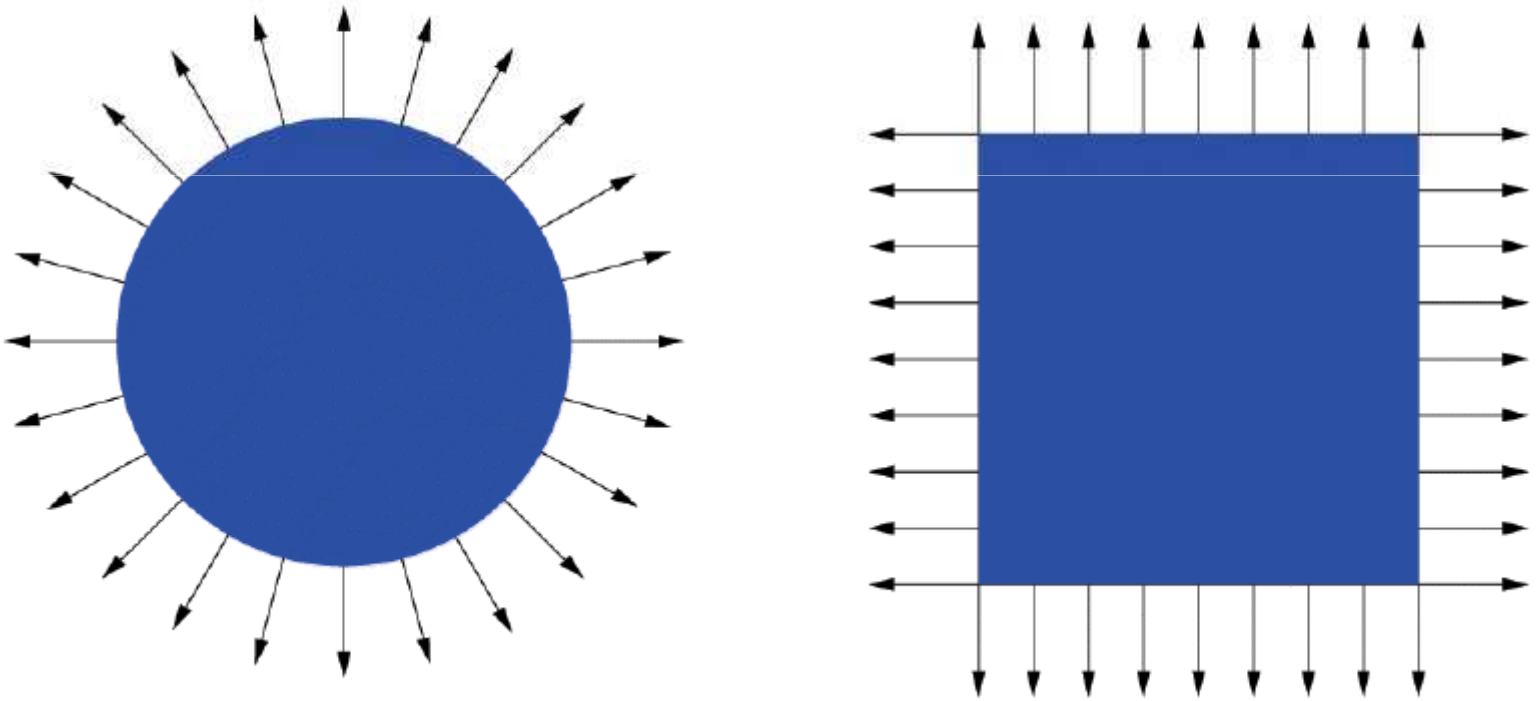
- ข้อสังเกต
  - ray ที่ให้มานั้นเป็น ray ใน world space
  - แต่วัตถุส่วนใหญ่ถูกนิยามใน object space
  - ค่า t เป็นเวลาทั้งใน world space และ object space
  - เราต้องการ output เป็นจุด $\mathbf{p}$ และเวกเตอร์ $\mathbf{n}$ ใน world space
- ดังนั้น จุดตัดหาง่าย

$$\mathbf{p} \leftarrow r.\mathbf{o} + t(r.\mathbf{d})$$

- แต่เวกเตอร์ตั้งฉากหายากกว่า

เวกเตอร์ตั้งฉาก

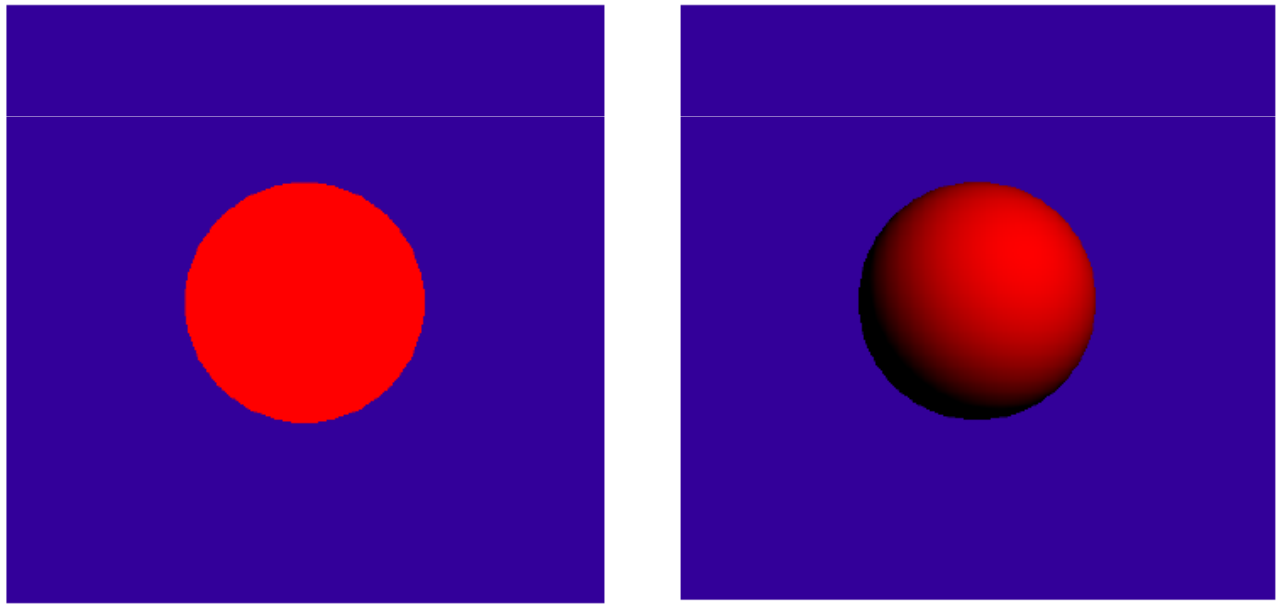
- เวกเตอร์ตั้งฉาก = เวกเตอร์ทั้งตั้งฉากกับผิววัตถุที่จุดตัด



รูปจาก Durand and Culter, *Transformation in Ray Tracing*. http://ocw.mit.edu

เวกเตอร์ตั้งฉาก (ต่อ)

- เวกเตอร์ตั้งฉากเป็นข้อมูลสำคัญในการให้สี
- ช่วยให้วัตถุดูเป็นสามมิติขึ้นมา



object color only Diffuse Shading

รูปจาก Durand and Culter, *Transformation in Ray Tracing*. http://ocw.mit.edu

เวกเตอร์ตั้งฉาก (ต่อ)

- นอกจากนี้ ยังเป็นตัวบอกเวลาเราอยู่ในหรือนอกวัตถุ
- เรากำหนดให้เวกเตอร์ตั้งฉากพุ่งออกนอกวัตถุเสมอ
- การรู้ด้านในด้านนอกเป็นประโยชน์เวลาจำลองการหักเหของแสง
- แค่เช็คมุมของทางเดินแสงกับเวกเตอร์ตั้งฉากก็จะรู้ได้ว่าแสงกำลังจะออกหรือเข้าไปในวัตถุ
- สังเกตว่าถ้ากลับทิศเวกเตอร์ตั้งฉาก เราจะกลับด้านในด้านนอกของวัตถุ

หาเวกเตอร์ตั้งฉาก

- เพื่อความง่าย เรามาคิดถึงการหาเวกเตอร์ตั้งฉากใน object space กันก่อน
- กล่าวคือ เราอาจกำหนดให้วัตถุแต่ละตัวมีฟังก์ชัน

  $$\text{Vector } \textsc{Get-Object-Space-Normal}(\text{Ray } r, \text{ float } t)$$

  ซึ่งคืนค่าเวกเตอร์ตั้งฉากใน object space

  โดยมีข้อแม้ว่า ray ที่ให้ต้องอยู่ใน object space เรียบร้อยแล้ว
- เพื่อความสะดวก เวกเตอร์ตั้งฉากจะเป็นเวกเตอร์หนึ่งหน่วยเสมอ

เวกเตอร์ตั้งฉากของสี่เหลี่ยม

- เรานิยามสี่เหลี่ยมว่าเป็นเซต $\{(x, y, 0) : -1 \leq x, y \leq 1\}$
- กล่าวคือมันขนานกับระนาบ xy
- มีเวกเตอร์หนึ่งหน่วยที่ตั้งฉากกับระนาบ xy สองตัว
  - (0,0,1)
  - (0,0,-1)
- เราเลือกให้ (0,0,1) เป็นเวกเตอร์ตั้งฉาก

เวกเตอร์ตั้งฉากของสี่เหลี่ยม (ต่อ)

Vector Square::GET-OBJECT-SPACE-NORMAL(Ray r, float t)
{
    return $(0, 0, 1)$
}

เวกเตอร์ตั้งฉากของสามเหลี่ยม

- เรานิยมนิยามสามเหลี่ยมโดยให้จุดมุมทั้งสามเรียงกัน**ทวนเข็มนาฬิกา**
- กล่าวคือ ถ้าเดินจากจุด $\mathbf{p}_1$ ไป $\mathbf{p}_2$ ไป $\mathbf{p}_3$ เราจะเดินวนซ้าย
- เวกเตอร์ตั้งฉากกับเวกเตอร์สามเหลี่ยมนิยามโดยใช้กฎมือขวา
  - เมื่อใช้มือขวาตวัดจาก $\mathbf{p}_1$ไป $\mathbf{p}_2$ไป $\mathbf{p}_3$ แล้วเวกเตอร์ตั้งฉากจะชี้ไปทางด้านที่นิ้วโป้งอยู่
- พูดเป็นภาษาคณิตศาสตร์คือ

$$\mathbf{n} = \textsc{Normalize}((\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1))$$

เวกเตอร์ตั้งฉากของสามเหลี่ยม (ต่อ)

Vector Triangle::GET-OBJECT-SPACE-NORMAL(Ray r, float t)
{
    return NORMALIZE($(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)$)
}

เวกเตอร์ตั้งฉากของทรงกลม

- ใน object space วงกลมของเราเป็นวงกลมหนึ่งหน่วย
- จุดตัดก็ต้องอยู่บนวงกลมหนึ่งหน่วย
- เรากำหนดให้เวกเตอร์ตั้งฉากพุ่งออกนอกวงกลม
- เวกเตอร์ตั้งฉากคือ**เวกเตอร์จากจุดศูนย์กลางไปหาจุดตัด**
- พูดง่ายๆ คือ มันมีค่าเท่ากับจุดตัดนั่นเอง (เพราะจุดศูนย์กลางมีพิกัด (0,0,0))

$$\mathbf{n} = r.\mathbf{o} + t(r.\mathbf{d})$$

เวกเตอร์ตั้งฉากของทรงกลม (ต่อ)

Vector Sphere::GET-OBJECT-SPACE-NORMAL(Ray r, float t)
{

return $r.\mathbf{o} + t(r.\mathbf{d})$

}

หาเวกเตอร์ตั้งฉากใน world space

- เมื่อได้เวกเตอร์ตั้งฉากใน object space มาแล้ว เราจะต้องแปลงให้มันอยู่ใน world space
- สมมติว่าการแปลง M เป็นการแปลงจาก object space ไป world space
- เราจะแปลงเวกเตอร์ตั้งฉากอย่างไร?

หาเวกเตอร์ตั้งฉากใน world space (ต่อ)

DiffGeom GET-DIFFERENTIAL-GEOMETRY(Ray r, float t)

{

$dg \leftarrow$ **new** DiffGeom()

$dg.\mathbf{p} \leftarrow r.\mathbf{o} + t(r.\mathbf{d})$

$r_{\text{object}} \leftarrow M^{-1}$.TRANSFORM-RAY($r$)

$\mathbf{n}_{\text{object}} \leftarrow$ GET-OBJECT-SPACE-NORMAL(r_{object}, t)

$dg.\mathbf{n} \leftarrow$???

return dg

}

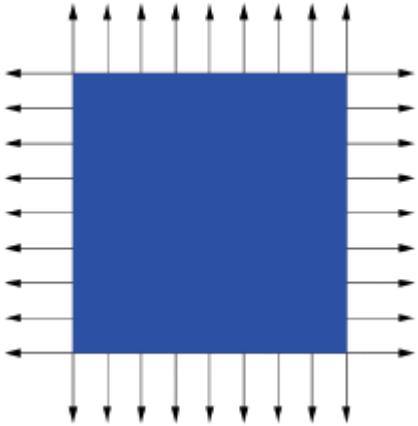
หาเวกเตอร์ตั้งฉากใน world space (ต่อ)

- ก็แปลงมันเหมือนเวกเตอร์ธรรมดาไม่ได้เหรอ?

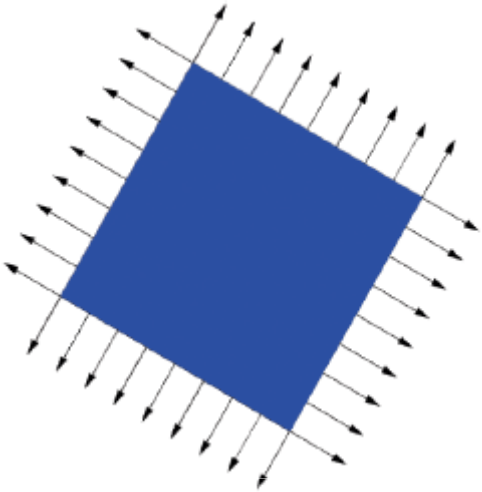
$$dg.\mathbf{n} \leftarrow M.\textsc{Transform-Direction}(\mathbf{n}_{\text{object}})$$

- มาดูกันสักหน่อย

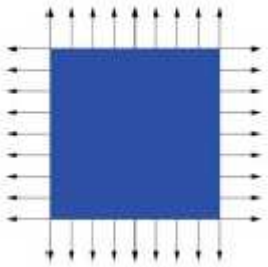
หาเวกเตอร์ตั้งฉากใน world space (ต่อ)



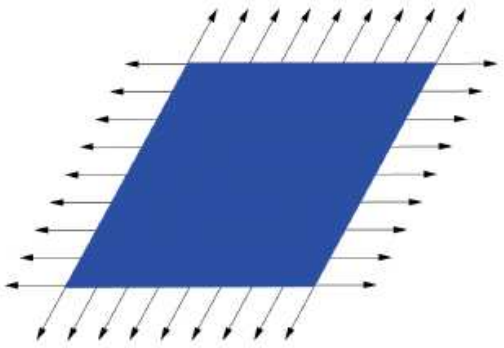
ต้นแบบใน object space



หมุน (ยังดีอยู่)



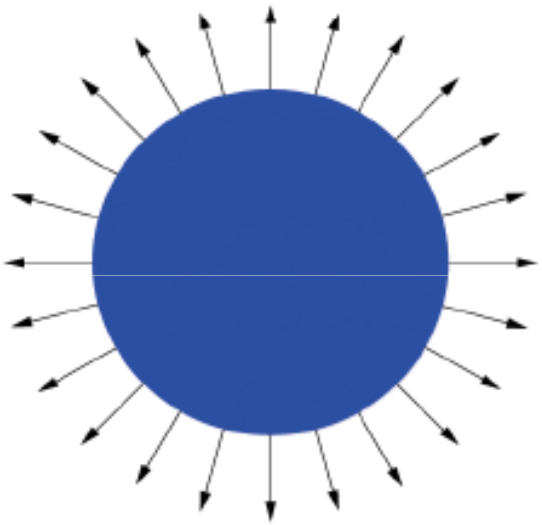
ย่อ/ขยาย (ยังดีอยู่)



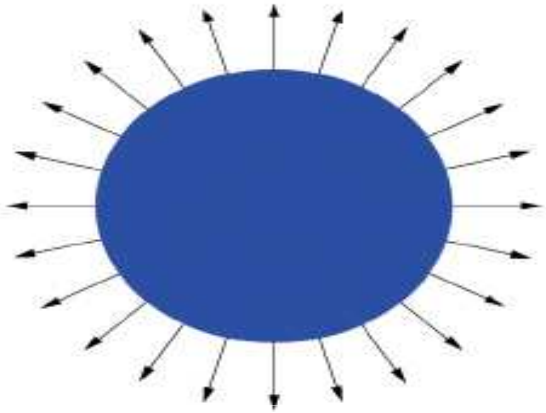
เบ้ (ผิดแล้ว)

รูปจาก Durand and Culter, *Transformation in Ray Tracing*. http://ocw.mit.edu

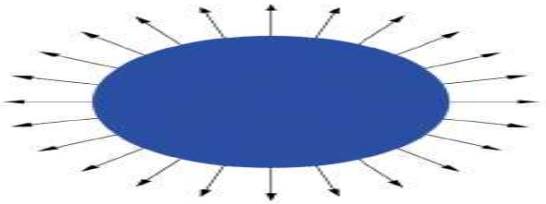
หาเวกเตอร์ตั้งฉากใน world space (ต่อ)



ต้นแบบใน object space



ย่อลงหน่อย (ผิดแล้ว)

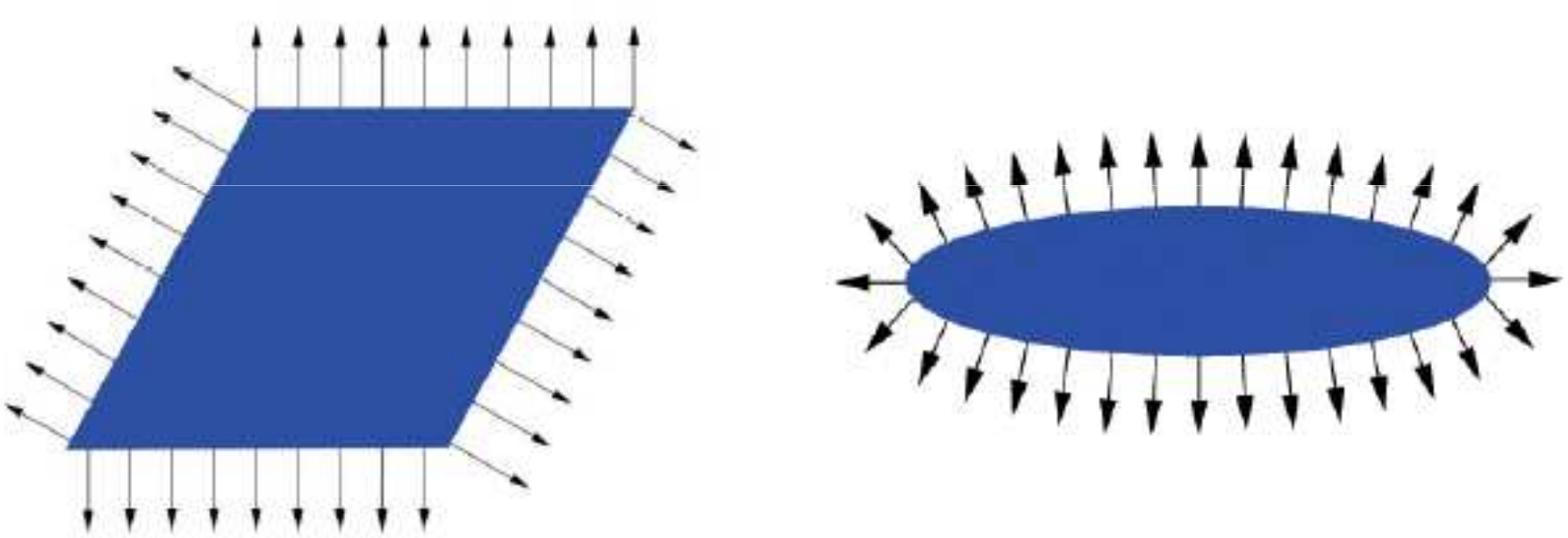


ย่อลงอีก (ผิดขึ้นไปอีก)

รูปจาก Durand and Culter, *Transformation in Ray Tracing*. http://ocw.mit.edu

หาเวกเตอร์ตั้งฉากใน world space (ต่อ)

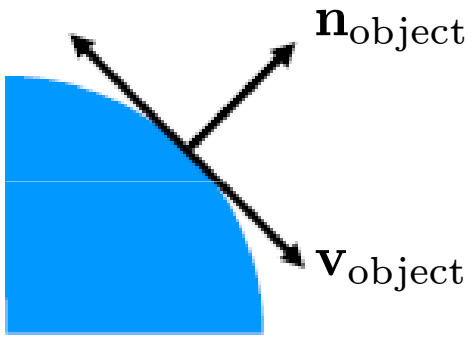
- สิ่งที่เราต้องการ



รูปจาก Durand and Culter, *Transformation in Ray Tracing*. http://ocw.mit.edu

หาเวกเตอร์ตั้งฉากใน world space (ต่อ)

- แทนที่จะคิดว่าจะแปลงเวกเตอร์ตั้งฉาก ลองคิดถึงการเวกเตอร์ในระนาบที่ขนานกับพื้นผิวที่จุดตัด



- ให้ $\mathbf{v}_{\text{object}}$ เป็นเวกเตอร์ที่อยู่ในระนาบขนานพื้นผิว

หาเวกเตอร์ตั้งฉากใน world space (ต่อ)

- เราได้ว่าเราสามารถแปลง $\mathbf{v}_{\text{object}}$ ให้อยู่ใน world space เหมือนกับการแปลงเวกเตอร์ทั่วไป

$$\mathbf{v}_{\text{world}} = M\mathbf{v}_{\text{object}}$$

หาเวกเตอร์ตั้งฉากใน world space (ต่อ)

- เนื่องจากเวกเตอร์ตั้งฉาก $\mathbf{n}_{\text{world}}$ จะต้องตั้งฉากกับ $\mathbf{v}_{\text{world}}$ เราได้ว่า

$$\mathbf{v}_{\text{world}} \cdot \mathbf{n}_{\text{world}} = \left(\mathbf{v}_{\text{world}}\right)^T \mathbf{n}_{\text{world}} = 0$$

แต่

$$\mathbf{v}_{\text{world}} = M\mathbf{v}_{\text{object}}$$

ฉะนั้น

$$\mathbf{v}_{\text{object}}^T M^T \mathbf{n}_{\text{world}} = 0$$

หาเวกเตอร์ตั้งฉากใน world space (ต่อ)

- ถ้าเราให้

$$\mathbf{n}_{\text{world}} = (M^{-1})^T \mathbf{n}_{\text{object}}$$

จะได้ว่า

$$\begin{aligned} \mathbf{v}_{\text{object}}^T M^T \mathbf{n}_{\text{world}} &= \mathbf{v}_{\text{world}}^T M^T (M^{-1})^T \mathbf{n}_{\text{object}} \\ &= \mathbf{v}_{\text{object}}^T \mathbf{n}_{\text{object}} = 0 \end{aligned}$$

สูตรข้างบนใช้ได้

หาเวกเตอร์ตั้งฉากใน world space (ต่อ)

DiffGeom GET-DIFFERENTIAL-GEOMETRY(Ray r, float t)

{

$dg \leftarrow$ **new** DiffGeom()

$dg.\mathbf{p} \leftarrow r.\mathbf{o} + t(r.\mathbf{d})$

$r_{\text{object}} \leftarrow M^{-1}$.TRANSFORM-RAY($r$)

$\mathbf{n}_{\text{object}} \leftarrow$ GET-OBJECT-SPACE-NORMAL(r_{object}, t)

$dg.\mathbf{n} \leftarrow (M^{-1})^T$.TRANSFORM-DIRECTION($\mathbf{n}_{\text{object}}$)

return dg

}