418382 สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์
การบรรยายครั้งที่ 13

ประมุข ขันเงิน

pramook@gmail.com

วัตถุจริงๆ ตามธรรมชาติ

- ขรุขระ

วัตถุจริงๆ ตามธรรมชาติ

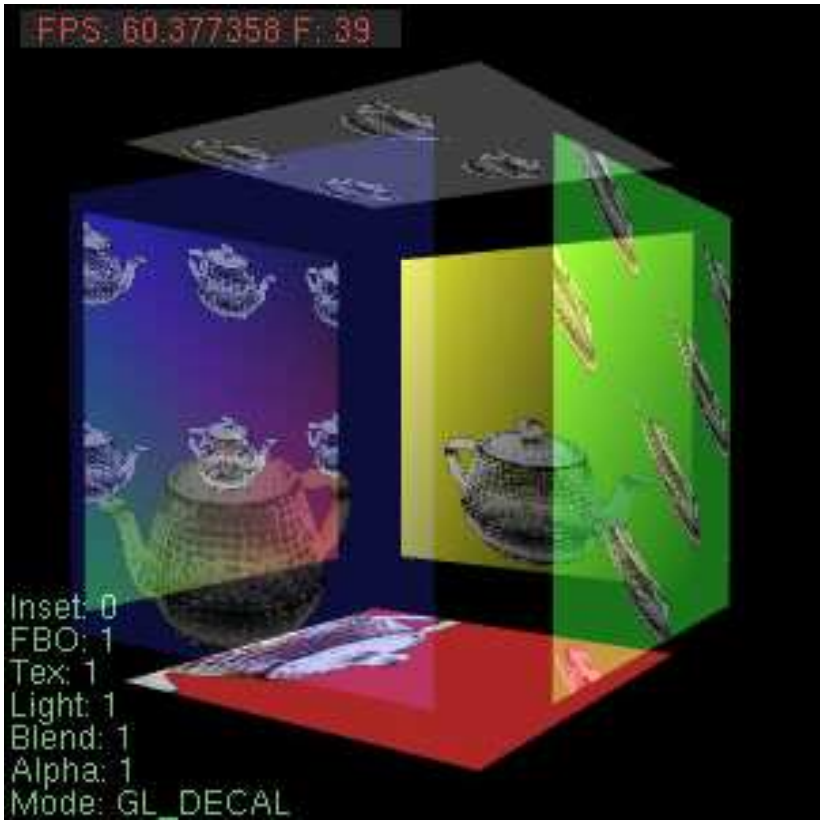
- ขรุขระ

วัตถุจริงๆ ตามธรรมชาติ

- ขรุขระ

วัตถุที่เราสร้างได้ด้วย OpenGL

- เรียบ



Texture Mapping

- เราสามารถใช้ texture เพิ่มรายละเอียดให้วัตถุได้
- แต่นั่นก็เป็นแค่การเปลี่ยนสี ไม่มีลักษณะความขรุขระหรือตื้นลึก

ภาพจาก Oliveira, Bishop, McAllister, **Relief Texture Mapping**

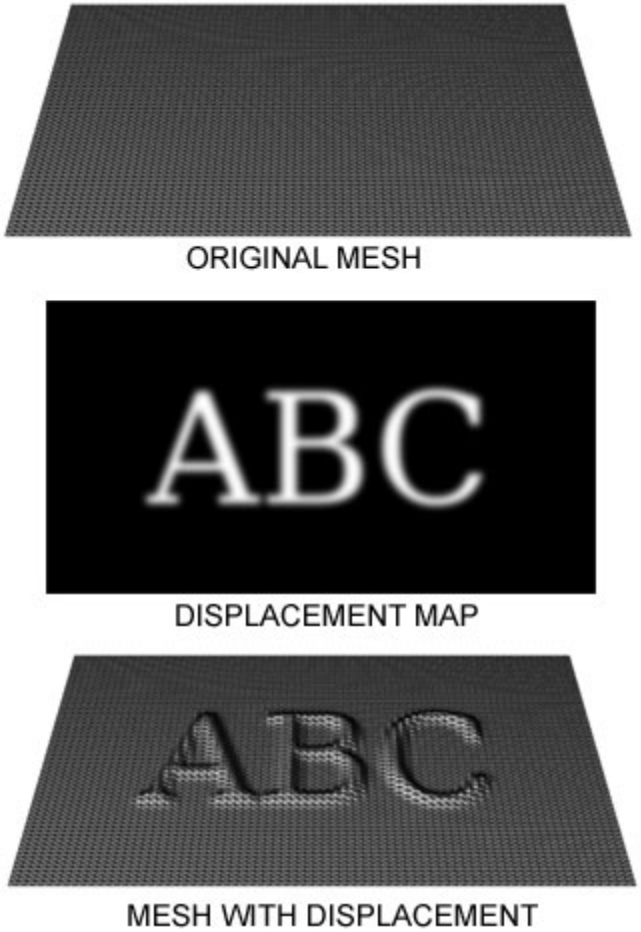
การทำให้พื้นผิวดูขรุขระ

1. Displacement Mapping
2. Normal Mapping
3. Bump Mapping
4. Relief Mapping

Displacement Mapping

- Cook (1984)
- เริ่มต้นจากพื้นผิวเรียบที่สร้างจาก polygon จำนวนมาก
- texture แสดงความ “นูน” ของพื้นผิว
- ใช้ texture ในการยกหรือกด vertex บนพื้นผิวนั้น
- ผลลัพธ์ได้เป็นพื้นผิวขรุขระจริงๆ

Displacement Mapping



Displacement Mapping

- ข้อดี
  - ได้ภาพที่สมจริงจริงๆ
- ข้อเสีย
  - ต้องใช้ polygon จำนวนมากเพื่อสร้างรายละเอียด
  - เปลืองหน่วยความจำ
  - วาดช้า

Normal Mapping

- จำลองความขรุขระโดยใช้ texture ระบุ normal ของแต่ละ fragment เอาเอง
- มี texture เพิ่มมาหนึ่งอันใช้เก็บ normal
- เวลาคำนวณ normal ให้นำ normal จาก texture มาใช้
  - ไม่ได้เอา normal ตามที่ OpenGL ให้มา
- นิยมใช้ตามซอฟต์แวร์สร้าง content สามมิติต่างๆ
  - 3ds Max, Maya, Blender, ฯลฯ

Normal Mapping

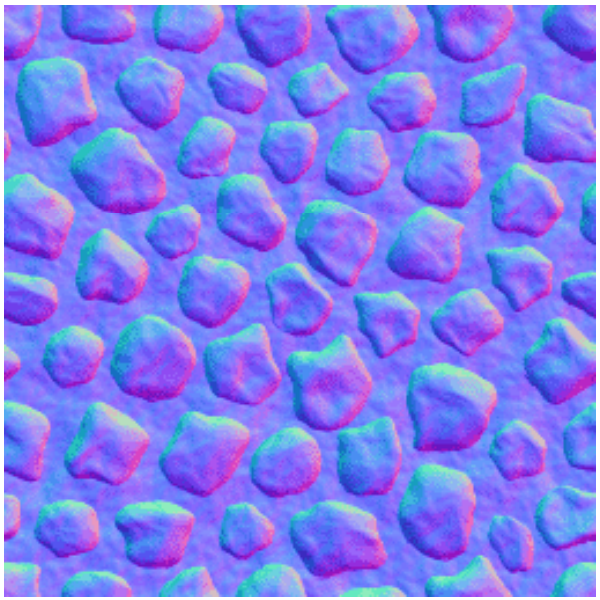
- การเก็บ normal ในรูปภาพ
  - ใช้ R แทนค่า x
  - ใช้ G แทนค่า y
  - ใช้ B แทนค่า z
- ความจริง normal จะเป็นเวกเตอร์หนึ่งหน่วย
  - ดังนั้น $z = sqrt(1 - x^2 - y^2)$
  - ฉะนั้นเก็บแค่ x และ y ก็ได้

Normal Mapping

- x, y, และ z จะมีค่าได้ตั้งแต่ -1 ถึง 1
- แต่ R, G, และ B มีค่าได้ตั้งแต่ 0 ถึง 1 เท่านั้น
- เพราะฉะนั้นต้องแทน
  - -1 ด้วย 0
  - 1 ด้วย 1
- กล่าวคือ R = x/2 + 0.5
- สีของ normal map จึงดูสว่างๆ

Normal Mapping

- ตัวอย่าง normal map



จาก http://www.bencloward.com/

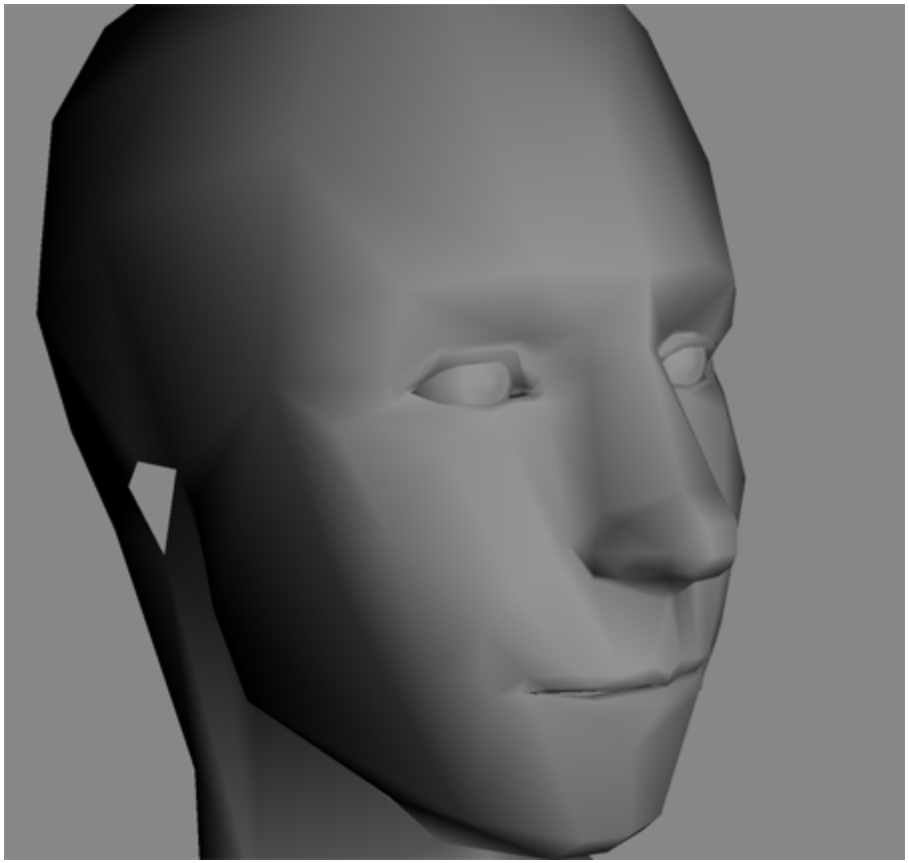
จาก http://planetpixelemporium.com/

Normal Map ทำให้เกิดความขรุขระได้อย่างไร?

- ใน Phong lighting model เราใช้ normal ในการคำนวณ
  - สี diffuse
  - สี specular
- เราไม่ได้ใช้ตำแหน่งของ fragment โดยตรงในการคำนวณสี
  - เว้นแต่ตอนที่หาเวกเตอร์จากตาไปยัง fragment
- พื้นผิวขรุขระ → ความสูงเปลี่ยนเร็ว → normal เปลี่ยนเร็ว
- ใช้ normal map เก็บ normal ไว้ → สามารถหาสีได้เหมือนกับพื้นผิวขรุขระ โดยไม่ต้องเก็บพื้นผิว

Normal Mapping

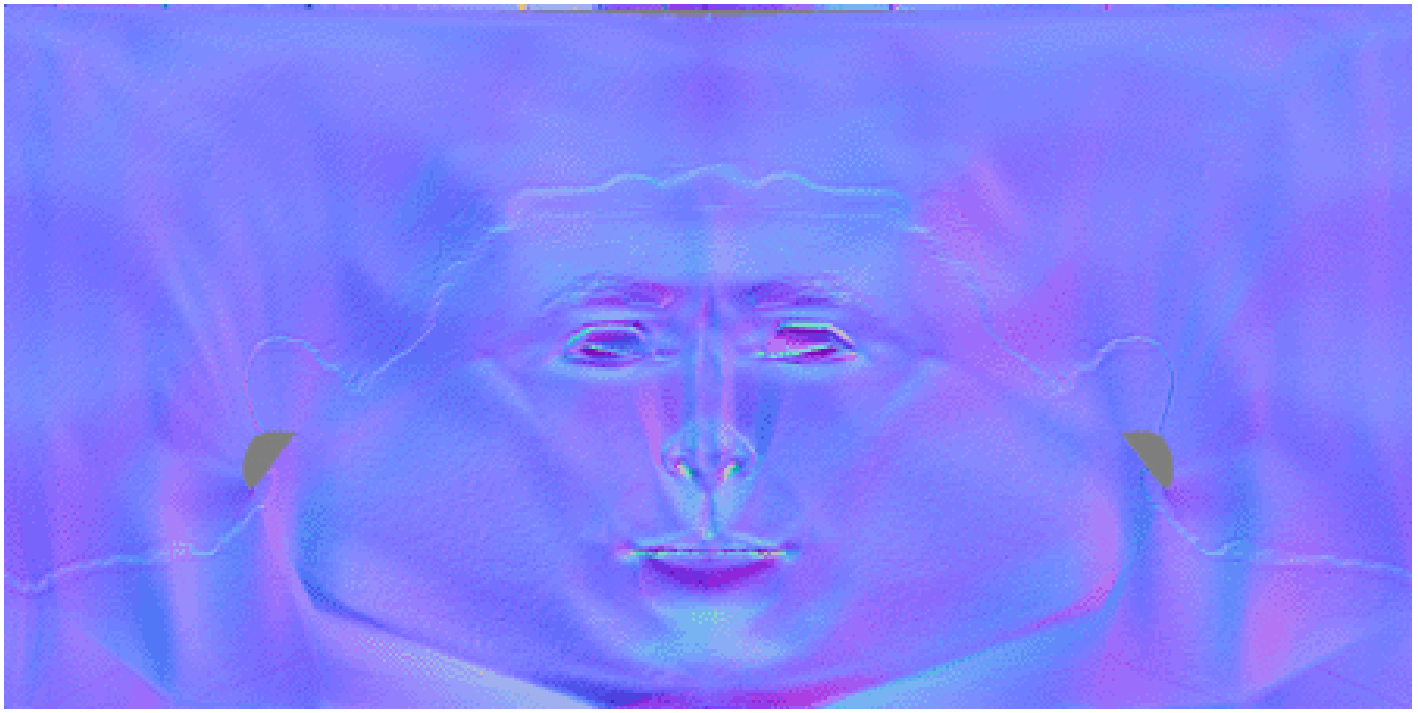
- โมเดลที่ทำจาก polygon เรียบๆ ไม่กี่ polygon



จาก http://www.bencloward.com/

Normal Mapping

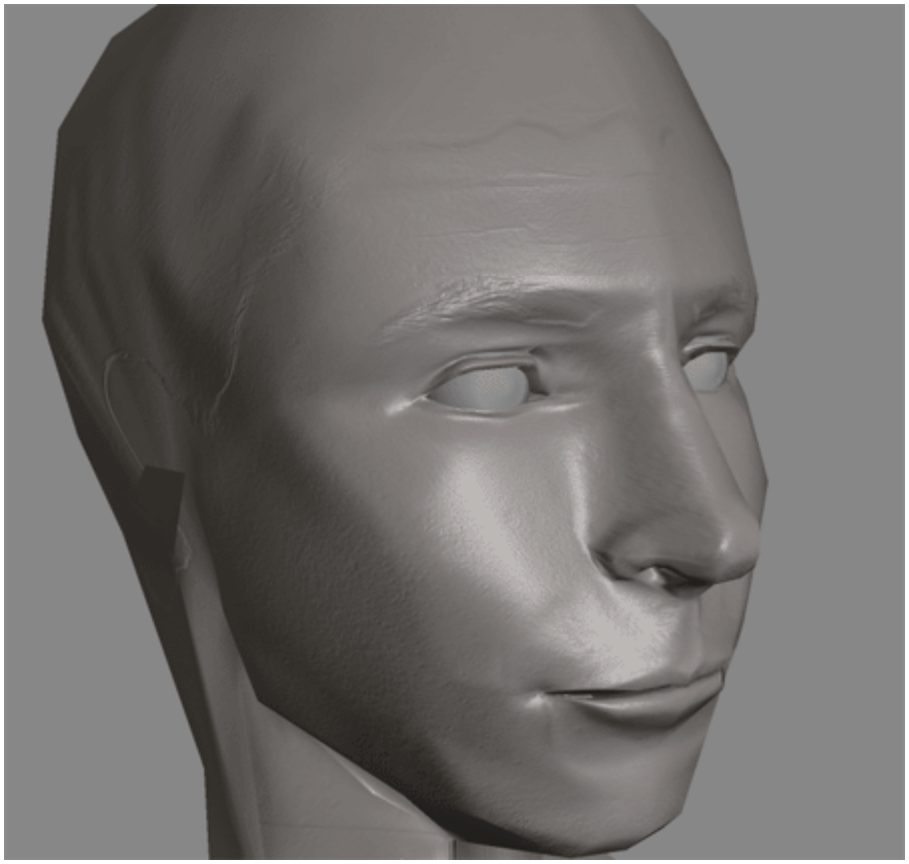
- เพิ่ม normal map



จาก http://www.bencloward.com/

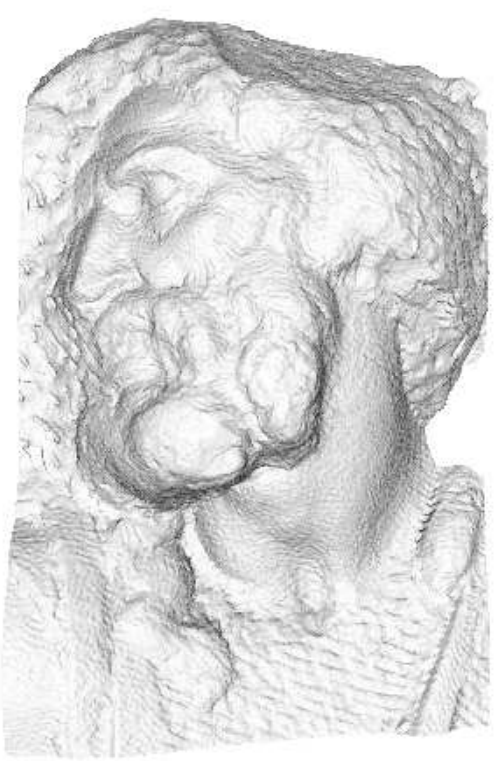
Normal Mapping

- ภาพที่มีรายละเอียดสูง

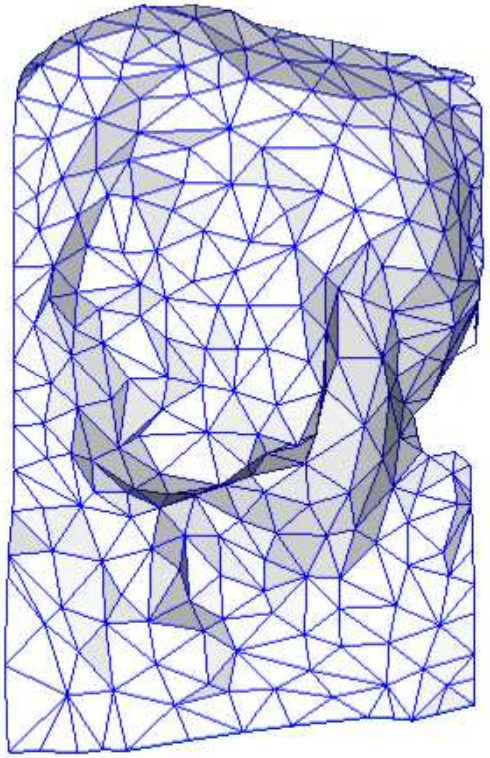


จาก http://www.bencloward.com/

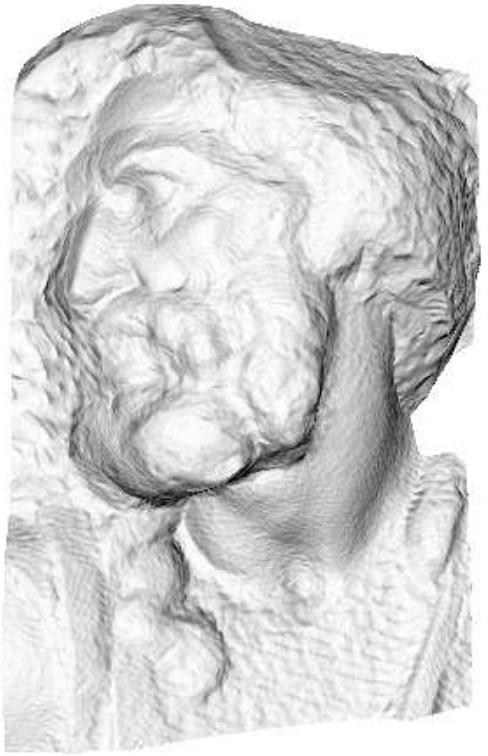
Normal Mapping



original mesh
4M triangles

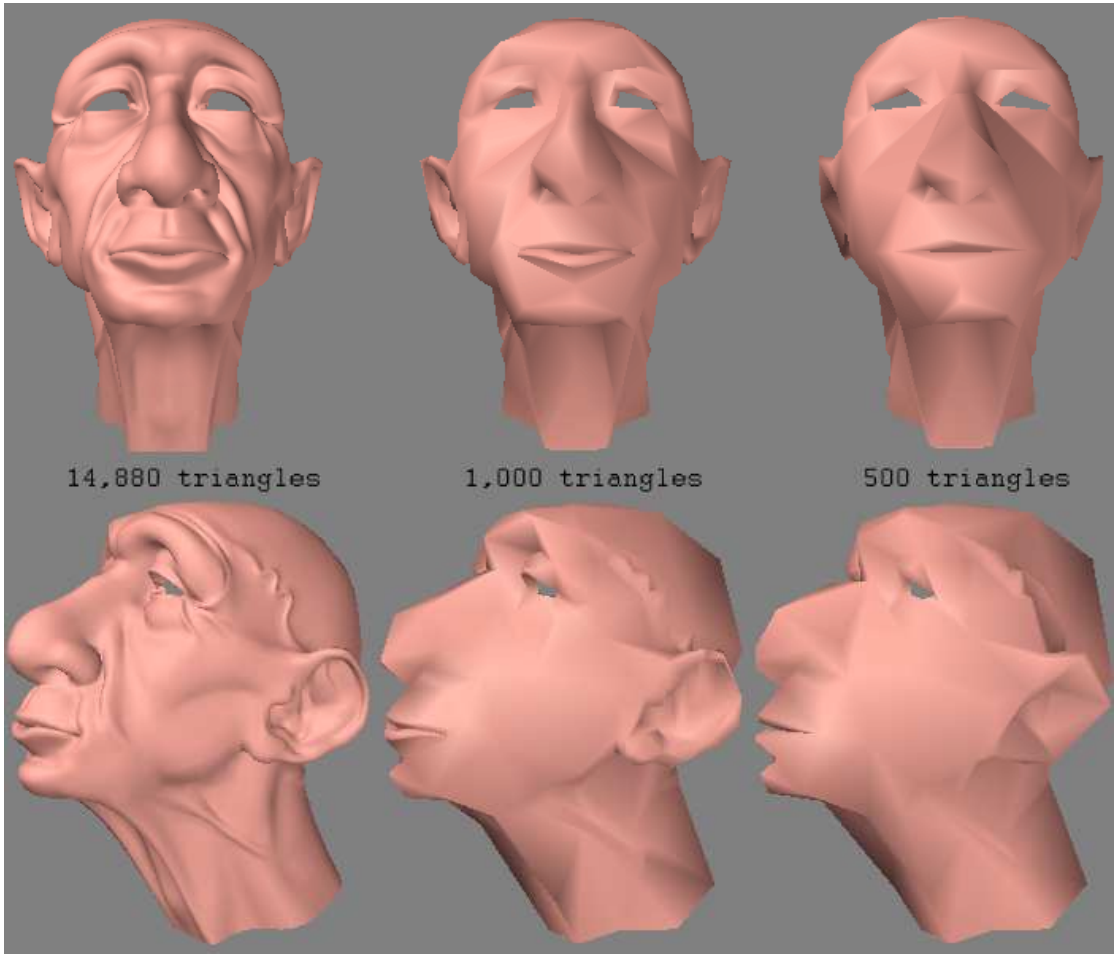


simplified mesh
500 triangles



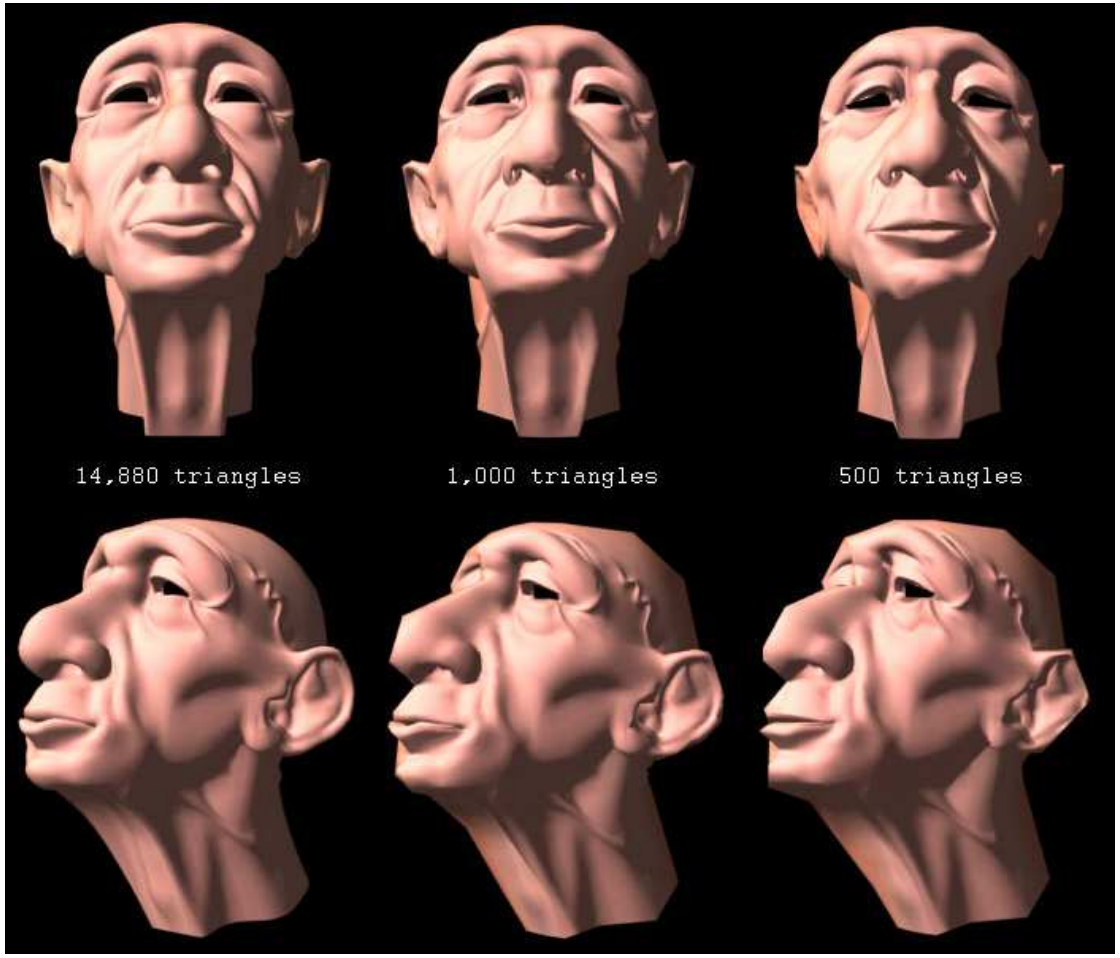
simplified mesh
and normal mapping
500 triangles

Normal Mapping



http://amber.rc.arizona.edu/lw/normalmaps.html

Normal Mapping



http://amber.rc.arizona.edu/lw/normalmaps.html

เราจะสร้าง normal map อย่างไร?

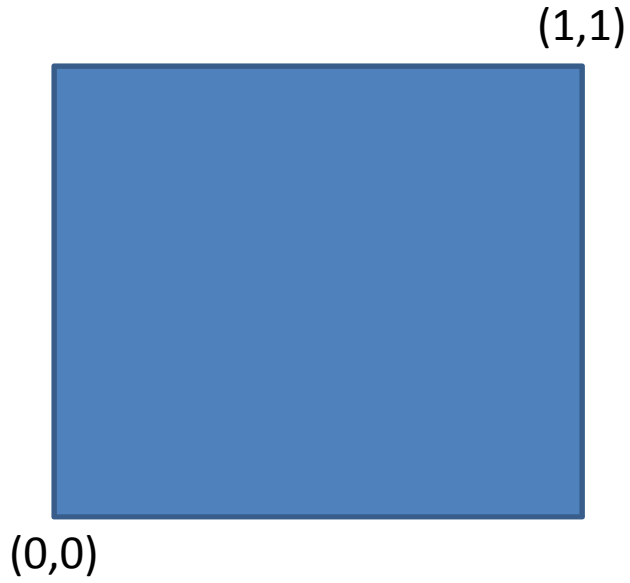
- มีสองวิธี
  - สร้างจากโมเดลรายละเอียดสูง
    - ไปดึงเอา normal ณ จากตำแหน่งต่างๆ มาเก็บไว้ใน texture
    - ซอฟต์แวร์สร้างเนื้อหาสามมิติส่วนใหญ่มี feature ให้คุณสามารถสร้าง normal map ได้
  - คำนวณจาก bump map

Bump Mapping

- วิธีการสร้างความขรุขระโดยใช้ texture ที่กำหนด “ความสูง” ของพื้นผิว
  - Texture แบบเดียวกับที่ใช้ทำ displacement mapping
- ต้องการได้ normal สำหรับแต่ละ fragment เหมือน normal map
  - Normal map → ผู้ใช้กำหนด normal ให้
  - Bump map → คำนวณเองจากความสูง

Surface Parameterization

- ก่อนทำ **bump mapping** ได้เราจะต้องสามารถระบุจุดแต่ละจุดบนพื้นผิวได้ด้วยพิกัด **(u,v)**
- ยกตัวอย่างเช่น ถ้าเป็นพื้นผิวสี่เหลี่ยมธรรมดา เราอาจจะกำหนดพิกัด **(u,v)** ดังต่อไปนี้



Surface Parameterization

- ในกรณีที่เป็นพื้นผิวอื่นๆ surface parameterization คือ ฟังก์ชัน $\mathbf{p} : \mathbb{R}^2 \to \mathbb{R}^3$
- ฟังก์ชันนี้จะรับพิกัด (u,v) แล้วคืนจุด (x,y,z) ในสามมิติมาให้
- สมมติว่าพื้นผิวสี่เหลี่ยมจัตุรัสในข้อที่แล้วมี
  - มุมล่างซ้ายที่จุด (-1,-1,0) และ
  - มุมบนขวาที่จุด (1,1,0) แล้ว

  เราจะได้ว่า

$$\mathbf{p}(u,v) = \begin{bmatrix} 2u-1 \\ 2v-1 \\ 0 \end{bmatrix}$$

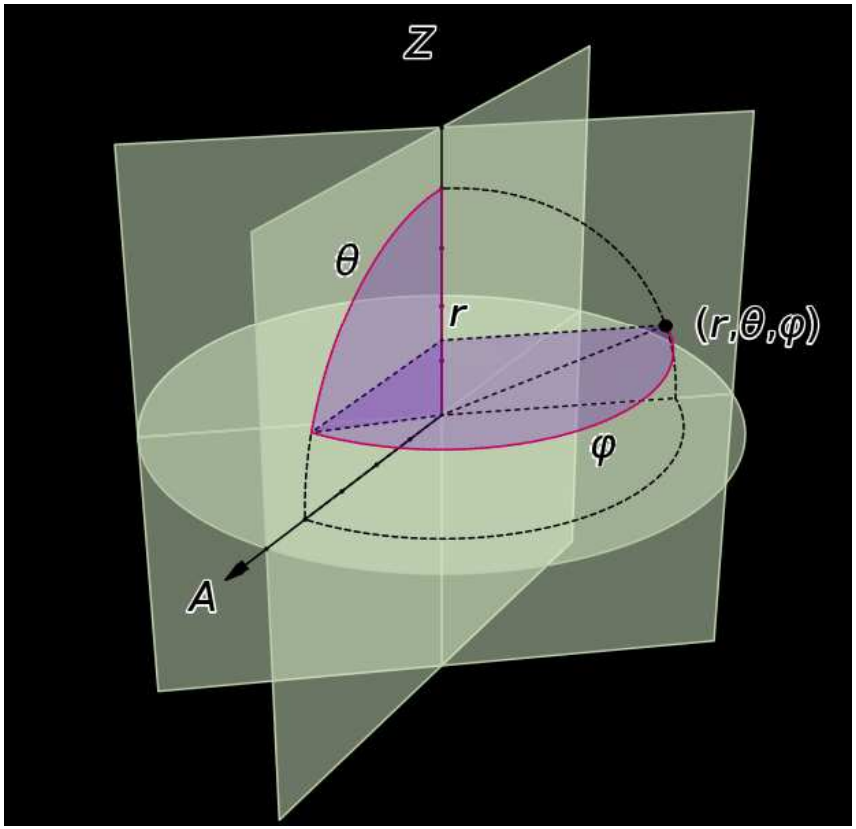
Surface Parameterization

- สำหรับทรงกลม เราอาจจะใช้ spherical coordinate

$$\mathbf{p}(\theta,\phi) = \begin{bmatrix} \cos\phi\sin\theta \\ \sin\phi\sin\theta \\ \cos\theta \end{bmatrix}$$

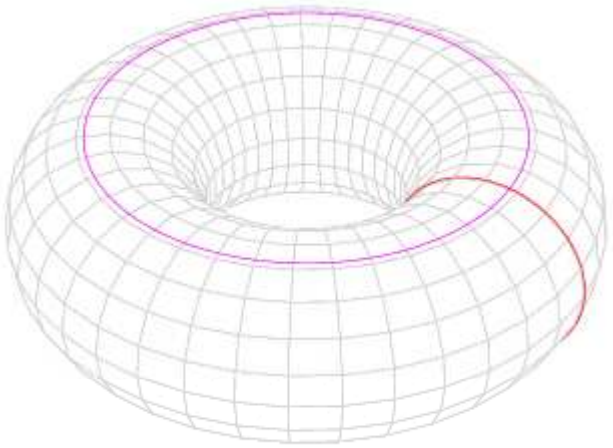
- แต่ปกติแล้ว u และ v จะมีค่าตั้งแต่ 0 ถึง 1 ฉะนั้นเราจะใช้

$$\mathbf{p}(u,v) = \begin{bmatrix} \cos(2\pi v)\sin(\pi u) \\ \sin(2\pi v)\sin(\pi u) \\ \cos(\pi u) \end{bmatrix}$$



Surface Parameterization

- ถ้าเป็นโดนัทที่มีรัศมีหลอดเท่ากับ r และรัศมีของวงกลมใหญ่เท่ากับ R เราอาจใช้



$$\mathbf{p}(u,v) = \begin{bmatrix} (R + r\cos(2\pi v))\cos(2\pi u) \\ (R + r\cos(2\pi v))\sin(2\pi u) \\ r\sin(2\pi v) \end{bmatrix}$$

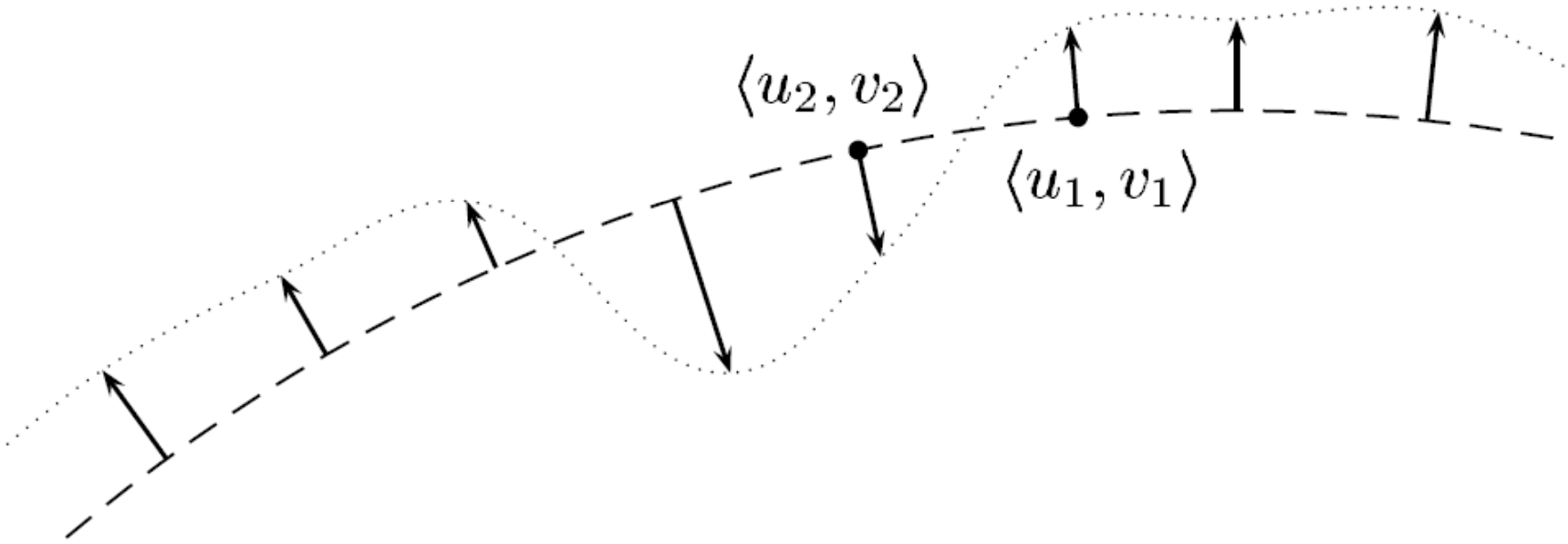
Bump Map

- Bump map เป็นฟังก์ชัน $d : \mathbb{R}^2 \to \mathbb{R}$ โดยที่ $d(u, v)$ มีค่าเท่ากับระยะทางที่พื้นผิว ณ พิกัด (u,v) ถูกทำให้สูงขึ้นหรือต่ำลงตามแนวของ normal ที่จุดนั้น
- ฉะนั้นพื้นผิวใหม่ที่ได้คือ

$$\mathbf{p}^*(u, v) = \mathbf{p}(u, v) + d(u, v)\mathbf{n}(u, v)$$

โดยที่ $\mathbf{n}(u, v)$ คือ normal ที่พื้นผิวพิกัด (u,v)

Bump Map



การคำนวณ Normal

- เราต้องการคำนวณ normal ของพื้นผิวใหม่ **p***
- ปกติแล้วเขาจะคำนวณ normal กันอย่างไร?
- ถ้าเรามีฟังก์ชัน **p** ของพื้นผิวใดๆ เราจะได้ว่า

$$\mathbf{n}(u,v) = \text{normalize}\left(\frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v}\right)$$

เมื่อ

$\frac{\partial \mathbf{p}}{\partial u}$ คือ partial derivative ของ **p** เมื่อเทียบกับ u

$\frac{\partial \mathbf{p}}{\partial v}$ คือ partial derivative ของ **p** เมื่อเทียบกับ v

normalize คือการทำให้เป็น vector หนึ่งหน่วย

ตัวอย่าง

- สำหรับฟังก์ชัน **p** ของรูปสี่เหลี่ยม

$$\mathbf{p}(u,v) = \begin{bmatrix} 2u-1 \\ 2v-1 \\ 0 \end{bmatrix}$$

- เราจะได้ว่า

$$\frac{\partial \mathbf{p}}{\partial u} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}, \frac{\partial \mathbf{p}}{\partial v} = \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}$$

$$\mathbf{n}(u,v) = \text{normalize}\left(\frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v} \right) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

การคำนวณ Normal ของ **p***

- เราได้ว่า

$$\mathbf{n}^*(u,v) = \text{normalize}\left(\frac{\partial \mathbf{p}^*}{\partial u} \times \frac{\partial \mathbf{p}^*}{\partial v}\right)$$

- เพื่อความง่าย เราจะคำนวณ

$$\mathbf{m}^* = \frac{\partial \mathbf{p}^*}{\partial u} \times \frac{\partial \mathbf{p}^*}{\partial v}$$

ก่อน

- **m*** เป็นเวกเตอร์ที่ตั้งฉากกับพื้นผิว แต่มันไม่ใช่เวกเตอร์หนึ่งหน่วย

การคำนวณ Normal ของ **p***

- เราจะได้อีกว่า

$$\frac{\partial \mathbf{p}^*}{\partial u} = \frac{\partial(\mathbf{p} + d\mathbf{n})}{\partial u} = \frac{\partial \mathbf{p}}{\partial u} + \frac{\partial(d\mathbf{n})}{\partial u}$$
$$= \frac{\partial \mathbf{p}}{\partial u} + \frac{\partial d}{\partial u}\mathbf{n} + \frac{\partial \mathbf{n}}{\partial u}d$$

- ในทำนองเดียวกัน

$$\frac{\partial \mathbf{p}^*}{\partial v} = \frac{\partial \mathbf{p}}{\partial v} + \frac{\partial d}{\partial v}\mathbf{n} + \frac{\partial \mathbf{n}}{\partial v}d$$

การคำนวณ Normal ของ **p***

- เพื่อให้การคำนวณง่ายขึ้น เราจะประมาณว่า

$$\frac{\partial \mathbf{p}^*}{\partial u} \approx \frac{\partial \mathbf{p}}{\partial u} + \frac{\partial d}{\partial u}\mathbf{n}$$
$$\frac{\partial \mathbf{p}^*}{\partial v} \approx \frac{\partial \mathbf{p}}{\partial v} + \frac{\partial d}{\partial v}\mathbf{n}$$

- ทั้งนี้เป็นเพราะว่า $\frac{\partial \mathbf{n}}{\partial u}$ และ $\frac{\partial \mathbf{n}}{\partial v}$ นั้นคำนวณลำบาก
- การตัดเทอมสุดท้ายออกยังค่อนข้างสมเหตุสมผล เนื่องจาก
  - d(u,v) มีค่าน้อย (ถ้ามีค่ามากเกินไปก็ควรจะทำสร้างโมเดลใหม่เสีย)
  - พื้นผิวตั้งต้นเป็นพื้นผิวเรียบ ดังนั้น $\frac{\partial \mathbf{n}}{\partial u}$ และ $\frac{\partial \mathbf{n}}{\partial v}$ จึงมีค่าน้อย

    (กล่าวคือ normal ไม่เปลี่ยนแปลงอย่างรวดเร็ว)

การคำนวณ Normal ของ **p***

- ฉะนั้น

$$\mathbf{m}^* \approx \left(\frac{\partial \mathbf{p}}{\partial u} + \frac{\partial d}{\partial u}\mathbf{n}\right) \times \left(\frac{\partial \mathbf{p}}{\partial v} + \frac{\partial d}{\partial v}\mathbf{n}\right)$$

$$= \left(\frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v}\right) + \left(\frac{\partial d}{\partial u}\mathbf{n} \times \frac{\partial \mathbf{p}}{\partial v}\right) - \left(\frac{\partial d}{\partial v}\mathbf{n} \times \frac{\partial \mathbf{p}}{\partial u}\right)$$

- แล้ว

$$\mathbf{n}^* = \text{normalize}(\mathbf{m}^*) = \frac{\mathbf{m}^*}{\|\mathbf{m}^*\|}$$

การคำนวณ Normal ของ **p***

- ที่เหลือคือต้องคำนวณ $\frac{\partial d}{\partial u}$ และ $\frac{\partial d}{\partial v}$
- ถ้ามีสูตรของ d เราสามารถทำการคำนวณมันได้อย่างง่ายดาย
- แต่ปกติแล้ว d จะให้มาเป็น texture
- อย่างไรก็ดี เราสามารถประมาณอนุพันธ์ได้ดังต่อไปนี้

$$\left.\frac{\partial d}{\partial u}\right|_{(u',v')} \approx \frac{d(u'+\varepsilon, v') - d(u', v')}{\varepsilon}$$

$$\left.\frac{\partial d}{\partial u}\right|_{(u',v')} \approx \frac{d(u', v'+\varepsilon) - d(u', v')}{\varepsilon}$$

โดยที่ ε คือค่าคงที่ที่มีค่าน้อยค่าหนึ่ง

การคำนวณ Normal ของ **p***

- ในภาษา Cg เราจะให้ d เป็นตัวแปรประเภท sampler2D
- สมมติว่า d เป็น texture ที่มีความกว้าง w pixel และสูง h pixel
- เราสามารถคำนวณ partial derivative ของ d ณ พิกัด (u,v) ได้ดังต่อไปนี้

```
float dddu = w*(tex2D(d, float2(u+1.0f/w,v)) -
    tex2D(d, float2(u,v))).r;
float dddv = h*(tex2D(d, float2(u,v+1.0f/h)) -
    tex2D(d, float2(u,v))).r;
```

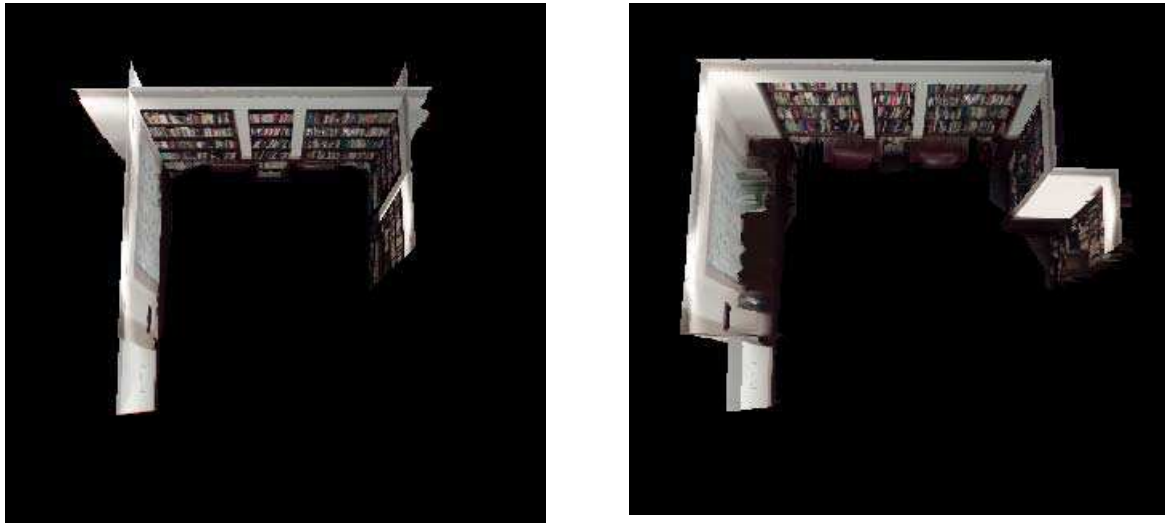
Normal Mapping

ข้อจำกัดของ Normal และ Bump Mapping

- ไม่มีการทอดเงาลงบนตัวเอง
- ไม่มี parallax
  - ส่วนที่นูนขึ้นมาเคลื่อนที่ไปพร้อมกับส่วนที่เว้าลงไป
  - ความจริงแล้วทั้งสองส่วนนี้ควรจะเคลื่อนที่เมื่อเทียบกับตาแตกต่างกันเล็กน้อย

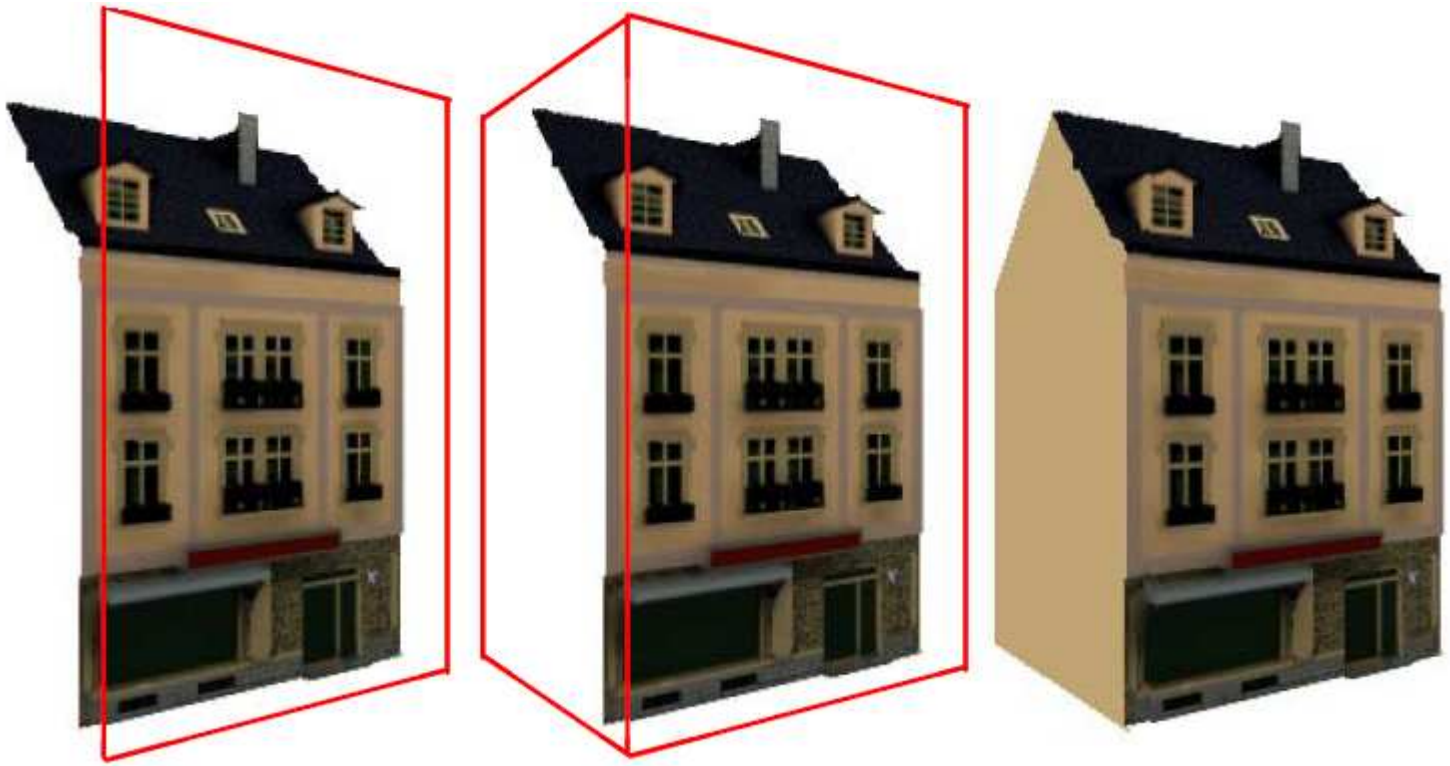
Relief Mapping

- เทคนิคการแสดงผลพื้นผิวที่มีรายละเอียดสูงโดยใช้ข้อมูลความสูงของ fragment แต่ละ fragment ประกอบ
- คิดว่าจริงๆ แล้วพื้นผิวเป็น “กล่อง”



ภาพจาก Oliveira, Bishop, McAllister, **Relief Texture Mapping**

Relief Mapping



ภาพจาก Oliveira, Bishop, McAllister, **Relief Texture Mapping**

Relief Mapping

(a) (b) (c)

Figure 4: Rendering comparison from the same viewpoint. (a) Color image rendered as a conventional texture. (b) Relief texture mapping rendering. (c) Rendering of the color and depth data in Figure 3 as a mesh of micro-polygons.

Policarpo, Oliveira, Comba. **Real-Time Relief Mapping on Arbitrary Polygonal Surfaces**

Relief Map

- สำหรับพื้นผิวที่ต้องการแสดง จะมี texture อยู่ 2 texture
  - Normal map
  - Height map บอกความ “ลึก” ของแต่ละ texel
    - 0 = ตื้น, 1 = ลึก
- สามารถแทนทั้งหมดข้างบนนี้ได้ด้วย texture RGBA แค่ texture เดียว
  - Normal map → RGB
  - Height map → A

Relief Map

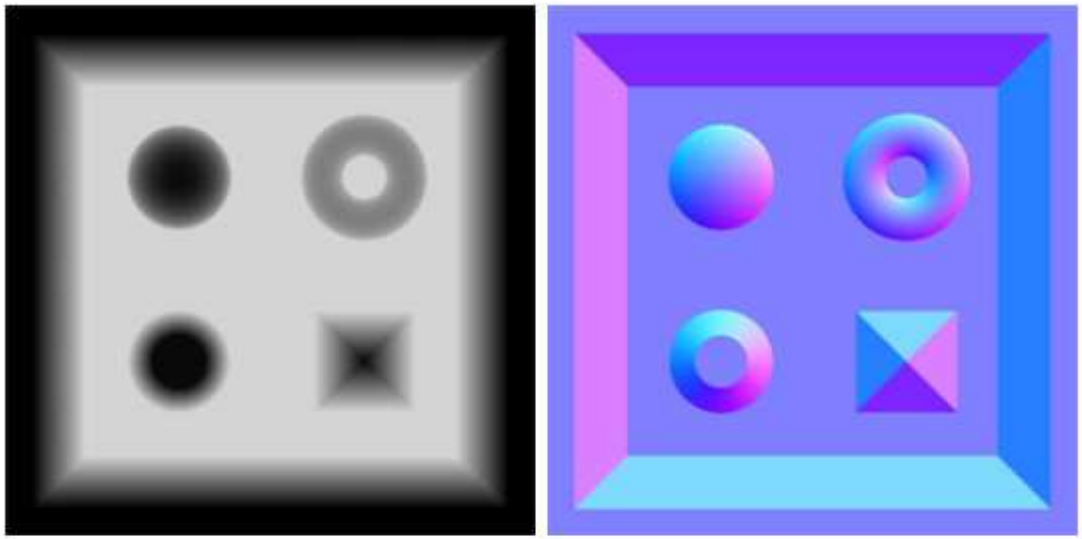


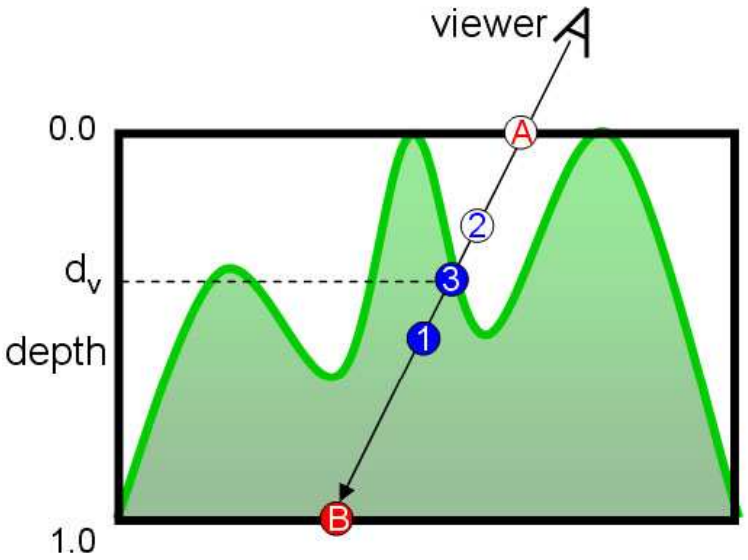
Figure 6: A relief texture represented by a depth (left) and a normal map (right). The normals are mapped to the [0,1] range and stored as an RGB image.

Policarpo, Oliveira, Comba. **Real-Time Relief Mapping on Arbitrary Polygonal Surfaces**

การแสดงผล Relief Map

- วาดพื้นผิวเรียบที่มี relief map ที่ติดอยู่ด้วยตามธรรมดา
- เมื่อไปถึงขั้นของการประมวลผล fragment เราจะมีข้อมูล
  - ตำแหน่งของ fragment
  - Texture coordinate
- คิดว่า fragment อยู่ที่ “ผิวหน้า” ของกล่อง
- ลากรังสีจากสายตาไปยัง fragment
- เราต้องการหาว่ารังสีนั้นตัดกับพื้นผิวจริงๆ ที่ไหน

การแสดงผล Relief Map



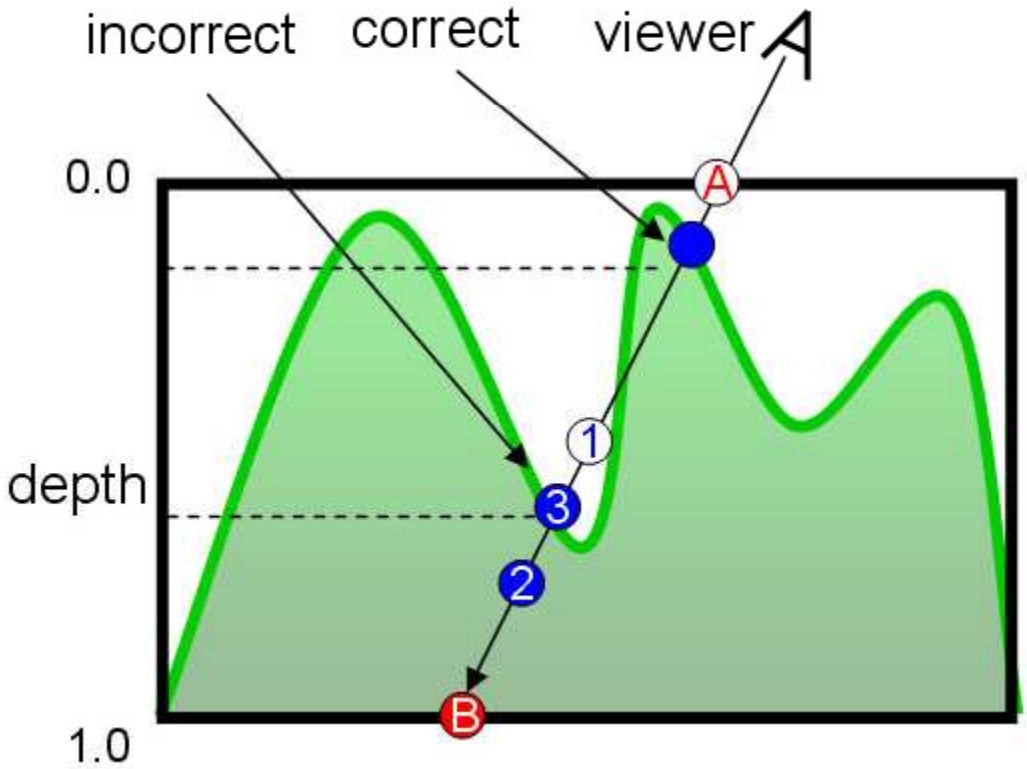
- ตำแหน่ง A = ตำแหน่งของ fragment
- ต้องการหาจุด (3) ซึ่งเป็นจุดแรกที่รังสีตัดกับพื้นผิว

การแสดงผล Relief Map

- เราสามารถจุดที่รังสีตัดกับวัตถุเป็นจุดแรกได้ด้วยการทำ binary search
  - คำนวณจุด B ซึ่งเป็นจุดที่รังสีเดินทางถึงความลึก 1
  - คำนวณจุดตรงกลางระหว่าง A กับ B (ในภาพคือจุด (1))
  - ถ้าจุดตรงกลางนั้นอยู่ใต้พื้นผิว
    - เปลี่ยน B ไปเป็นจุด (1)
  - ถ้าจุดตรงกลางอยู่เหนือพื้นผิว
    - เปลี่ยน A ไปอยู่ที่จุด (1)
  - ทำเช่นนี้ไปเรื่อยๆ จน A กับ B ใกล้กันมากๆ (ส่วนใหญ่ 8 รอบก็พอ)
  - ใช้ normal และสีที่จุดกลางระหว่าง A กับ B ในการแสดงผล

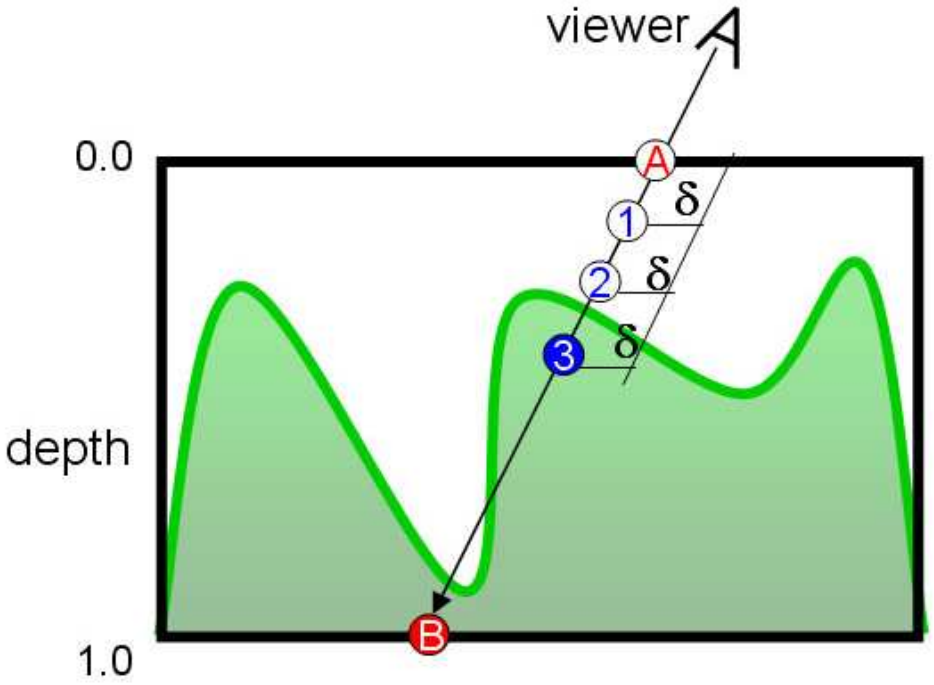
การแสดงผล Relief Map

- อย่างไรก็ดีการหาจุดดังกล่าวอาจทำให้เราเห็นพื้นผิวที่ถูกบังได้
- กรณีที่มีพื้นผิวแคบๆ ระหว่างจุด A กับ (1)



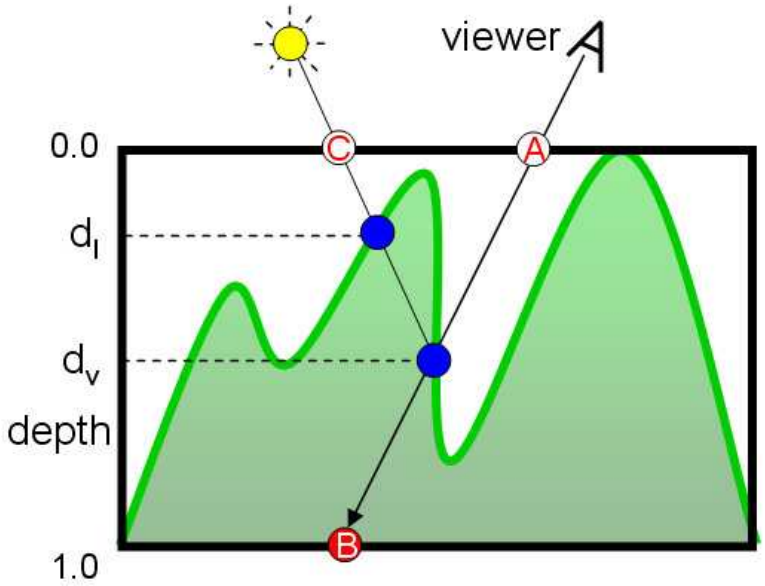
การแสดงผล Relief Map

- เพื่อหลีกเลี่ยงความผิดพลาดดังกล่าว เราจะทดสอบจุดที่อยู่ห่างจากจุด A ไปทีละ δ จนกระทั่งเจอพื้นผิวแรก
- หลังจากนั้นใช้ binary search



การแสดงผล Relief Map

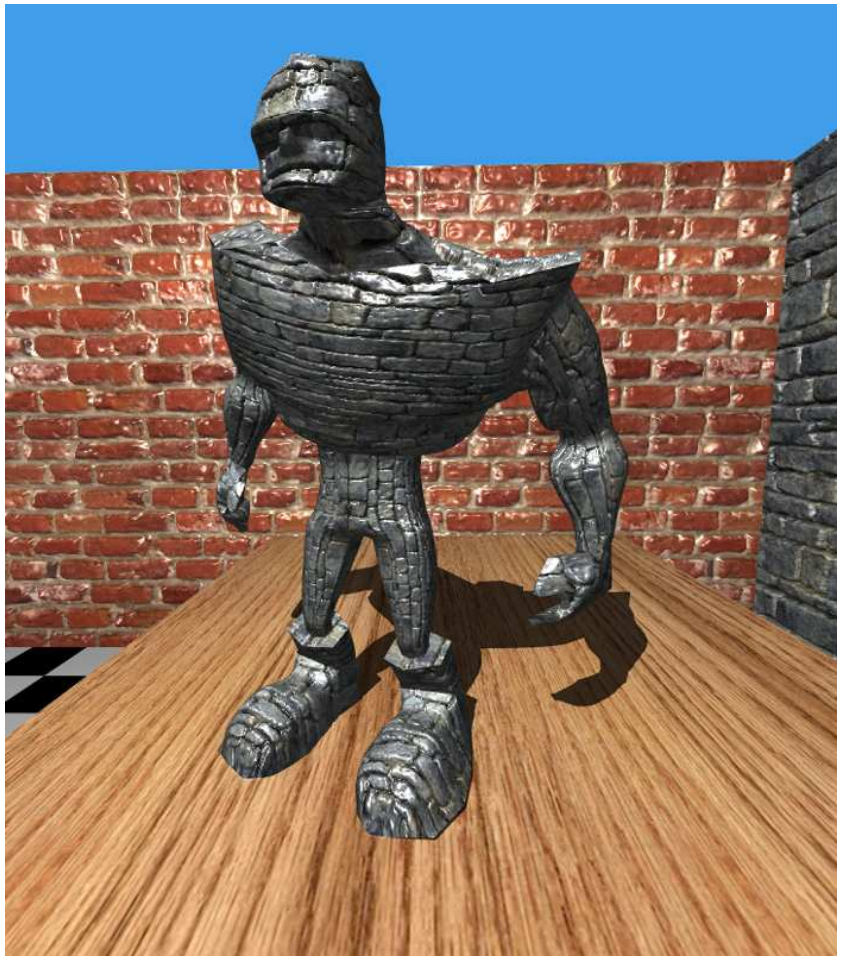
- เราสามารถนำอัลกอริทึมเดียวกันไปใช้หาว่าจุดที่เห็นได้รับแสงหรือไม่ด้วย
- ทำการลากเส้นจากจุดบนพื้นผิวไปยังแหล่งกำเนิดแดง
- หาว่ามีพื้นผิวระหว่างจุดตัดกับแหล่งกำเนิดแสดงหรือไม่



Relief Mapping

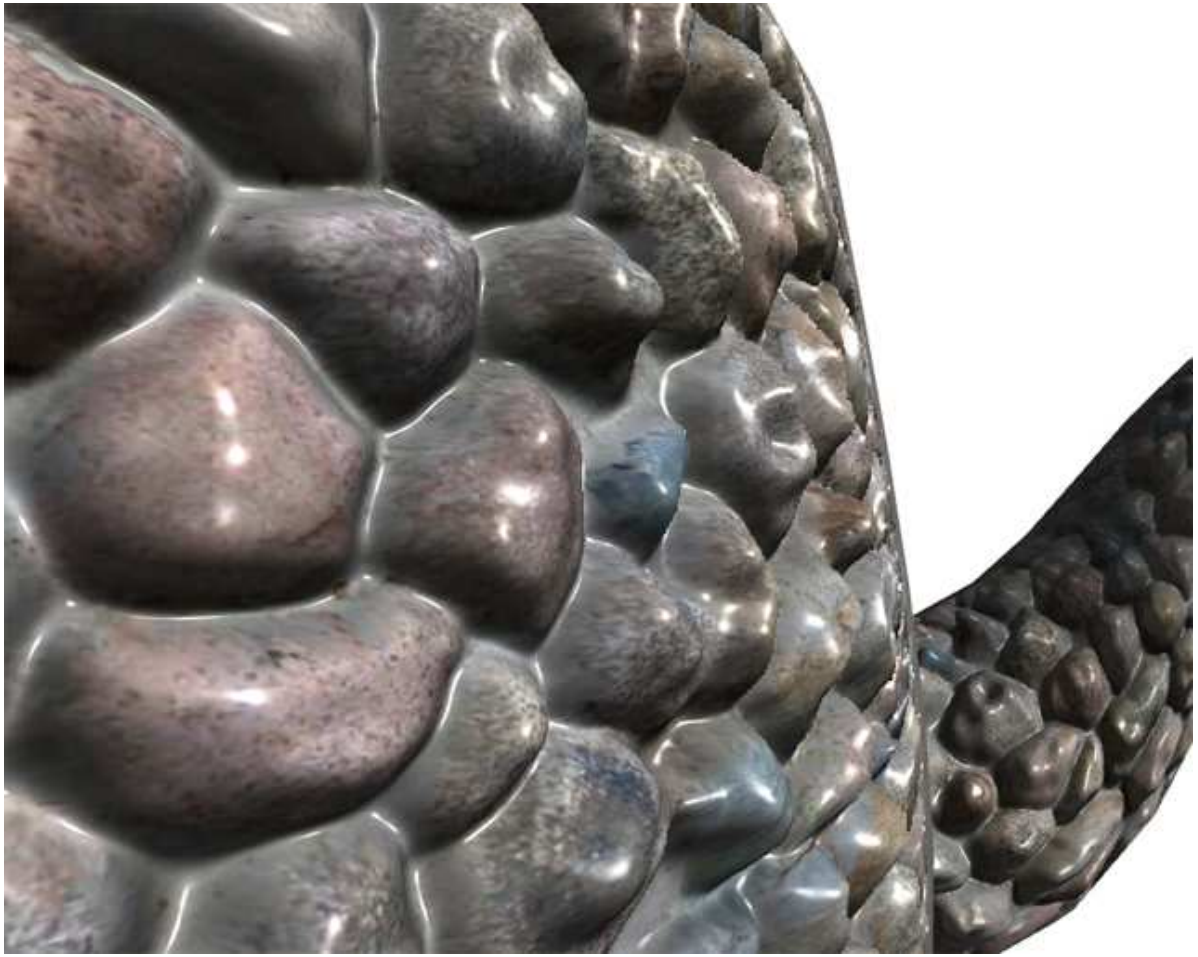
Policarpo, Oliveira, Comba. **Real-Time Relief Mapping on Arbitrary Polygonal Surfaces**

Relief Mapping

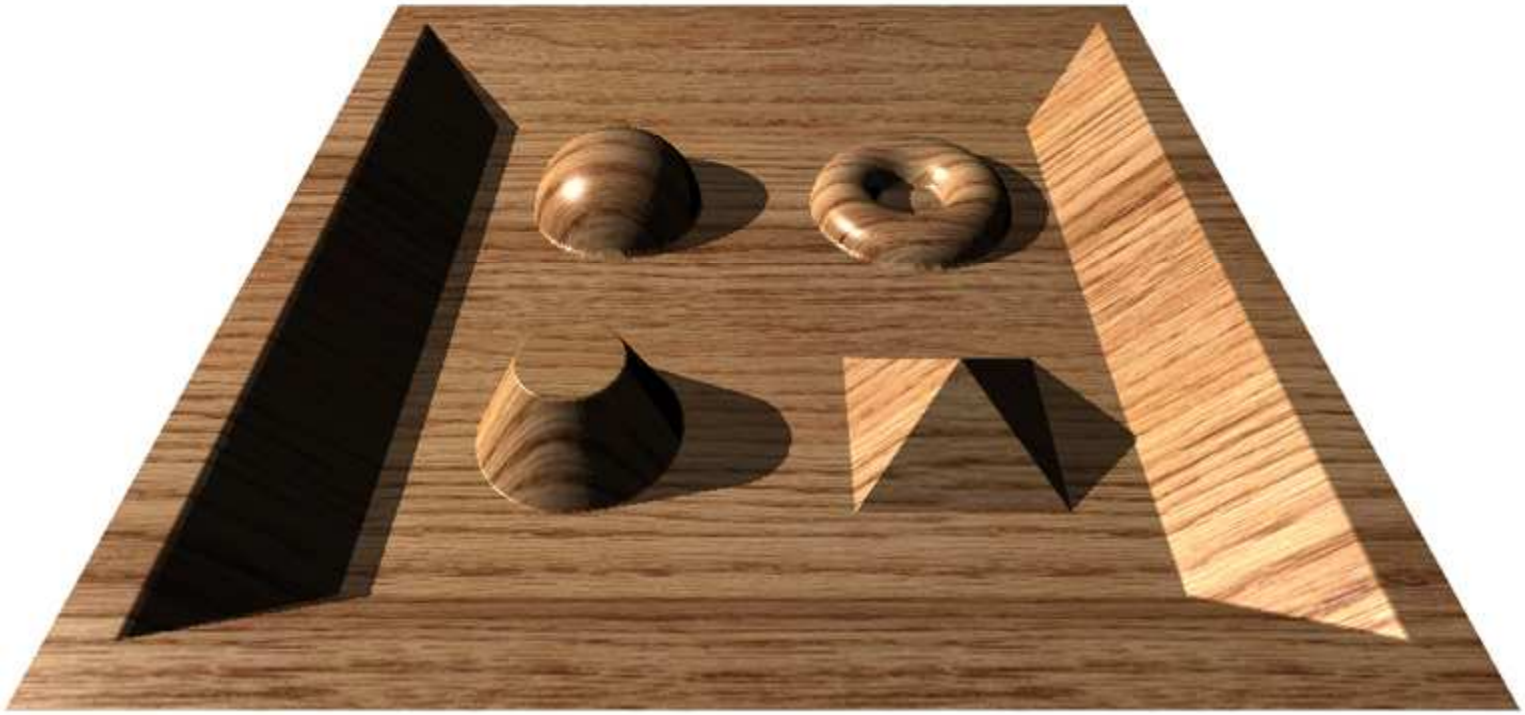


Policarpo, Oliveira, Comba. **Real-Time Relief Mapping on Arbitrary Polygonal Surfaces**

Relief Mapping



Relief Mapping



ข้อจำกัดของ Relief Mapping

- ไม่สามารถทำให้เงาของวัตถุขรุขระตามไปด้วยได้