418341 สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์
การบรรยายครั้งที่ 10

ประมุข ขันเงิน

Blending

- เวลาเรากำหนดสีด้วย glColor*(...) เราสามารถกำหนด alpha ของสีได้ด้วย
- Alpha มีไว้ใช้ควบคุมการทำ blending
- Blending คือการคำนวณสีของ pixel ใหม่เมื่อ fragment ใหม่กำลังจะถูกเขียนลงบน framebuffer โดยนำเอาสีของ pixel เดิมมาร่วมคำนวณด้วย

Blending (ต่อ)

- ที่เราเรียนผ่านมาใช้วิธีการคำนวณสีของ pixel สองแบบ
  - เอาสีของ fragment ที่มาทีหลังเขียนทับลงบน framebuffer
  - เก็บสีของ fragment ที่ใกล้ตาที่สุดเอาไว้ (z-buffer algorithm)
- OpenGL สามารถทำการคำนวณสีได้หลายแบบกว่านี้มาก
- การคำนวณสีหลายๆ แบบ ที่ไม่ใช่สองแบบข้างบนนี้เราก็เรียกตีขลุมว่า blending

การใช้และเลิกใช้ blending

- เวลาจะทำ blending ต้องสั่งคำสั่ง

  glEnable(GL_BLEND);

- เวลาจะเลิกทำ blending (กลับไปใช้ z-buffer หรือการวาดทับ) ให้สั่ง

  glDisable(GL_BLEND);

Source และ Destination

- เวลาทำ blending จะมีข้อมูลสีที่เกี่ยวข้องสองตัว
  - Source คือ สี RGBA ของ fragment อันใหม่ที่กำลังจะถูกเขียนลงบน pixel หนึ่งของ framebuffer
  - Destination คือ สี RGBA ของ pixel ที่อยู่บน framebuffer เรียบร้อยแล้ว
- เราอาจคิดได้ว่า destination คือสีที่คำนวณจากสีของ fragment ที่เกิดจากรูปร่างที่ถูกสั่งวาดก่อน source ทั้งหมด

สมการ Blending

- การคำนวณสีใหม่จะเป็นไปตามสมการข้างล่างนี้

$$\text{new color} = (R_s S_r + R_d D_r, G_s S_g + G_d D_g, B_s S_b + B_d D_b, A_s S_a + A_d D_a)$$

โดยที่

(S_r, S_g, S_b, S_a) คือสีของ source

(D_r, D_g, D_b, D_a) คือสีของ destination

(R_s, G_s, B_s, A_s) เรียกว่า “blending factor” ของ source

(R_d, G_d, B_d, A_d) เรียกว่า “blending factor” ของ dest’

คำสั่งกำหนด Blending Factor

- เราสามารถกำหนด blending factor ได้ด้วยคำสั่ง

  glBlendFunc(GLenum src, GLenum dest)

  และคำสั่ง

  glBlendFuncSeparate(

  GLenum srcRGB, GLenum srcAlpha,

  GLenum destRGB, GLenum destAlpha)

- สองฟังก์ชันนี้ต่างกันตรงที่ glBlendFuncSeparate สามารถกำหนด blending factor โดยแยก RGB กับ A ออกจากกันได้

ค่าคงที่กำหนดสำหรับ Blending Factor

- ค่าที่ใส่ให้ argument แต่ละตัวเป็นค่าคงที่ในตารางต่อไปนี้

Constant	Relevant Factor	Computed Blend Factor
GL_ZERO	source or destination	(0, 0, 0, 0)
GL_ONE	source or destination	(1, 1, 1, 1)
GL_DST_COLOR	source	(Rd, Gd, Bd, Ad)
GL_SRC_COLOR	destination	(Rs, Gs, Bs, As)
GL_ONE_MINUS_DST_COLOR	source	(1, 1, 1, 1)-(Rd, Gd, Bd, Ad)
GL_ONE_MINUS_SRC_COLOR	destination	(1, 1, 1, 1)-(Rs, Gs, Bs, As)
GL_SRC_ALPHA	source or destination	(As, As, As, As)
GL_ONE_MINUS_SRC_ALPH A	source or destination	(1, 1, 1, 1)-(As, As, As, As)
GL_DST_ALPHA	source or destination	(Ad, Ad, Ad, Ad)
GL_ONE_MINUS_DST_ALPH A	source or destination	(1, 1, 1, 1)-(Ad, Ad, Ad, Ad)
GL_SRC_ALPHA_SATURATE	source	(f, f, f, 1); f=min(As, 1-Ad)

กำหนดเครื่องหมายที่ใช้ในสมการ Blending

- เรายังสามารถเครื่องหมายที่ใช้รวมสีของ source กับ destination เข้าด้วยกันได้ มันไม่จำเป็นต้องเป็นเครื่องหมายบวก
- เครื่องหมายนี้สามารถกำหนดได้ด้วยคำสั่ง

  glBlendEquation(GLenum mode)

  และ

  glBlendEquationSeparate(GLenum modeRGB,

  GLenum modeAlpha)
- เช่นเคย glBlendEquationSeparate สามารถกำหนดเครื่องหมายของ RGB และ A แยกกันได้

ค่าคงที่ในการกำหนดเครื่องหมายในการทำ blending

ค่าคงที่	การคำนวณสีใหม่
GL_FUNC_ADD	$C_s S + C_d D$
GL_FUNC_SUBTRACT	$C_s S - C_d D$
GL_FUNC_REVERSE_SUBTRACT	$C_d D - C_s S$
GL_MIN	$\min(C_s S, C_d D)$
GL_MAX	$\max(C_s S, C_d D)$
GL_LOGIC_OP	$C_s S \ \text{op} \ C_d D$

GL_LOGIC_OP

- กรณีที่ใช้ GL_LOGIC_OP คำว่า “op” จะถูกแทนด้วยเครื่องหมายทางตรรกศาสตร์ซึ่งสามารถกำหนดได้ด้วยคำสั่ง

  glLogicOp()

ตัวอย่าง: Matting

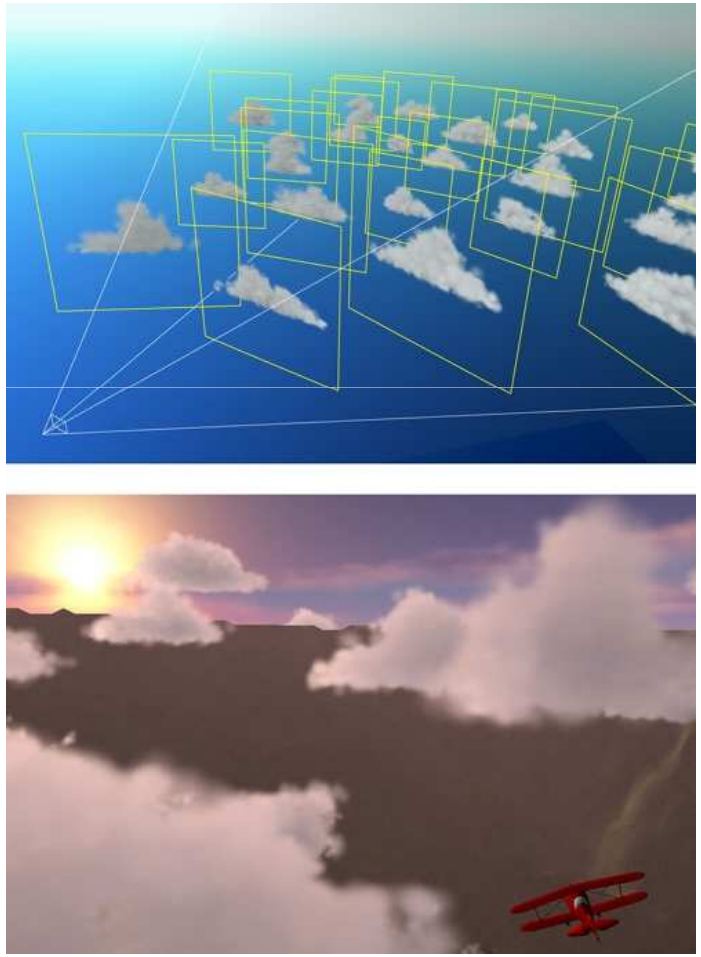
- การนำรูปสองรูปมาซ้อนกัน
- รูปที่อยู่ด้านหน้าบาง pixel อาจมี alpha ไม่เป็น 0 แสดงว่า pixel โปร่งแสง ทำให้เห็นรูปที่อยู่ข้างหลัง
- กรณีนี้ให้ destination blending factor เป็น GL_ONE_MINUS_SRC_ALPHA
- และให้ source blending factor เป็น GL_SRC_ALPHA

ดู demo

ตัวอย่าง: Billboarding

- การสร้างวัตถุคล้ายสามมิติจากภาพ เช่น ต้นไม้ เป็นต้น
- ทำให้ไม่ต้องวางสามเหลี่ยมจำนวนมาก
- ใช้ภาพที่บาง pixel มี alpha เท่ากับ 0
- จะว่าไปก็เหมือนทำ matting ในสามมิติ
- ให้ destination blending factor เป็น GL_ONE_MINUS_SRC_ALPHA
- และ source blending factor เป็น GL_SRC_ALPHA

ตัวอย่าง: Billboarding (ต่อ)



ตัวอย่าง: การรวมรูปสามรูปเข้าด้วยกัน

- ให้ destination blending factor เป็น GL_ONE
- ให้ source blending factor เป็น GL_SRC_ALPHA
- วาดรูปแต่ละรูปด้วย alpha = 1/3

ตัวอย่าง: ปรับสี RGB

- วาดภาพภาพหนึ่งลงบน framebuffer โดยไม่ต้องทำ blending
- หลังจากนั้นใช้ destination blending factor เป็น GL_SRC_COLOR
- และใช้ source blending factor เป็น GL_ZERO
- หลังจากนั้นวาดสีเหลี่ยมสีเดียวทับลงบนภาพนั้น

การวาดวัตถุโปร่งแสงในสามมิติ

- ต้องวาดวัตถุทึบแสดงให้หมดก่อนโดยใช้ depth buffer
- หลังจากนั้นเปิด blending
- แล้ววาดวัตถุโปร่งแสง
- **ข้อควรระวัง 1:** เพื่อให้ได้ภาพที่ถูกต้อง เราต้องวาดวัตถุโปร่งแสงจากด้านหลังไปด้านหน้า

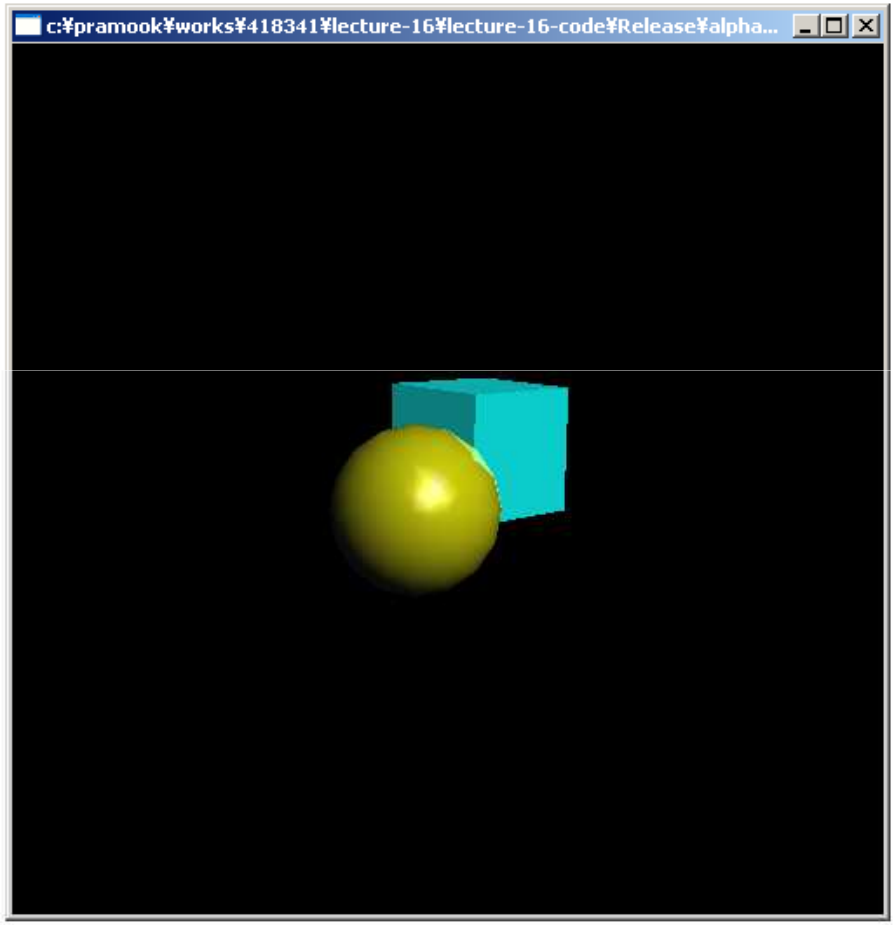
การวาดวัตถุโปร่งแสงในสามมิติ (ต่อ)

- **ข้อควรระวัง 2:** เวลาวาดวัตถุโปร่งแสงจะต้องทำการป้องกันไม่ให้วัตถุโปร่งแสงไปเปลี่ยนแปลง depth buffer
- เราสามารถทำให้ depth buffer ไม่ถูกเขียนทับได้ด้วยการสั่ง

  glDepthMask(GL_FALSE);
- และทำให้มันถูกเขียนทับได้ใหม่อีกครั้ง (เพื่อวาดรูปวัตถุทึบแสงใหม่) ด้วยคำสั่ง

  glDepthMask(GL_TRUE);

ดู demo


c:¥pramook¥works¥418341¥lecture-16¥lecture-16-code¥Release¥alpha...

ดูโค้ด

```
glPushMatrix ();
   glTranslatef (-0.15, -0.15, solidZ);
   glMaterialfv(GL_FRONT, GL_EMISSION, mat_zero);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_solid);
   glCallList (sphereList);
glPopMatrix ();

glPushMatrix ();
   glTranslatef (0.15, 0.15, transparentZ);
   glRotatef (15.0, 1.0, 1.0, 0.0);
   glRotatef (30.0, 0.0, 1.0, 0.0);
   glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent);
   glEnable (GL_BLEND);
   glBlendFunc (GL_SRC_ALPHA, GL_ONE);
   glDepthMask (GL_FALSE);
   glCallList (cubeList);
   glDepthMask (GL_TRUE);
   glDisable (GL_BLEND);
glPopMatrix ();
```

หมอก

- หมอกใน OpenGL ทำให้ pixel มีสีเปลี่ยนไปตามระยะห่างจากตา
- ทำให้เกิด effect เหมือนหมอกจริงๆ ได้
- เวลาใช้หมอกต้องสั่ง

  glEnable(GL_FOG);

  เวลาเลือกใช้สั่ง

  glDisable(GL_FOG);

การคำนวณหมอก

- การคำนวณหมอกต้องมี depth buffer เพื่อวัดระยะห่างจากตา ดังนั้นต้องมี GLUT_DEPTH ใน glutInitDisplay
- การคำนวณหมอกจะทำหลังจาก render รูปขั้นตอนอื่นทั้งหมด
- เวลาคำนวณสีของ pixel เมื่อมีหมอกจะใช้สูตรต่อไปนี้

$$C = f\,C_i + (1-f)\,C_f$$

C คือสีของ pixel หลังจากคำนวณแล้ว

f คือ fog factor ที่จะบอกวิธีคำนวณในสไลด์ต่อไป

C_i คือสีของ pixel ใน frame buffer

C_f คือสีของหมอก

Fog Factor

- เราสามารถกำหนดวิธีการคำนวณ fog factor ได้โดยคำสั่ง glFogi(GL_FOG_MODE, GLenum mode) โดย mode มีค่าได้สามค่าดังต่อไปนี้

  - GL_EXP $f = e^{-(density \cdot z)}$
  - GL_EXP2 $f = e^{-(density \cdot z)^2}$
  - GL_LINEAR $f = \frac{end - z}{end - start}$

การเซตหมอกค่าต่างๆ

- เราสามารถเซตค่าสีของหมอกได้โดยคำสั่ง

  glFogfv(GL_FOG_COLOR, Glfloat *color)

  เช่น

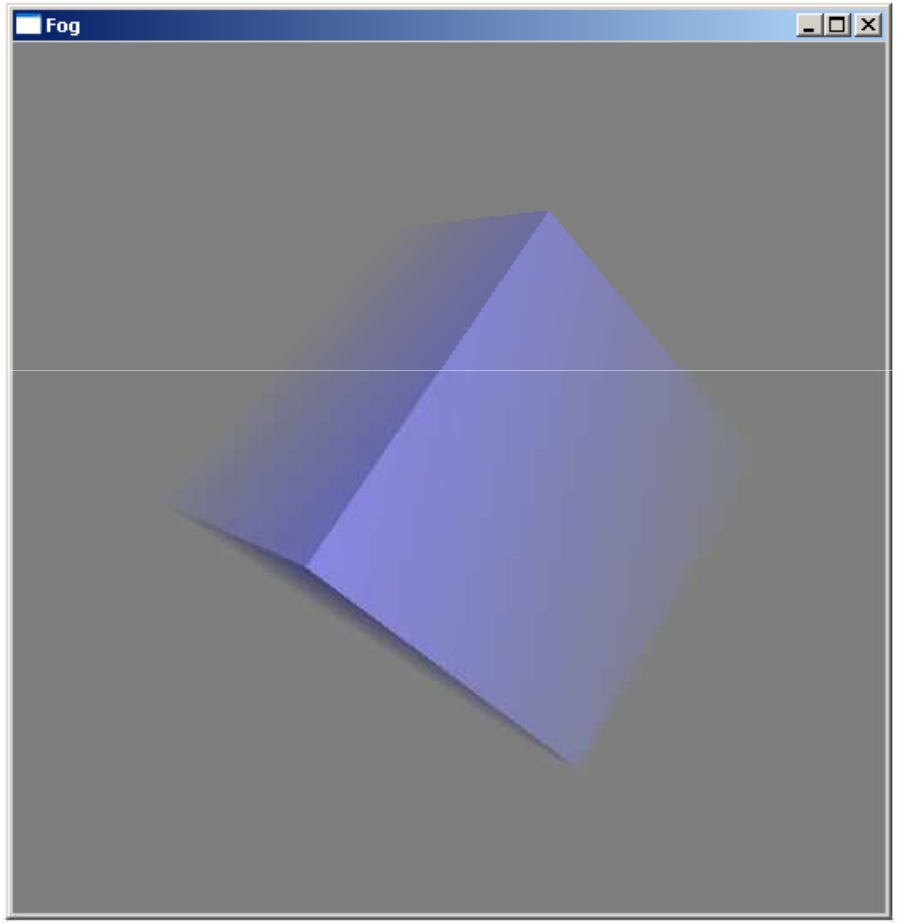
  float fog_color[4] = {0.5f, 0.5f, 0.5f, 1.0f};

  glFogfv(GL_FOG_COLOR, fog_color);

การเซตหมอกค่าต่างๆ (ต่อ)

- เราสามารถเซตค่า density ของหมอก (ใช้ในการคำนวณกรณี GL_EXP และ GL_EXP2) ได้ด้วยคำสั่ง
  glFogf(GL_FOG_DENSITY, GLfloat density)
- เราสามารถเซตค่า start และ end (ใช้ในการคำนวณกรณี GL_LINEAR) ได้ด้วยคำสั่ง
  glFogf(GL_FOG_START, GLfloat start) และ
  glFogf(GL_FOG_END, GLfloat end)
  ตามลำดับ

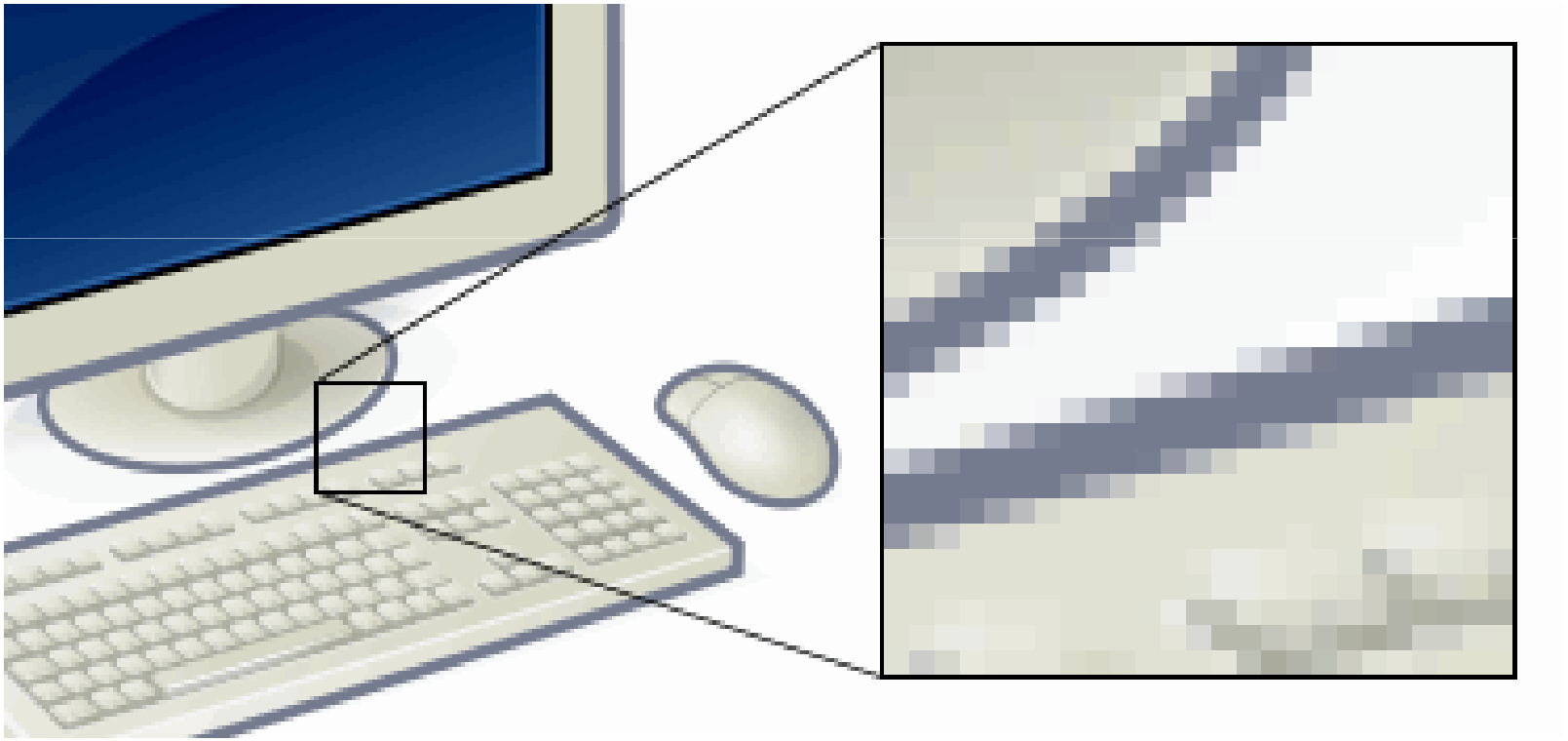
ดู demo



การชักตัวอย่างและการสร้างคืน

ภาพดิจิตอล

- ประกอบด้วย “พิกเซล” เรียงกันเป็นตาราง



รูปจาก http://en.wikipedia.org/wiki/Pixel

พิกเซลคืออะไร?

- พิกเซลไม่ใช่
  - กล่องสี่เหลี่
  - จุดวงกล
  - เส้นขีด

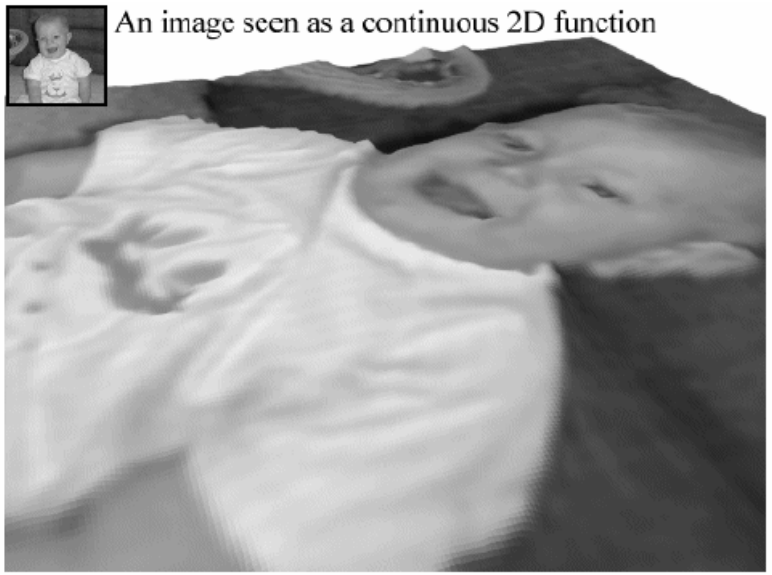
รูปจาก http://en.wikipedia.org/wiki/Pixel

พิกเซลคืออะไร? (ต่อ)

- มันคือตัวเลขสามตัวแทนค่า RGB ณ จุดจุดหนึ่งบนระนาบภาพ
- “จุด” ที่ว่านี้
  - ไม่มีขนาด ความกว้าง ความยาว
  - ไม่มีมิติ
  - มองไม่เห็น
  - มีแต่ “ตำแหน่ง”

ภาพคืออะไร?

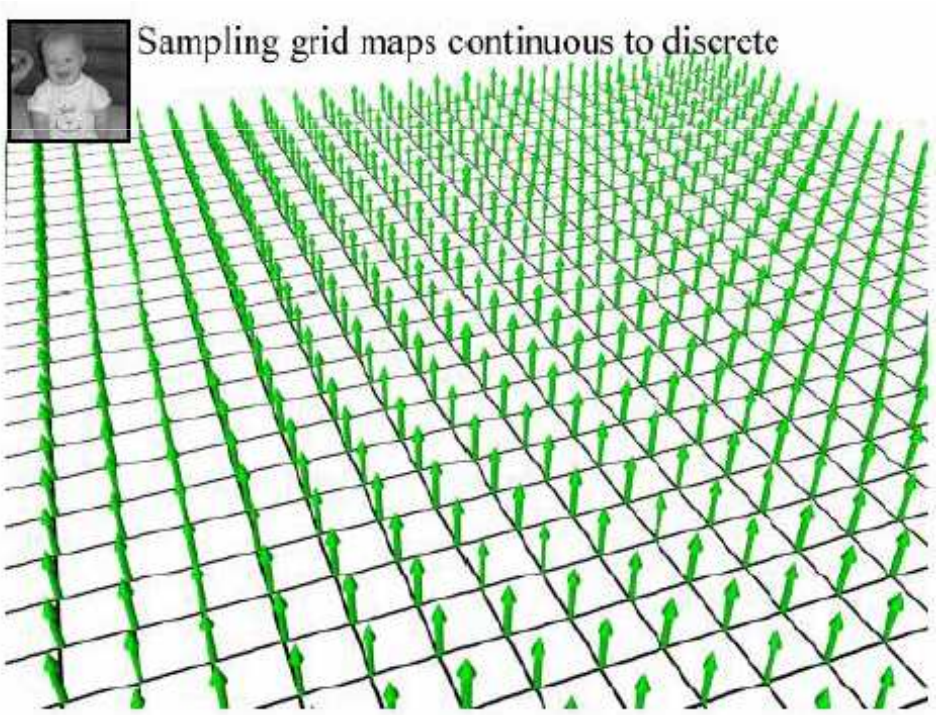
- ฟังก์ชันจากระนาบสองมิติ $\mathbb{R}^2$ ไปยังความเข้มแสง
- รูปภาพเป็นฟังก์ชันต่อเนื่อง
- สามารถมองว่าแต่ละจุดมีความสูงของมัน



Courtesy of Leonard McMillan, Computer Science at the University of North Carolina in Chapel Hill. Used with permission.

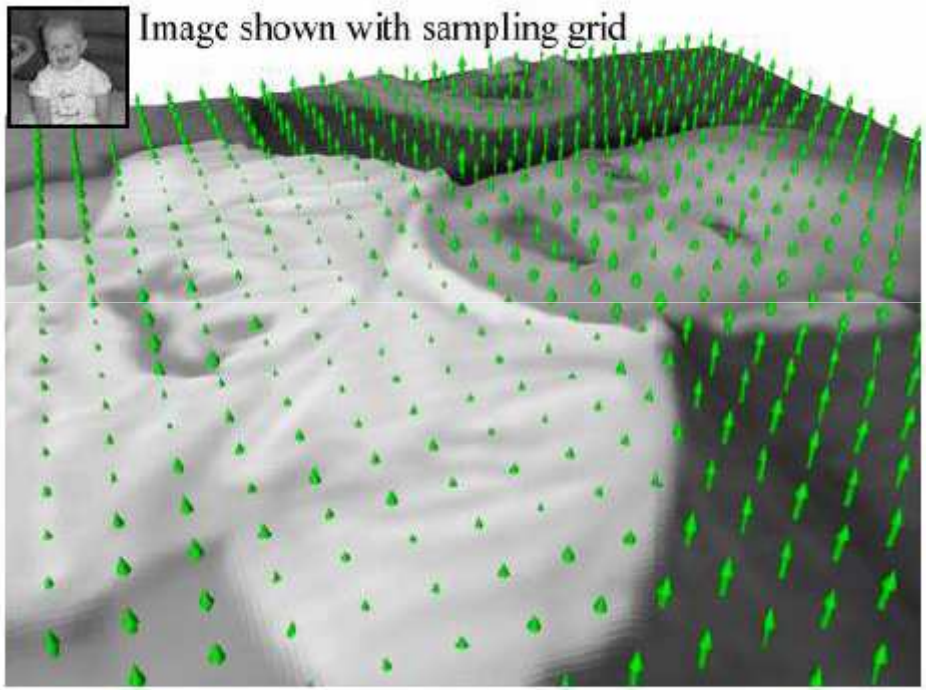
ภาพดิจิตอล

- สร้างโดยการหาค่าของฟังก์ชันภาพ ณ จุดที่เรียงกันเป็นตารางสี่เหลี่ยม
- กระบวนการทำเช่นนี้เรียกว่า **การชักตัวอย่าง** (sampling)



Courtesy of Leonard McMillan, Computer Science at the University of North Carolina in Chapel Hill. Used with permission.

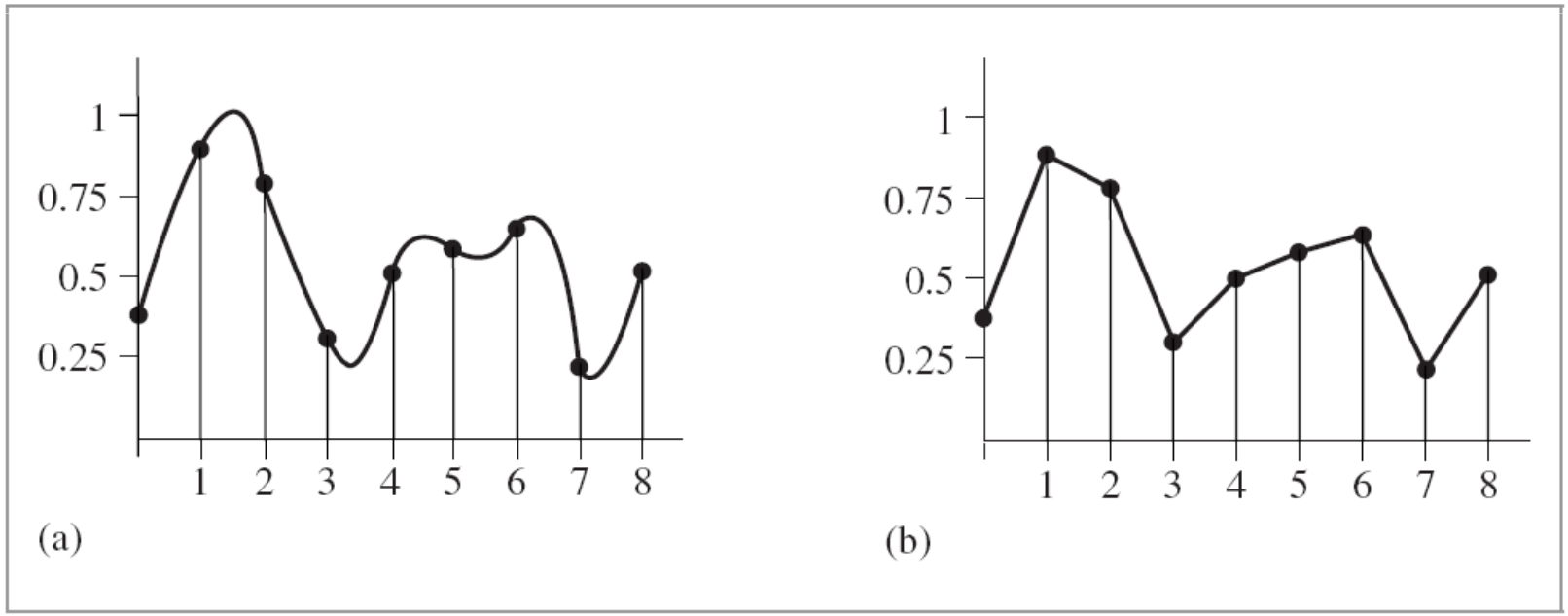
ภาพดิจิตอล (ต่อ)



Courtesy of Leonard McMillan, Computer Science at the University of North Carolina in Chapel Hill. Used with permission.

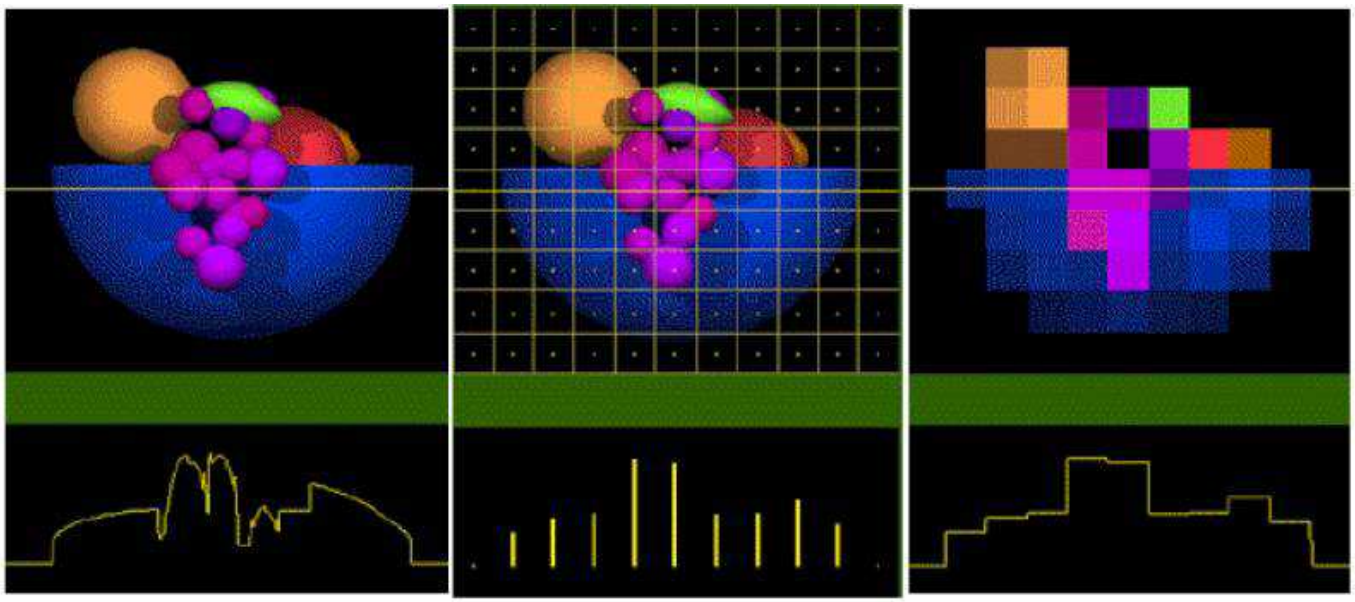
การสร้างคืน

- เมื่อได้ภาพดิจิตอล ซึ่งเป็นตัวอย่างหลายๆ ตัวอย่างที่เราชักมาจากภาพ
- เราต้องทำ**การสร้างคืน** (reconstruction) เพื่อให้ได้ฟังก์ชันต่อเนื่องสองมิติอันใหม่



การสร้างคืน (ต่อ)

- เราต้องการให้ภาพที่เราสร้างคืนเหมือนกับภาพต้นแบบที่สุดเท่าที่จะเป็นไปได้
- ปกติแล้วอุปกรณ์แสดงผลจะสร้างภาพคืนให้เราโดยอัตโนมัติ
- ส่วนมากด้วยการวาดสีของตัวอย่างที่ชักมาเป็นสี่เหลี่ยม

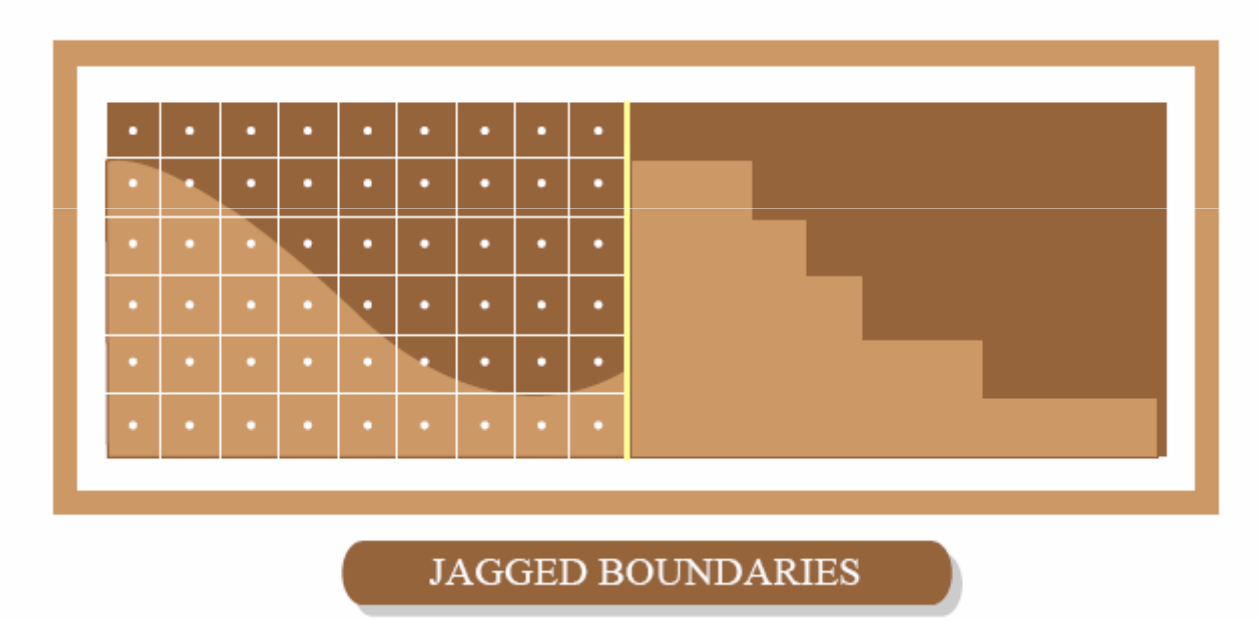


เอเลียสซิง

- แต่การสร้างคืนด้วยการวาดเป็นสี่เหลี่ยมนี่เองทำให้เกิดปัญหาตามมาหลายอย่าง เรียกรวมๆ ว่า **เอเลียสซิง** (aliasing)
- ทำให้ภาพออกมาดูไม่ดี จะดูไม่ดีมากโดยเฉพาะเวลาทำภาพเคลื่อนไหว

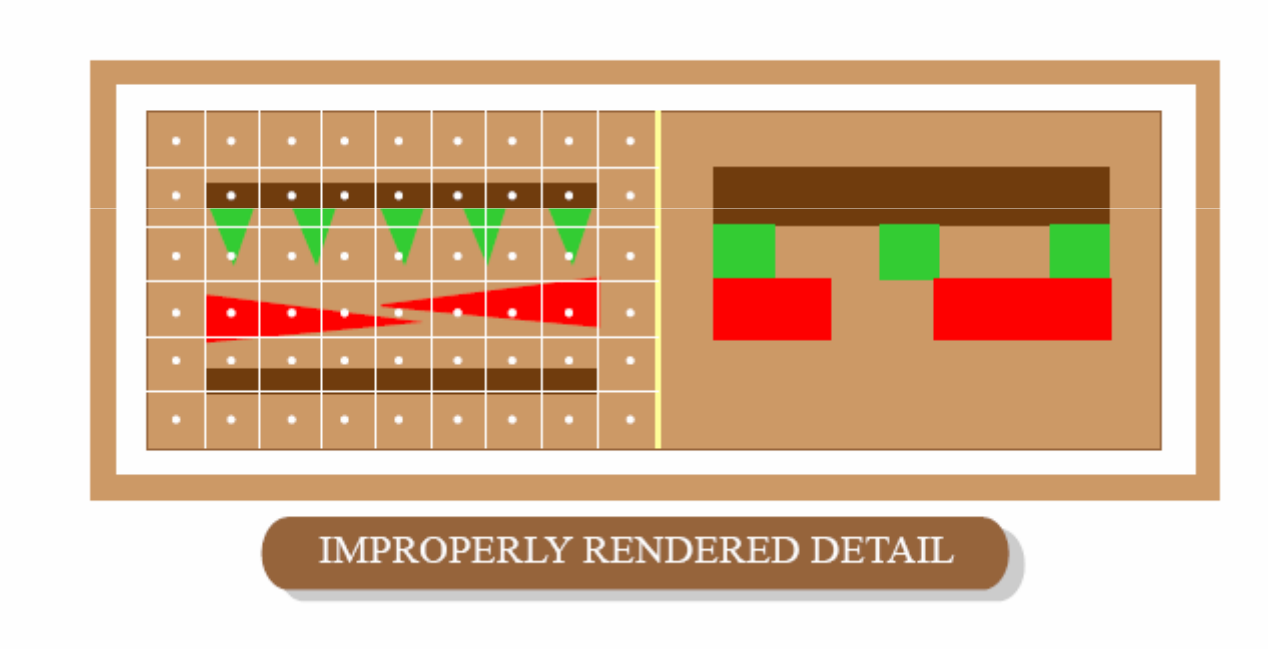
เอเลียสซิง (ต่อ)

- ขอบเป็นหยักๆ



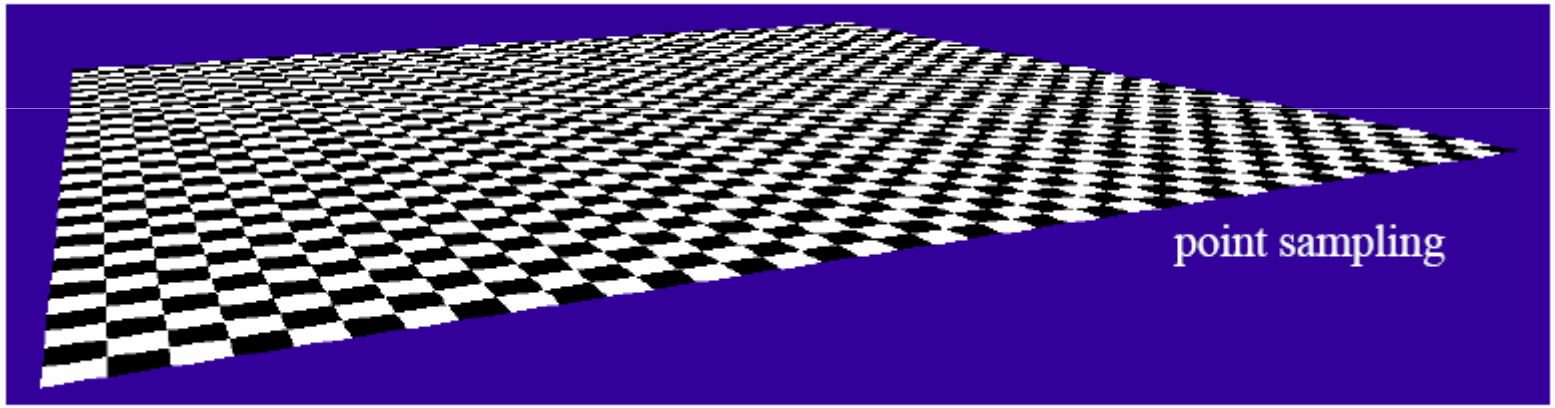
เอเลียสซิง (ต่อ)

- รายละเอียดหายหรือถูกขยายใหญ่เกินไป



เอเลียสซิง (ต่อ)

- จิตรกรรมฝาผนังดูไม่สวย



การแปลงฟูเรียร์

โดเมนความถี่

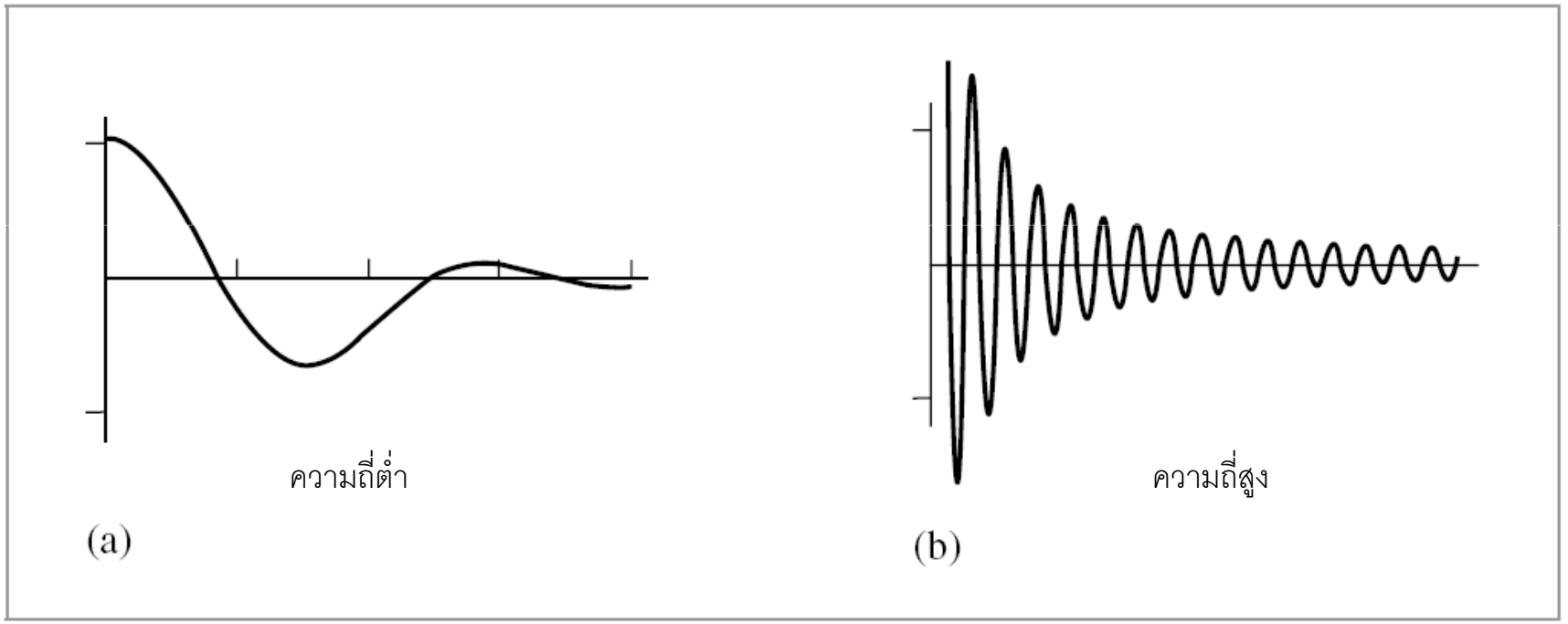
- ฟังก์ชันเกือบทุกฟังก์ชันสามารถเขียนอยู่ในรูปผลบวกของฟังก์ชันไซน์หรือโคไซน์ได้
- ถ้าเขียนในรูปแบบนี้ เราเรียกว่ามันอยู่ใน **โดเมนความถี่** (frequency domain)
- ถ้าเขียนแบบปกติจะเรียกว่าอยู่ใน **โดเมนปริภูมิ** (spatial domain)
- นึกถึงเสียง ซึ่งเป็นฟังก์ชันการเคลื่อนที่ของลำโพง
  - เราสามารถเอาเสียงไปใส่ spectrum analyzer
  - มันจะบอกว่าเสียงความถี่เท่านี้มีความดังเท่าไหร่

โดเมนความถี่ (ต่อ)

- ฟังก์ชันที่เปลี่ยนแปลงเร็ว มีความถี่สูง
- ฟังก์ชันที่เปลี่ยนแปลงช้า มีความถี่ต่ำ

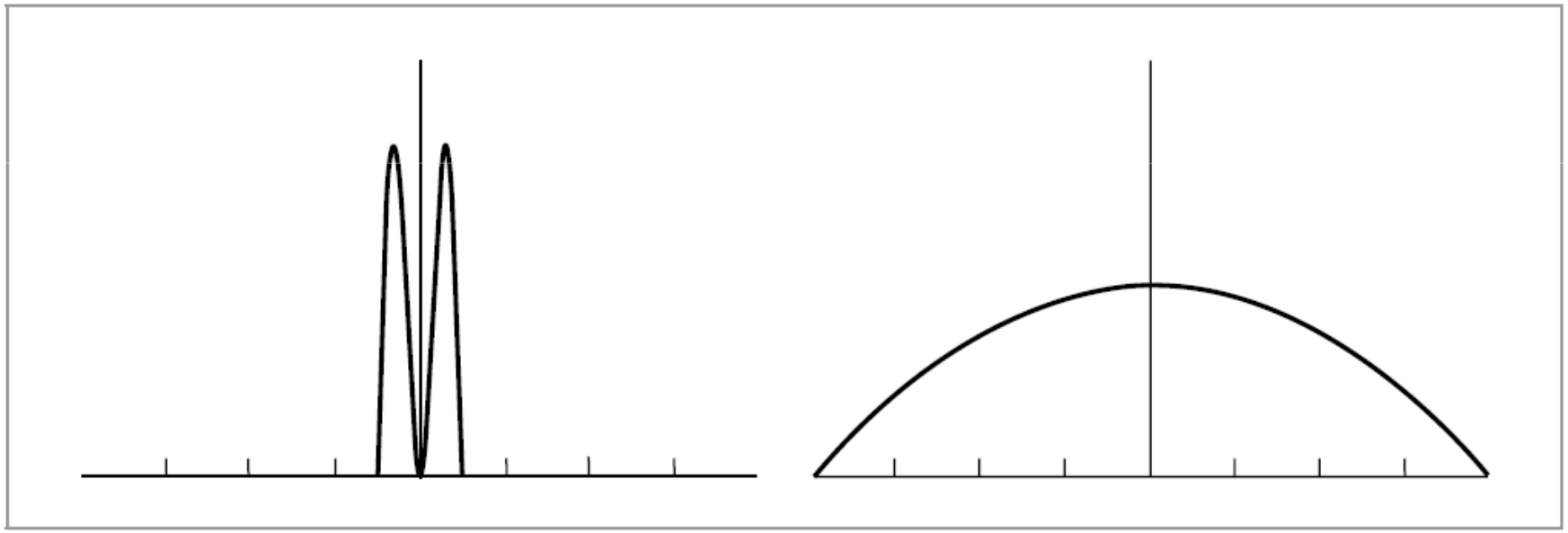
โดเมนความถี่ (ต่อ)

- ฟังก์ชันในโดเมนปริภูมิ



โดเมนความถี่ (ต่อ)

- เมื่ออยู่ในโดเมนความถี่



ขอบเขตแคบ

ขอบเขตกว้าง

การแปลงฟูเรียร์ (ต่อ)

- เรามีฟังก์ชัน $f(x)$ ในโดเมนปริภูมิ
- การแปลงฟูเรียร์ทำให้เราได้ฟังก์ชัน $F(\omega)$ ในโดเมนความถี่
- ความสัมพันธ์ระหว่างทั้งสองฟังก์ชัน:

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

$$f(x) = \int_{-\infty}^{\infty} F(\omega) e^{i2\pi\omega x} d\omega$$

การแปลงฟูเรียร์ (ต่อ)

- เราใช้สัญลักษณ์ $\mathcal{F}\{f(x)\}$ แทนการแปลงฟูเรียร์ของ $f(x)$

$$\mathcal{F}\{f(x)\} = F(\omega) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi\omega x}dx$$

- การแปลงฟูเรียร์เป็นฟังก์ชันเชิงเส้น

$$\mathcal{F}\{f(x) + g(x)\} = \mathcal{F}\{f(x)\} + \mathcal{F}\{g(x)\}$$
$$\mathcal{F}\{cf(x)\} = c\mathcal{F}\{f(x)\}$$

การคูณและการแปลงฟูเรียร์

- ถ้าเราทำการแปลงฟูเรียร์ของผลคูณของฟังก์ชันสองฟังก์ชัน
- เราจะได้**คอนโวลูชัน**ของการแปลงฟูเรียร์ของฟังก์ชันสองฟังก์ชันนั้น

$$\mathcal{F}\{f(x)g(x)\} = F(\omega) \otimes G(\omega)$$

- ในทำนองเดียวกัน

$$\mathcal{F}\{f(x) \otimes g(x)\} = F(\omega)G(\omega)$$

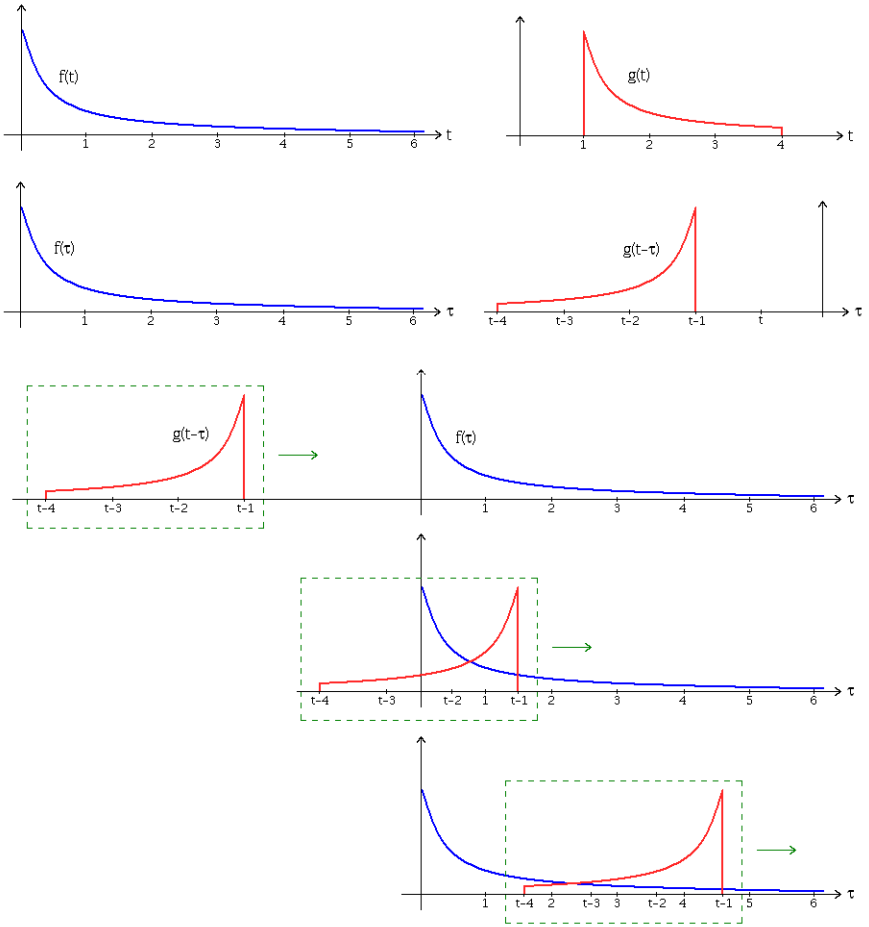
คอนโวลูชัน

- มีนิยามดังนี้

$$f(x) \otimes g(x) = (f \otimes g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$

- มีความหมายเหมือนกับการ “เลื่อน” $g(x)$ ไปคูณกับ $f(x)$ ทีละค่าของ x แล้วเอาผลลัพธ์ทั้งหมดมาบวกกัน

คอนโวลูชัน (ต่อ)


f(t)
t
1
2
3
4
5
6
g(t)
t
1
2
3
4
f(τ)
τ
1
2
3
4
5
6
g(t−τ)
τ
t−4
t−3
t−2
t−1
t
g(t−τ)
t−4
t−3
t−2
t−1
f(τ)
τ
1
2
3
4
5
6
t−4
t−3
t−2
1
t−1
2
3
4
5
6
τ
1
t−4
2
t−3
3
t−2
4
t−1
5
6
τ

ทฤษฎีการชักตัวอย่าง

ดิแรกเดลตาฟังก์ชัน

- สัญลักษณ์ $\delta(x)$
- ฟังก์ชันที่ทำคอนโวลูชันกับอะไรก็ได้ค่าของฟังก์ชันนั้น

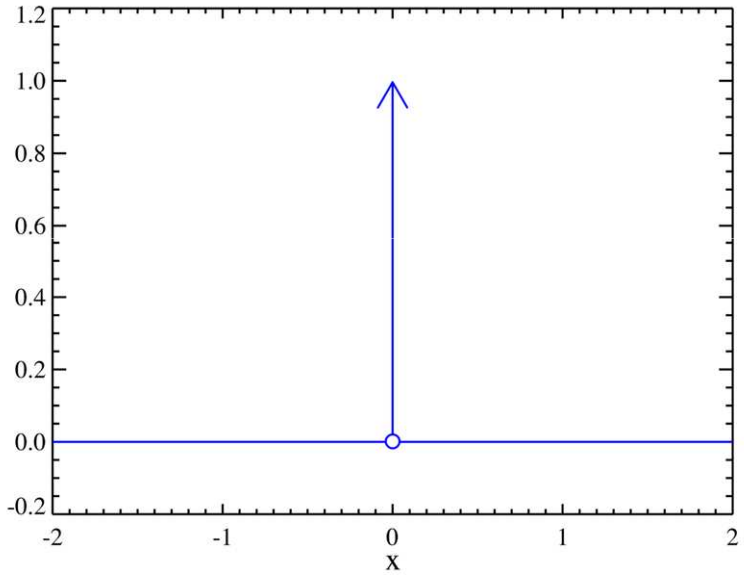
$$f(x) \otimes \delta(x) = f(x)$$

- ถ้าเลื่อน $\delta(x)$ แล้ว $f(x)$ ก็จะเลื่อนตามไปด้วย

$$f(x) \otimes \delta(x - t) = f(x - t)$$

ดิแรกเดลตาฟังก์ชัน (ต่อ)

- หน้าตาของมัน

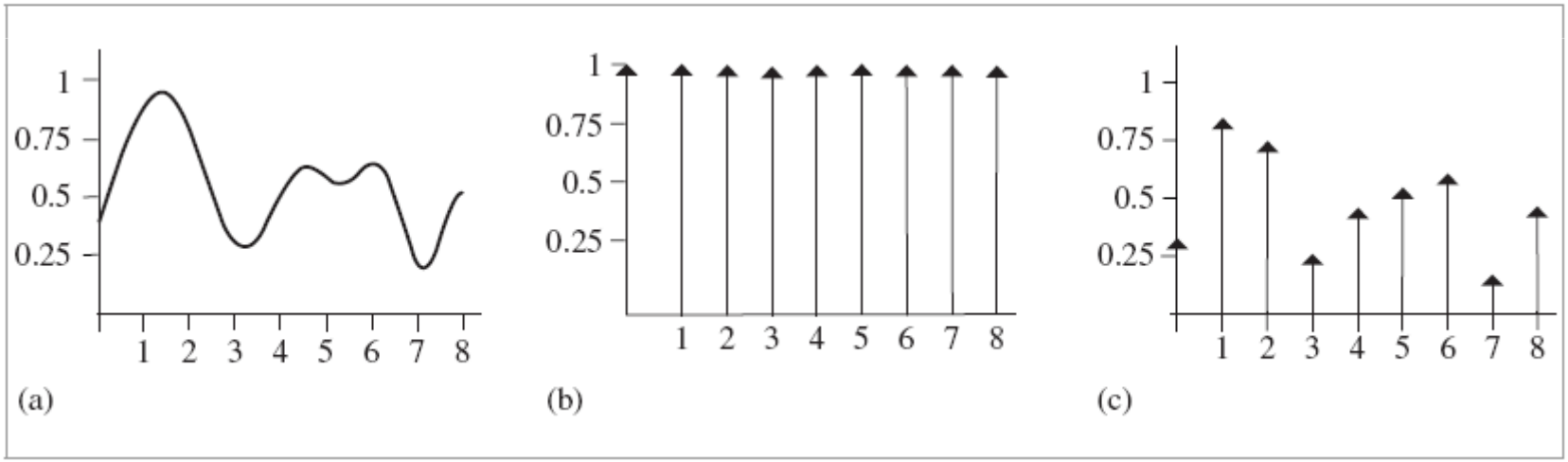


- หาค่าจริงไม่ได้ แต่รู้ว่า **integrate** แล้วได้ **1**

$$\int_{-\infty}^{\infty} \delta(x)\, dx = 1$$

การชักตัวอย่าง

- คือการเอาฟังก์ชันภาพมาคูณกับดิเรกเดลตาฟังก์ชันที่ว่างไว้ห่างเป็นระยะเท่าๆ กัน



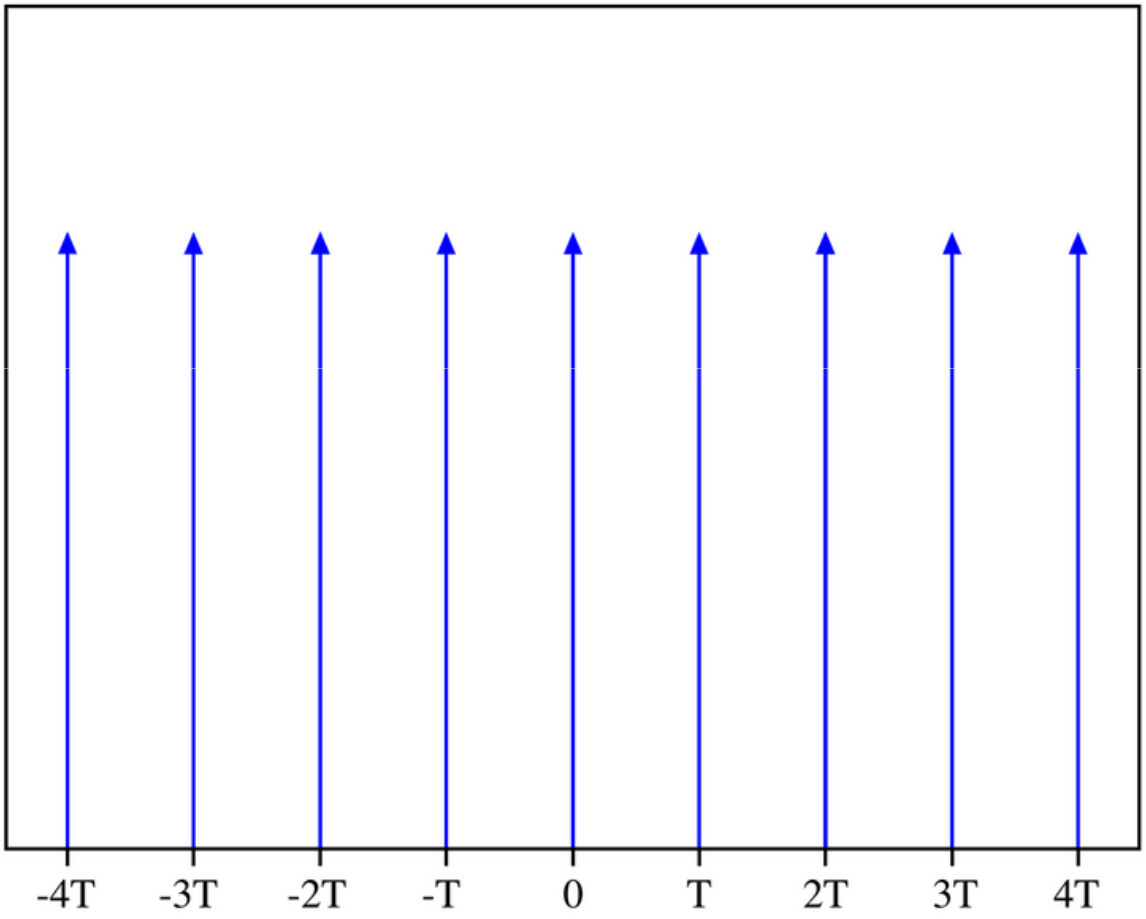
ชาฮ์ฟังก์ชัน

- ดิเรกเดลตาฟังก์ชันที่เอามาเรียงห่างกันเป็นระยะเท่าๆ กันมีชื่อเรียกว่า **ชาฮ์ฟังก์ชัน** (Shah) หรือ **หวีของดิแรก** (Dirac comb)
- สัญลักษณ์ $\mathrm{III}_T(x)$

$$\mathrm{III}_T(x) = T \sum_{k=-\infty}^{\infty} \delta(x - kT)$$

- โดยค่า T คือ “คาบ” ของฟังก์ชัน

ชาฮ์ฟังก์ชัน (ต่อ)



ชาฮ์ฟังก์ชัน (ต่อ)

- เมื่อนำชาฮ์ฟังก์ชันที่มีคาบ T ไปทำการแปลงฟูเรียร์ จะได้ชาฮ์ฟังก์ชันที่มีคาบ $1/T$

$$\mathcal{F}\{\mathrm{III}_T(x)\} = \frac{1}{T}\sum_{k=-\infty}^{\infty}\delta(\omega - k/T) = \mathrm{III}_{1/T}(\omega)$$

การชักตัวอย่าง (อีกครั้ง)

- ดังนั้นการชักตัวอย่างทุกๆ คาบ T สามารถเขียนเป็นประโยคสัญลักษณ์ได้ว่า

$$f(x)\mathrm{III}_T(x)$$

- การแปลงฟูเรียร์ของฟังก์ชันข้างบนคือ

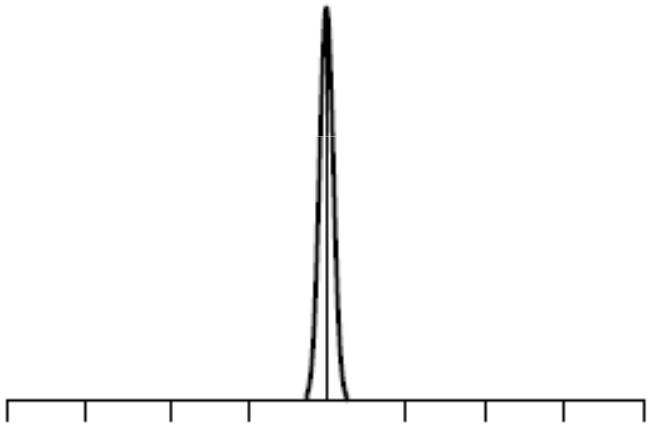
$$\mathcal{F}\{f(x)\mathrm{III}_T(x)\} = F(\omega) \otimes \mathrm{III}_{1/T}(\omega)$$

การชักตัวอย่าง (อีกครั้ง)

- เราทำการคูณ $f(x)$ กับ $\mathrm{III}_T(x)$ ในโดเมนปริภูมิ
- เราจะลองเขียนฟังก์ชันและผลลัพธ์ที่เกิดขึ้นกับมันของการคูณในโดเมนความถี่ดู
- ย้ำว่าเราไม่ได้ต้องแปลงฟังก์ชันเหล่านี้ให้อยู่ในโดเมนความถี่จริงๆ ฟังก์ชันทุกฟังก์ชันเขียนอยู่ในโดเมนความถี่ได้อยู่แล้ว เราแค่ดูเฉยๆว่าถ้าเขียนอยู่ในโดเมนความถี่แล้วเกิดอะไรขึ้น

การชักตัวอย่าง (ต่อ)

- สมมติว่าฟังก์ชัน $f(x)$ ในโดเมนความถี่คือ $F(\omega)$
- และให้ $F(\omega)$ มีหน้าตาเช่นนี้

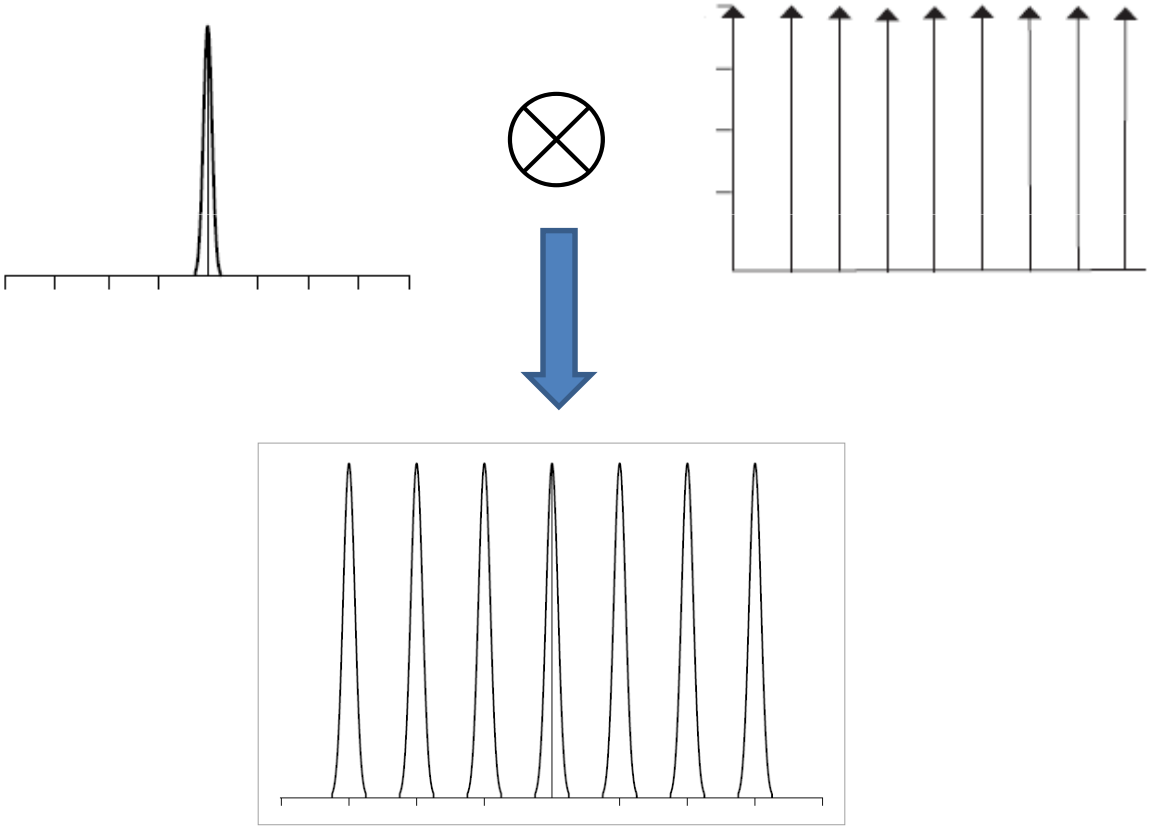


- การคูณ $f(x)$ กับ $\mathrm{III}_T(x)$ ในโดเมนปริภูมิ

  มีค่าเท่ากับการเอา $F(\omega)$ กับ $\mathrm{III}_{1/T}(\omega)$ มาทำคอนโวลูชันกัน

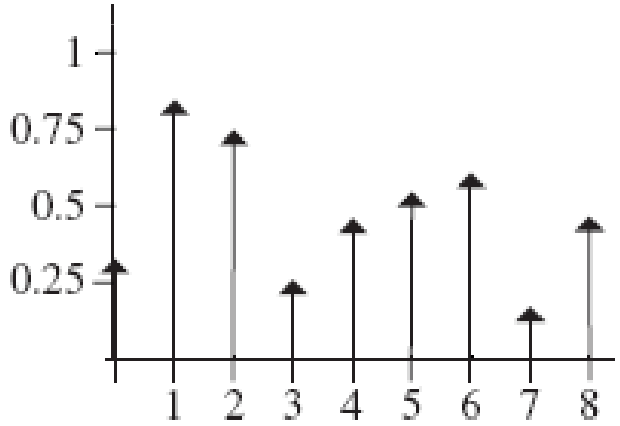
การชักตัวอย่าง (ต่อ)

- เมื่อเอามันมาทำคอนโวลูชันกับ $\mathrm{III}_{1/T}(\omega)$ จะได้



การชักตัวอย่าง (ต่อ)

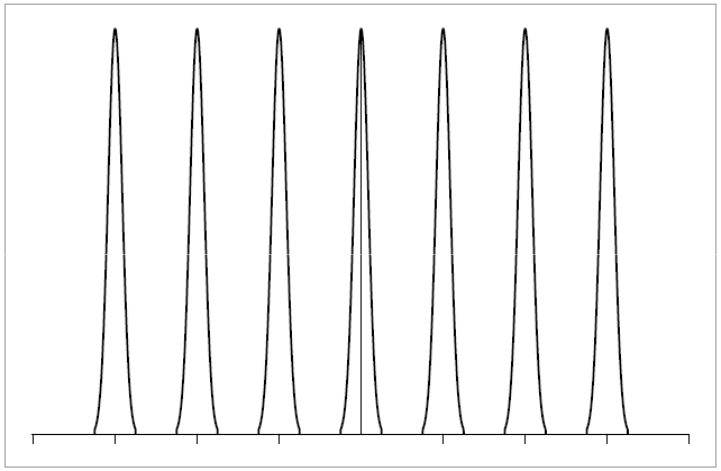
- แล้วตอนนี้เราได้อะไรบ้าง
- เรารู้ว่าหลังจากการชักตัวอย่างแล้ว เราจะได้ฟังก์ชัน $f(x)\mathrm{III}_T(x)$ ที่มีหน้าตาคล้ายๆ รูปข้างล่างในโดเมนปริภูมิ



- กล่าวคือมันเป็นดิแรกเดลตาฟังก์ชันความสูงต่ำต่างกันเรียงกันเป็นช่วงๆ

การชักตัวอย่าง (ต่อ)

- แต่ถ้าเราเขียนฟังก์ชันดิแรกเดลตาสูงๆ ต่ำในโดเมนความถี่ เราจะได้



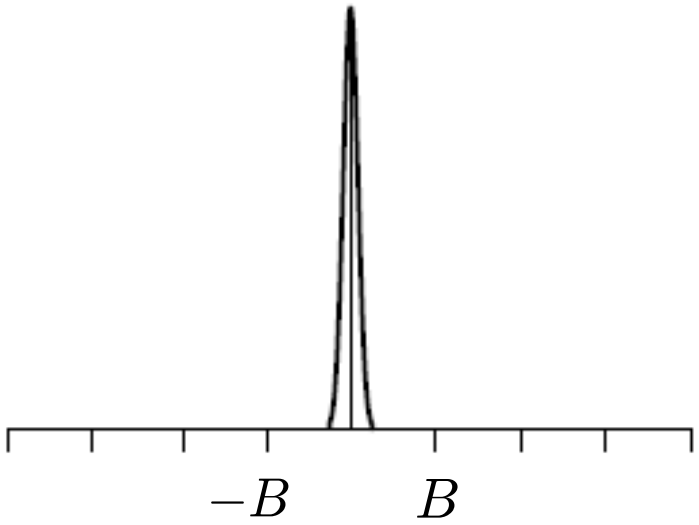
ซึ่งมีลักษณะเหมือนเอาฟังก์ชันต้นฉบับ $F(\omega)$ ในโดเมนความถี่มาถ่ายเอกสารแล้วแปะเป็นช่วงๆ ระยะห่างเท่ากัน

การชักตัวอย่าง (ต่อ)

- ข้อสังเกต:
  - ดิแรกเดลตาฟังก์ชันแต่ละตัวในโดเมนปริภูมิจะห่างกันเป็นระยะ T
  - แต่จุดศูนย์กลางของฟังก์ชัน $F(\omega)$ แต่ละตัวในโดเมนความถี่จะห่างกันเป็นระยะ $1/T$
  - ยิ่งเราสุ่มถี่ขึ้นเท่าไหร่ ก็อปปี้ของฟังก์ชัน $F(\omega)$ ก็จะห่างขึ้นไปเรื่อยๆ

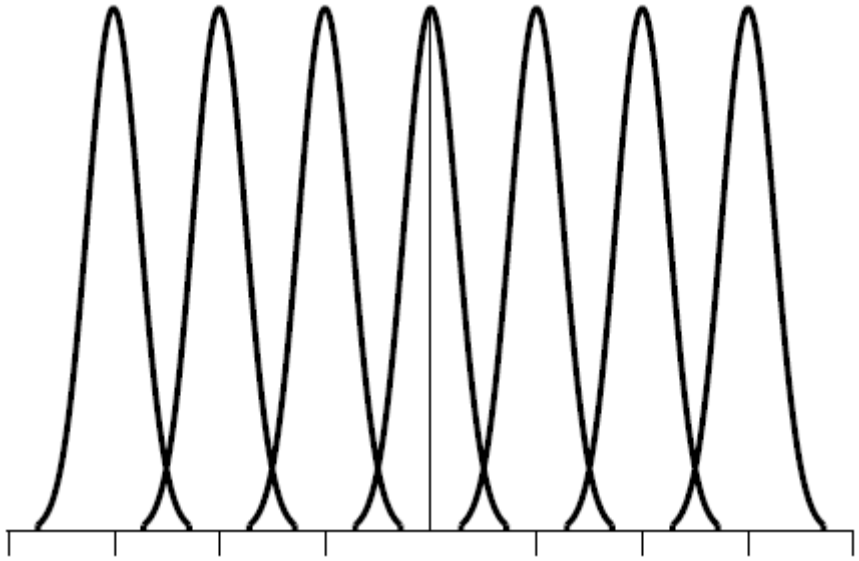
การชักตัวอย่าง (ต่อ)

- เรากล่าวว่าฟังก์ชัน $f(x)$ เป็นฟังก์ชัน**ความถี่จำกัด** (band limit) ถ้าเราเขียนมันอยู่ในโดเมนความถี่ $F(\omega)$ แล้วมันจะมีค่าเป็น $F(\omega)$ ถ้า ω มีค่าสูงถึงระดับหนึ่ง
- กล่าวคือมี B (ย่อมาจากคำว่า band)ที่ทำให้ $F(\omega) = 0$ ถ้า $|\omega| > B$



การชักตัวอย่าง (ต่อ)

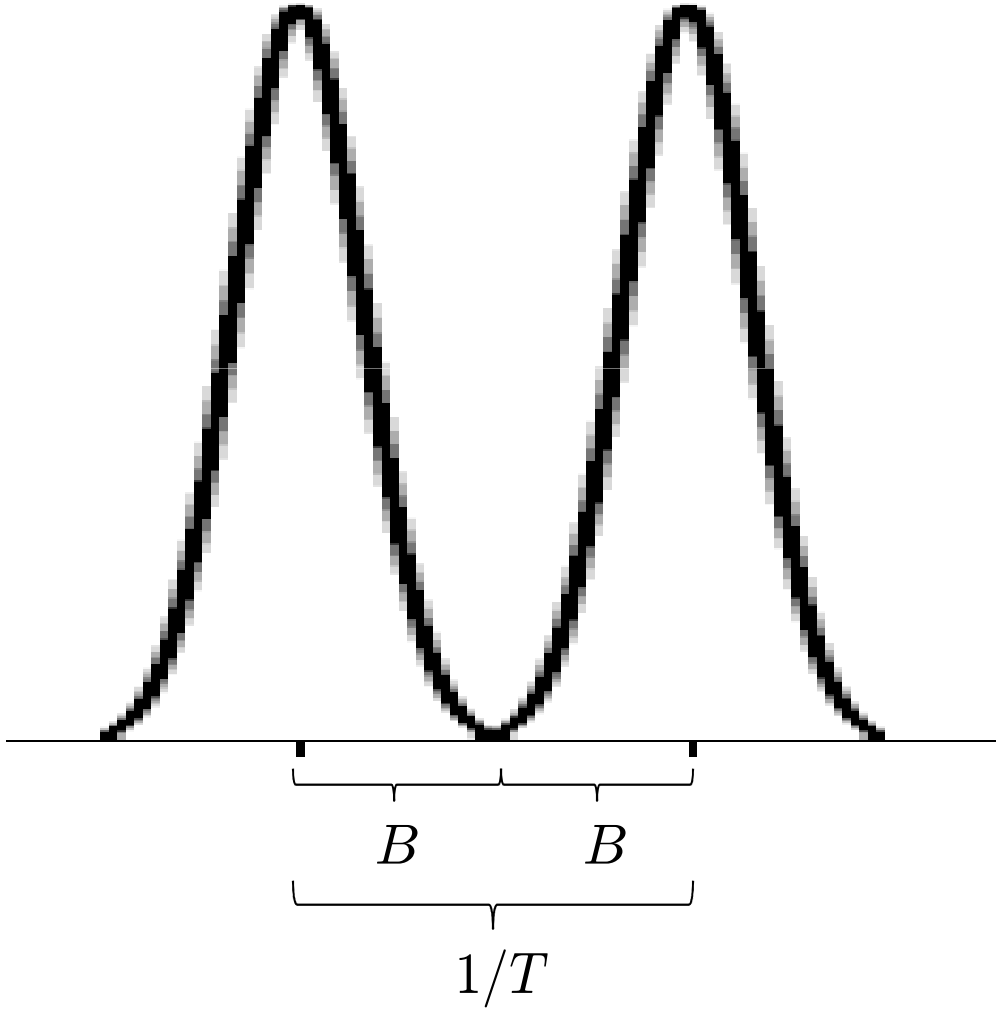
- ถ้า $f(x)$ มีความถี่จำกัด และถ้า T มีค่าน้อยพอ ก็อปปี้ของ $F(\omega)$ จะอยู่ห่างกันและไม่ซ้อนทับกัน
- แต่ถ้า T มีค่ามากเกินไป (เราชักตัวอย่างไม่ถี่พอ) ก็อปปี้ของ $F(\omega)$ อาจซ้อนทับกันได้



การชักตัวอย่าง (ต่อ)

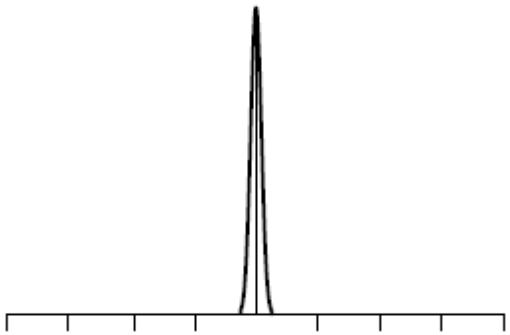
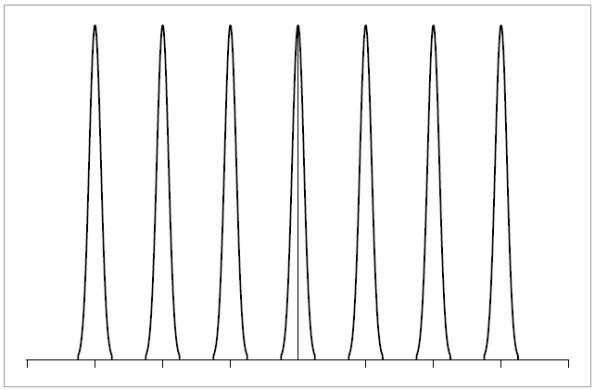
- แล้วต้องเลือก T ให้มีค่าเท่าไหร่ ถึงจะได้ก็อปปี้ที่ไม่ซ้อนทับกัน?
- คำตอบคือต้องเลือกให้ $1/T > 2B$
- นั่นคือต้องสุ่มให้ถี่กว่าความถี่สูงสุดของ $f(x)$ สองเท่า
- ความถี่นี้เรียกว่า **ความถี่ไนควิสท์** (Nyquist Frequency)

การชักตัวอย่าง (ต่อ)



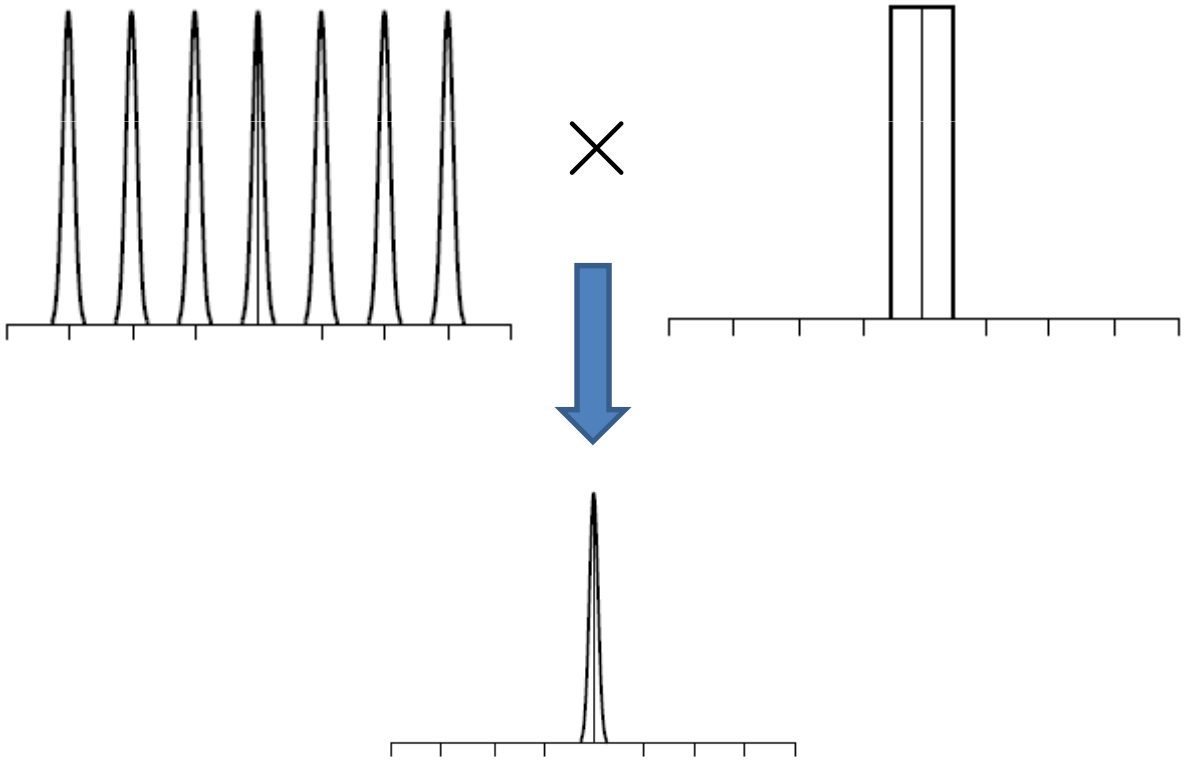
การสร้างกลับ

- ในการสร้างกลับ เราต้องการเอาฟังก์ชันที่ชักตัวอย่างมาแล้วมาทำอะไรสักอย่างให้ได้ฟังก์ชันที่คล้ายฟังก์ชันต้นฉบับมากที่สุด
- สมมติว่าฟังก์ชันต้นฉบับมีความถี่จำกัด และเราตัวอย่างมันที่ความถี่สูงกว่าความถี่ไนควิสท์ ปัญหาของเราคือ

ทำอย่างไรจะได้  จาก  ?

การสร้างกลับ (ต่อ)

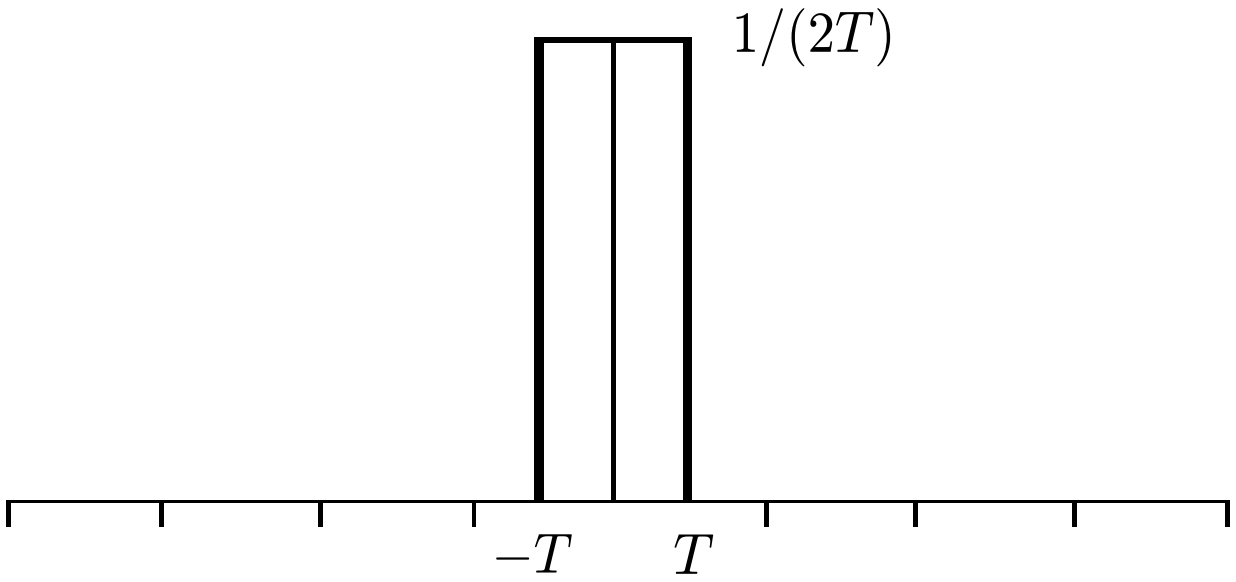
- ต้องลบส่วนเกินที่อยู่รอบนอกออกไป
- คูณด้วย “กล่อง” ในโดเมนความถี่



ฟังก์ชันกล่อง

- สัญลักษณ์ $\Pi_T(\omega)$

$$\Pi_T(\omega) = \begin{cases} 1/(2T) & |\omega| < T \\ 0 & \text{otherwise} \end{cases}$$



ซิงค์ฟังก์ชัน

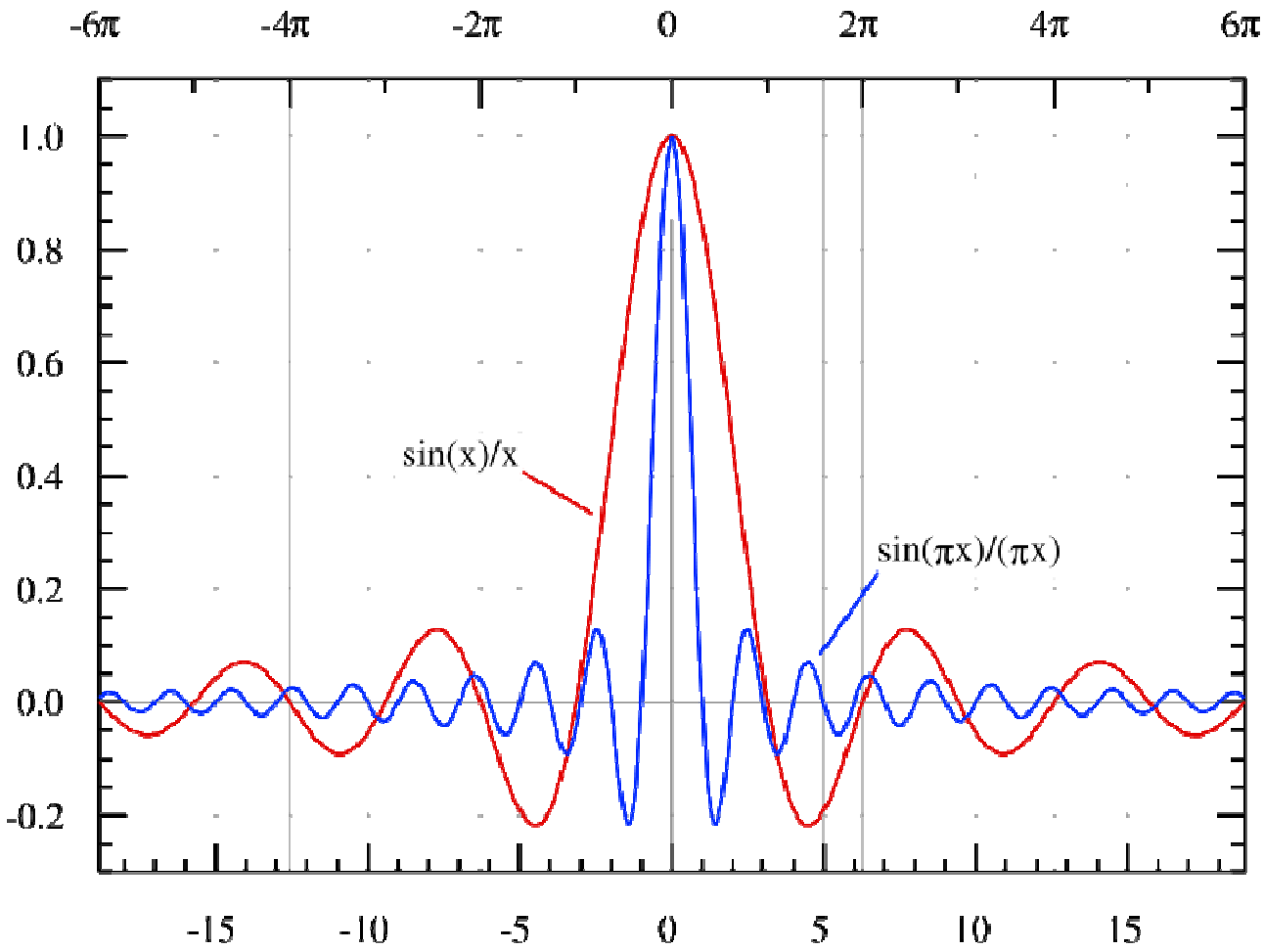
- ฟังก์ชันกล่องที่ว่ามานั้นอยู่ในโดเมนความถี่ ถ้าทำให้อยู่ในโดเมนปริภูมิแล้วจะมีหน้าตาเป็นอย่างไร?

$$\mathcal{F}^{-1}\{\Pi_T(x)\} = \operatorname{sinc}(Tx)$$

- โดยที่

$$\operatorname{sinc}(x) = \begin{cases} 1, & x = 0 \\ \sin x/x, & \text{otherwise} \end{cases}$$

การสร้างกลับ (ต่อ)



การสร้างกลับ (ต่อ)

- เขียนการสร้างกลับจากการชักตัวอย่างเป็นประโยคสัญลักษณ์ในโดเมนความถี่ได้ว่า

$$(F(\omega) \otimes III_{1/T}(\omega))\Pi_T(\omega)$$

- เมื่อเขียนอยู่ในโดเมนปริภูมิ จะได้เป็น

$$(f(x)III_T(x)) \otimes \mathrm{sinc}(Tx)$$

- ดังนั้น การสร้างกลับคือการเอาฟังก์ชันที่ชักตัวอย่างมาแล้วมาทำคอนโวลูชันกับซิงค์ฟังก์ชัน

การสร้างกลับ (ต่อ)

- ถ้า $f(x)$ มีความถี่จำกัดและเราชักตัวอย่างด้วยความถี่สูงกว่าความถี่ไนควิสท์ของมัน เราจะสามารถสร้างฟังก์ชัน $f(x)$ กลับคืนได้ เนื่องจาก

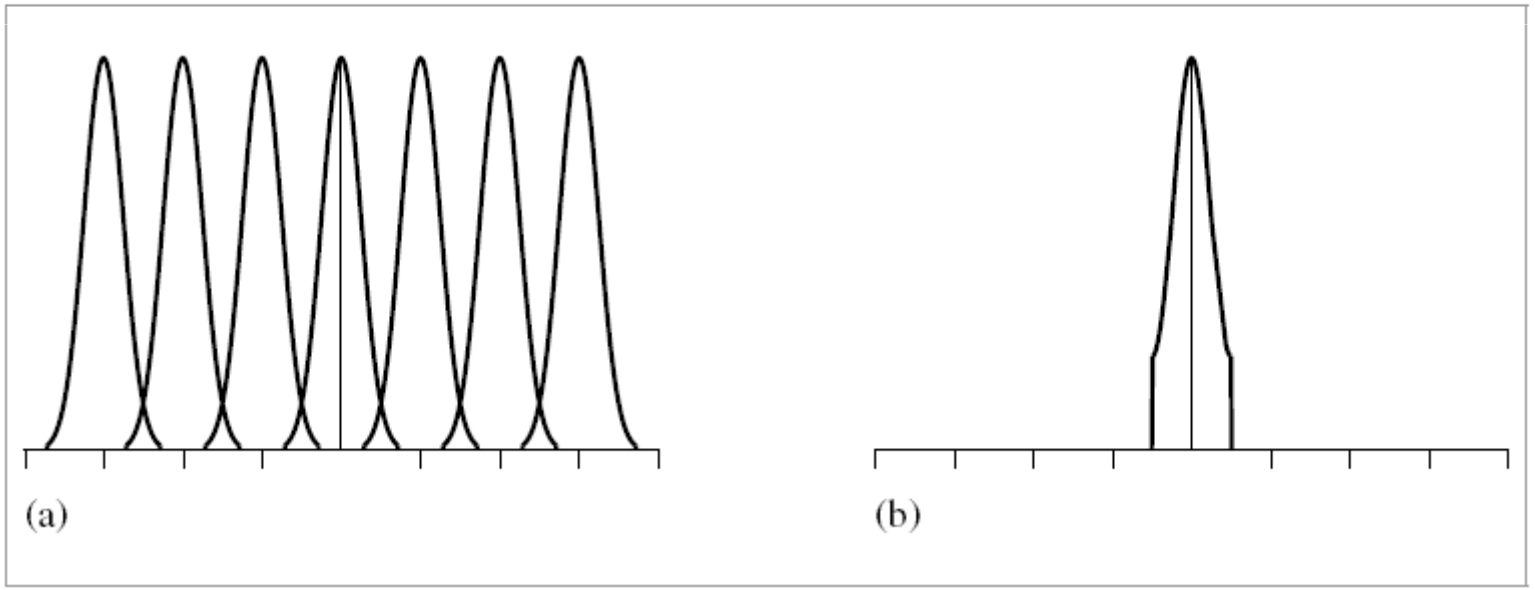
$$(F(\omega) \otimes III_{1/T}(\omega))\Pi_T(\omega) = F(\omega)$$

ดังนั้น

$$(f(x)III_T(x)) \otimes \mathrm{sinc}(Tx) = f(x)$$

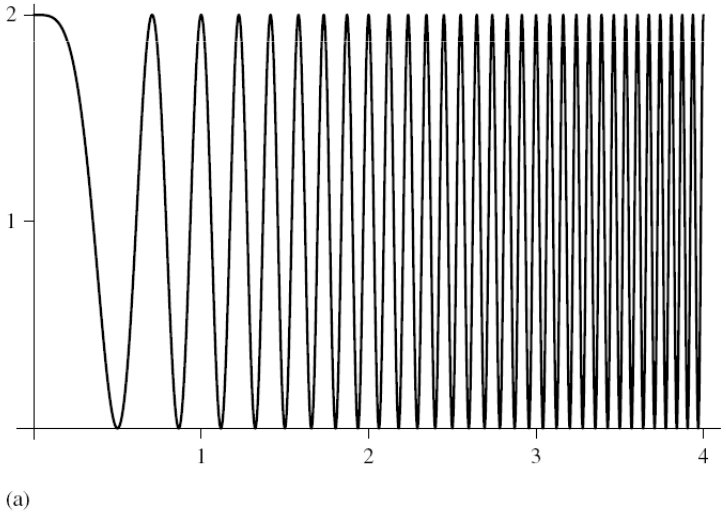
การสร้างกลับ (ต่อ)

- แต่ถ้าเราชักตัวอย่างด้วยความถี่ต่ำกว่าความถี่ไนควิสท์ ก็อปปี้ของ $F(\omega)$ จะซ้อนทับกัน และเมื่อคูณด้วยฟังก์ชันกล่องจะได้ฟังก์ชันอีกอันหนึ่งที่ไม่เหมือน $F(\omega)$ ทุกประการ

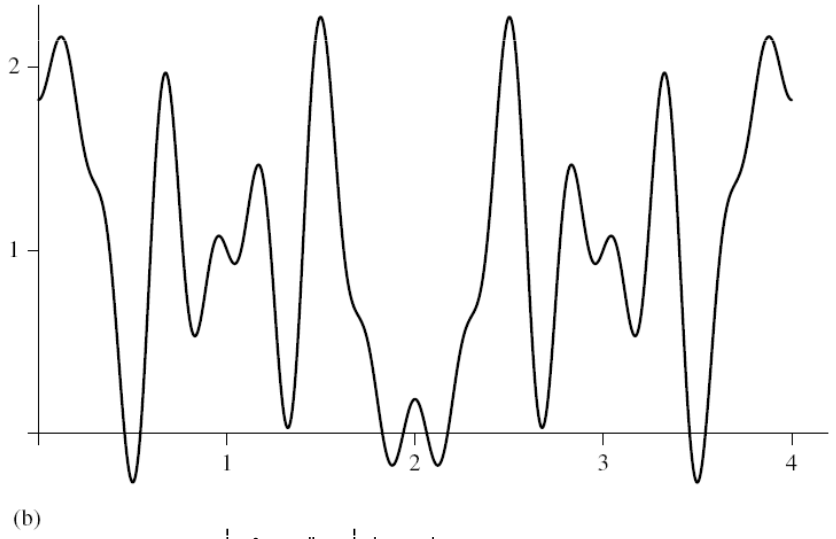


การสร้างกลับ (ต่อ)

- ความไม่เหมือนกันทุกประการที่เกิดจากการชักตัวอย่างด้วยความถี่ไม่มากพอนี่เองที่เป็นสาเหตุของเอเลียสซิงทั้งหมด



ฟังก์ชันต้นแบบ



ฟังก์ชันที่สร้างคืนที่มีเอเลียสซิง

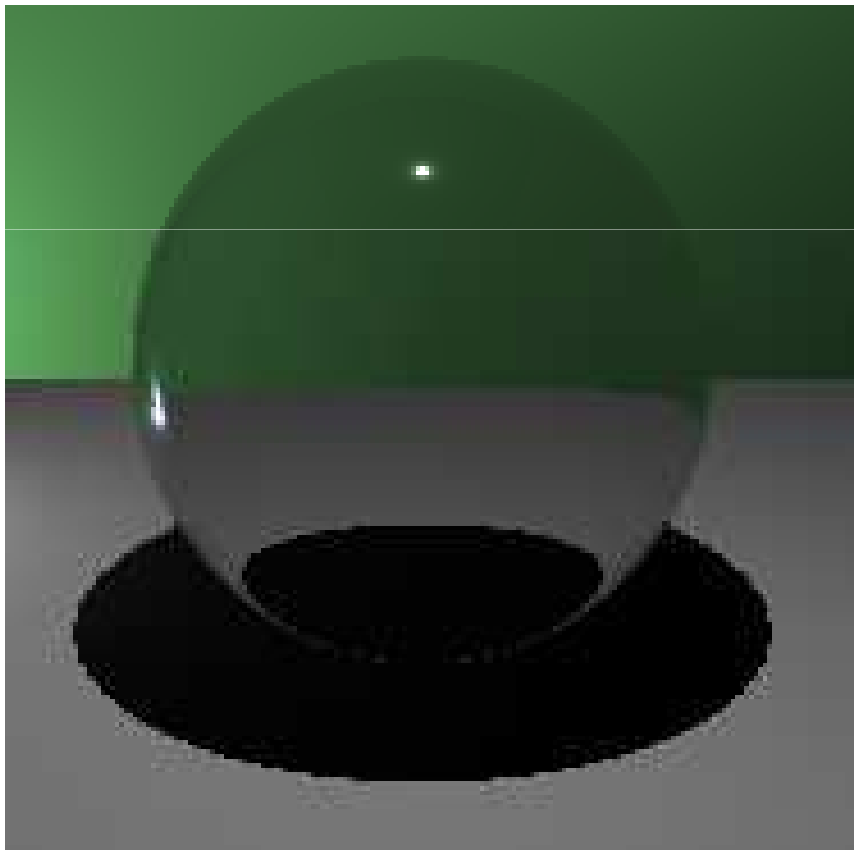
การบรรเทาเอเลียสซิง

เอเลียสซิง

- เอเลียสซิงเวลาทำการสร้างภาพสามมิติเกิดขึ้นจาก
  - ฟังก์ชันภาพเป็นฟังก์ชันความถี่ไม่จำกัด เนื่องจากมีความไม่ต่อเนื่องของสี ณ ขอบเขตของวัตถุและเงา
  - จิตรกรรมฝาผนังมีความไม่ต่อเนื่อง

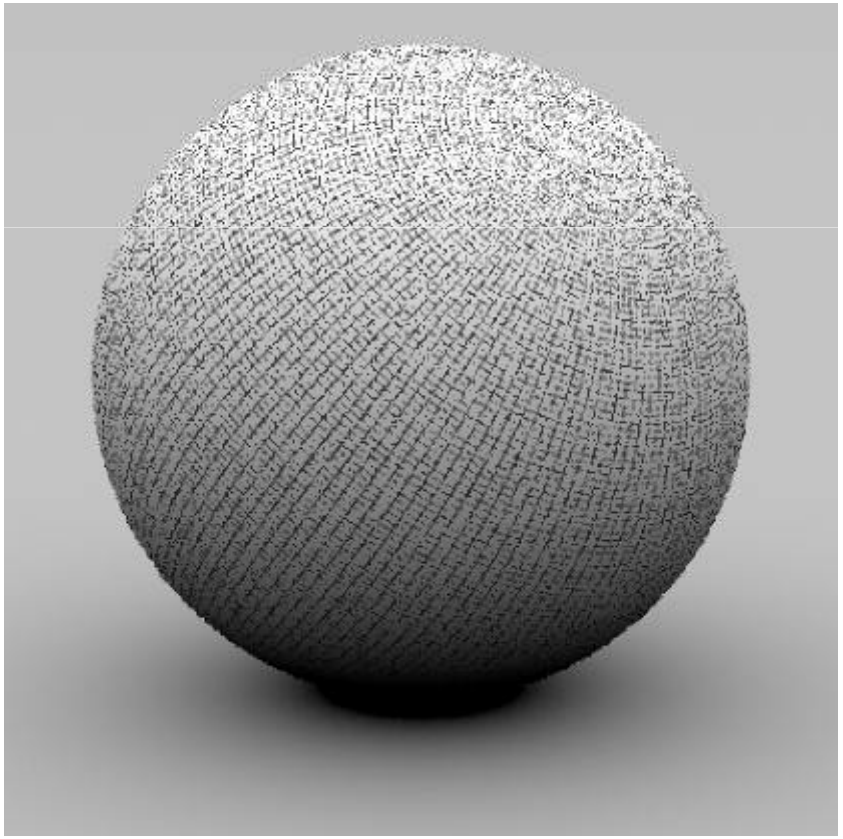
เอเลียสซิง (ต่อ)

- เอเลียสซิงที่เกิดจากขอบเขตของวัตถุ



เอเลียสซิง (ต่อ)

- เอเลียสซิงที่เกิดจากความไม่ต่อเนื่องของจิตรกรรมฝาผนัง



เอเลียสซิง (ต่อ)

- ปกติแล้วการบรรเทาเอเลียสซิงจากความไม่ต่อเนื่องของรูปทรงและเงาทำได้ยาก
  - ความไม่ต่อเนื่องของรูปทรงและเงาเป็นฟังก์ชันที่มีความถี่ไม่จำกัด
  - เราไม่สามารถลบส่วนความถี่ไม่จำกัดนี้ออกไปได้
- แต่การบรรเทาเอเลียสซิงจากความไม่ต่อเนื่องของจิตรกรรมฝาผนังทำได้ง่ายกว่า
  - เพราะเราสามารถดัดแปลงรูปเพื่อลบส่วนที่มีความถี่สูงออกไปได้ก่อนนำมันมาใช้คำนวณสี

เทคนิคการบรรเทาเอเลียสซิงทั่วๆ ไป

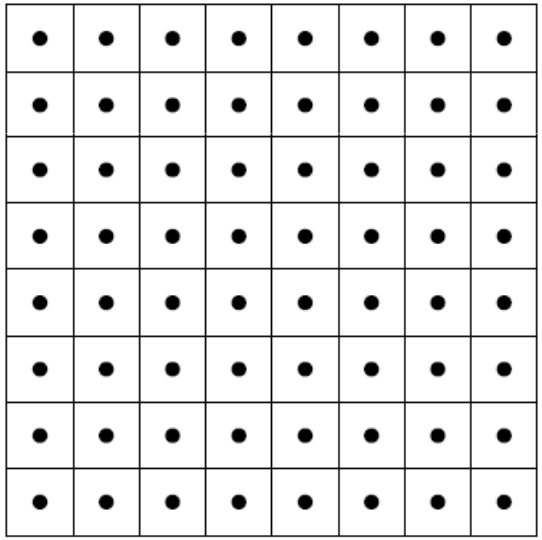
- เพิ่มความถี่การชักตัวอย่าง (วาดให้ละเอียดขึ้น)
  - ใช้ได้ผลดี แต่ไม่ค่อยคุ้มค่าเพราะต้องเสียวาดเพิ่มขึ้น
- ชักตัวอย่างแบบสุ่มๆ
  - แทนที่จะชักตัวอย่างแบบเป็นตารางมีระเบียบ ก็ชักให้มันหลุดๆ ออกไปจากตำแหน่งที่มันควรจะเป็นสักเล็กน้อย แบบสุ่มๆ
  - ทำให้ภาพมีคลื่นรบกวนเยอะขึ้น แต่เอเลียสซิงก็ถูกเปลี่ยนเป็นคลื่นรบกวนด้วย ทำให้ภาพดูสบายตาขึ้น

เทคนิคการบรรเทาเอเลียสซิงทั่วๆ ไป (ต่อ)

- เพิ่มความถี่การชักตัวอย่างโดยอาศัยการเรียนรู้จากภาพ
  - ถ้าภาพส่วนไหนมีความเปลี่ยนแปลงสีมาก ก็ให้สุ่มมากขึ้น
    - ส่วนขอบของวัตถุ
    - จุดสองจุดที่อยู่ข้างกันที่สีต่างกันมากๆ
  - ลดการคำนวณลงได้พอสมควร แต่ก็มีข้อบกพร่อง
    - จุดสองจุดที่อยู่ติดกันบางคู่ อาจจะมีสีต่างกันมาก แต่ความจริงไม่ต้องสุ่มเพิ่มก็ได้ เช่น จุดสองจุดข้างกันที่สีต่างกันเนื่องจากจิตรกรรมฝาผนัง
    - บริเวณที่ต้องการความถี่การสุ่มสูงอาจจะไม่ถูกสุ่มความถี่สูงขึ้นก็ได้

การชักตัวอย่างแบบ “มั่วในบล็อก”

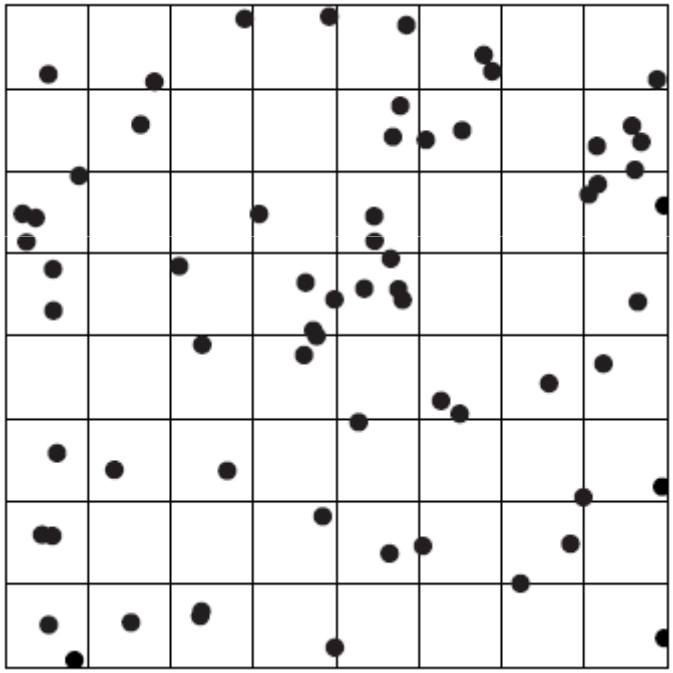
- ปกติเราจะชักตัวอย่างโดยในแต่ละพิกเซลเราจะยิง “รังสี” ไปที่กลางพิกเซลนั้น



- แต่มันทำให้เกิดเอเลียสซิงได้ง่าย

การชักตัวอย่างแบบ “มั่วในบล็อก” (ต่อ)

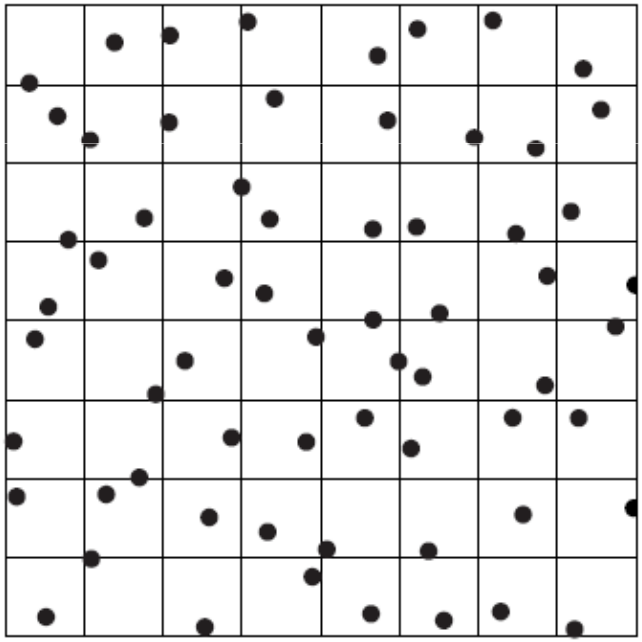
- เราสามารถชักตัวอย่างแบบมั่วๆ เพื่อเปลี่ยนเอเลียสซิงเป็นคลื่นรบกวน



- แต่แบบนี้บางพิกเซลจะไม่ถูกชักตัวอย่างออกมาเลย

การชักตัวอย่างแบบ “มั่วในบล็อก” (ต่อ)

- เราสามารถรวมสองวิธีข้างต้นเข้าด้วยกัน โดยในแต่ละพิกเซลเราจะชักตัวอย่างมาหนึ่งตัว แต่ตำแหน่งของตัวอย่างนั้นจะมั่วเอา



- วิธีนี้เป็นวิธีชักตัวอย่างที่ได้ผลดีมาก

การชักตัวอย่างแบบ “มั่วในบล็อก” (ต่อ)

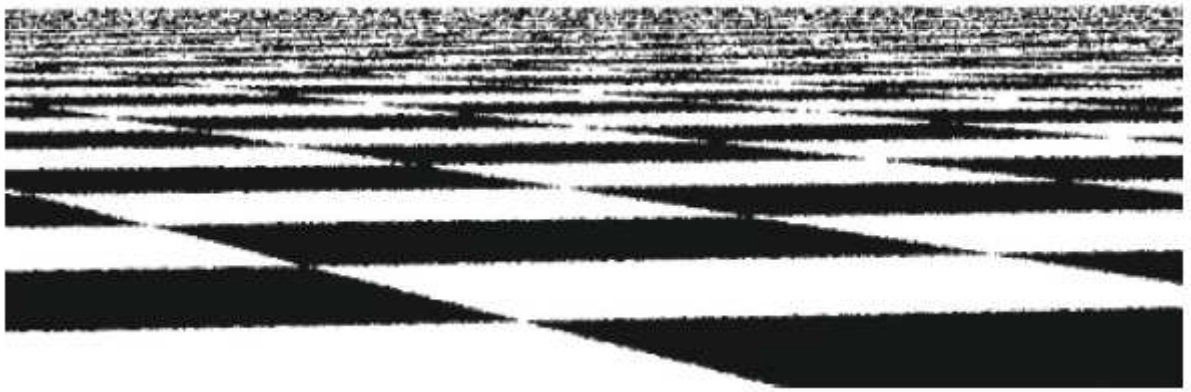
- ภาพในอุดมคติ คำนวณด้วยการชักตัวอย่าง 256 ตัวอย่างในหนึ่งพิกเซล

การชักตัวอย่างแบบ “มั่วในบล็อก” (ต่อ)

- 1 ตัวอย่างต่อหนึ่งพิกเซล

การชักตัวอย่างแบบ “มั่วในบล็อก” (ต่อ)

- มั่วในบล็อก หนึ่งตัวอย่างต่อหนึ่งพิกเซล



การชักตัวอย่างแบบ “มั่วในบล็อก” (ต่อ)

- มั่วในบล็อก 4 ตัวอย่างต่อหนึ่งพิกเซล

ฟิลเตอร์การสร้างกลับ

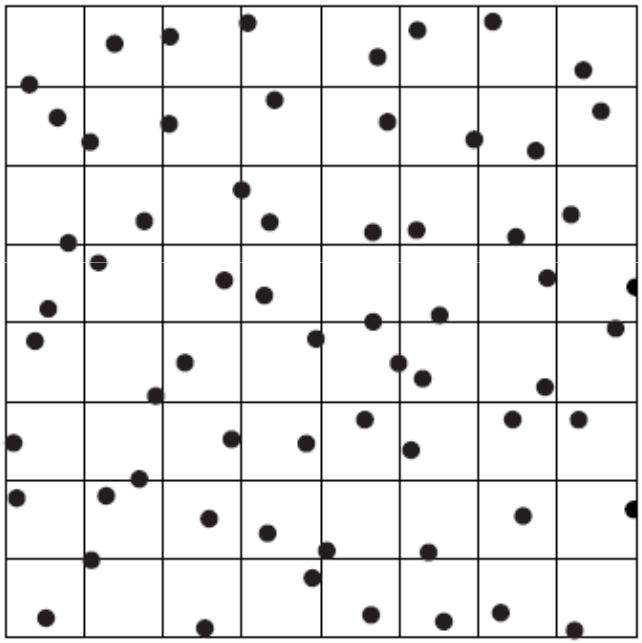
การสร้างกลับ (อีกรอบ)

- การสร้างภาพกลับคือการนำเอาฟังก์ชันที่ชักตัวอย่างออกมาแล้วมาทำคอนโวลูชันกับฟังก์ชันตัวหนึ่ง
- ฟังก์ชันตัวนี้เราจะเรียกว่า **เคอร์เนล (kernel)** หรือ **ฟิลเตอร์ (filter)**
- เรารู้ว่าฟังก์ชันซิงค์เป็นฟังก์ชันการสร้างกลับในอุดมคติ
  - มันสามารถสร้างฟังก์ชันเดิมออกมาได้ถ้าเราชักตัวอย่างด้วยความถี่สูงกว่าความถี่ไนควิสท์

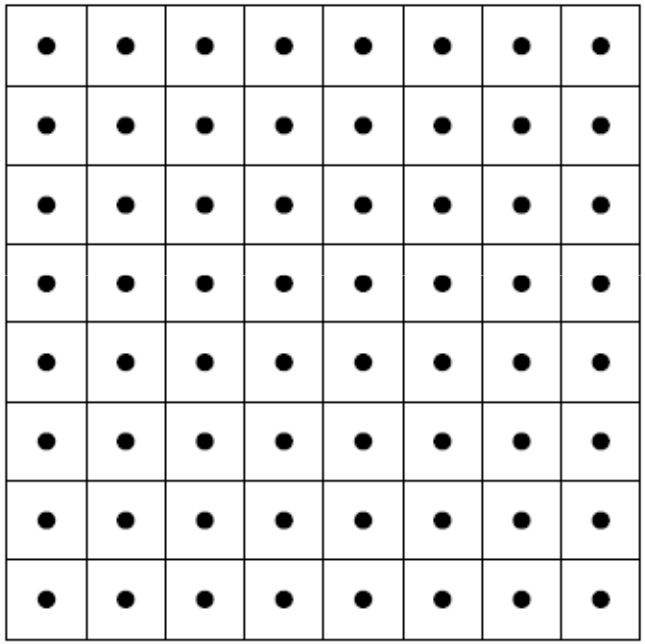
การสร้างกลับ (ต่อ)

- แต่จริงๆ เวลาจะแสดงผล เราไม่ได้สร้างฟังก์ชันต่อเนื่องต้นฉบับกลับแต่อย่างไร
- เวลาเราสั่งจอภาพให้แสดงภาพ หรือจะเซฟภาพลงในไฟล์ เราต้องส่งตัวอย่างของฟังก์ชันไปให้จอภาพหรือดิสก์อยู่ดี
- ตัวอย่างที่จะส่งให้จอภาพหรือดิสก์ เป็นตัวอย่างที่จุดของมันเรียงกันเป็นระเบียบ หนึ่งตัวอย่างต่อหนึ่งพิกเซล
- แต่ตอนเราชักตัวอย่าง เราอาจไม่ได้ชักตัวอย่างแบบนั้น (ถ้าชักแบบนั้นก็แล้วไป ไม่ต้องทำอะไร)
- บางที เราอาจชักตัวอย่างมากกว่าหนึ่งตัวอย่างในหนึ่งพิกเซล

การสร้างกลับ (ต่อ)



เราอาจจะชักตัวอย่างแบบนี้



แต่ตัวอย่างที่ต้องส่งให้จอภาพเป็นแบบนี้

การสร้างกลับ (ต่อ)

- สิ่งที่เราต้องทำคือต้องหาค่าของตัวอย่างที่ชักมาอย่างเป็นระเบียบ หนึ่งตัวอย่างต่อหนึ่งพิกเซล จากตัวอย่างที่เราชักมาจริงๆ ซึ่งอาจไม่ค่อยเป็นระเบียบนัก หรืออาจมีมากกว่าหนึ่งตัวต่อหนึ่งพิกเซล

การสร้างกลับ (ต่อ)

- การหาค่าที่ว่านี้เราจะนำเอาตัวอย่างที่อยู่ใกล้ๆ จุดที่เราต้องการหาค่าฟังก์ชันมาถ่วงน้ำหนักแล้วเฉลี่ย

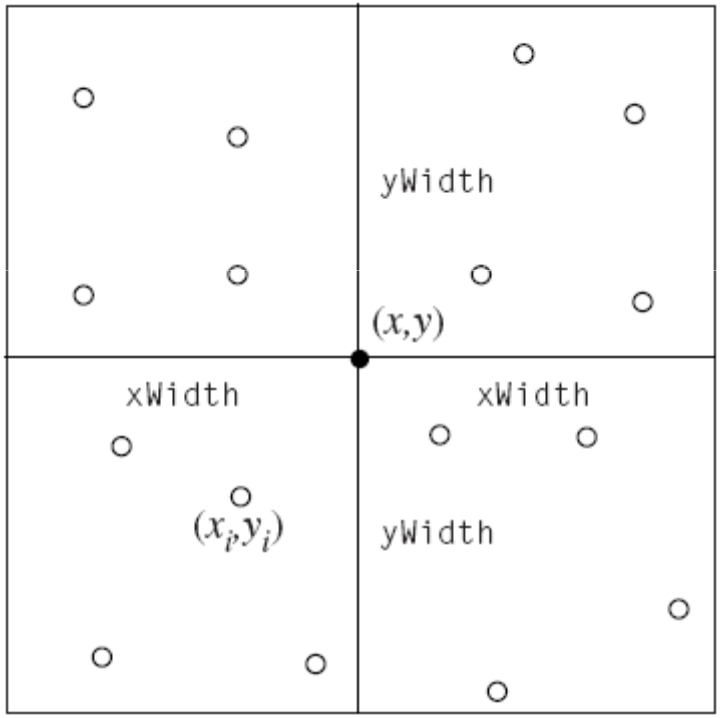
$$I(x,y) = \frac{\sum_i f(x - x_i, y - y_i) L(x_i, y_i)}{\sum_i f(x - x_i, y - y_i)}$$

โดยที่ $f(\cdot,\cdot)$ คือฟิลเตอร์

$L(\cdot,\cdot)$ คือค่าความเข้มแสงที่ได้จากการยิงเรย์

$I(\cdot,\cdot)$ คือค่าความเข้มแสงที่เราต้องการหา

การสร้างกลับ (ต่อ)



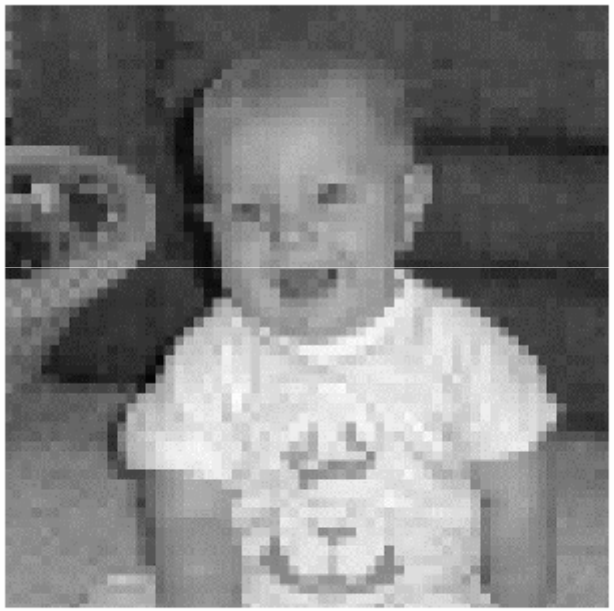
ฟิลเตอร์

- เราใช้ซิงค์ฟังก์ชันเป็นฟิลเตอร์ไม่ได้
  - มันมีขอบเขตกว้างเกินไป หมายความว่าจะต้องใช้ตัวอย่างทุกตัวที่สุ่มมา
  - นอกจากนี้มันยังอาจทำให้เกินคลื่นกระเพื่อม (ringing) ในภาพได้ ถ้าตัวอย่างที่สุ่มมาความถี่ต่ำกว่าความถี่ไนควิสท์

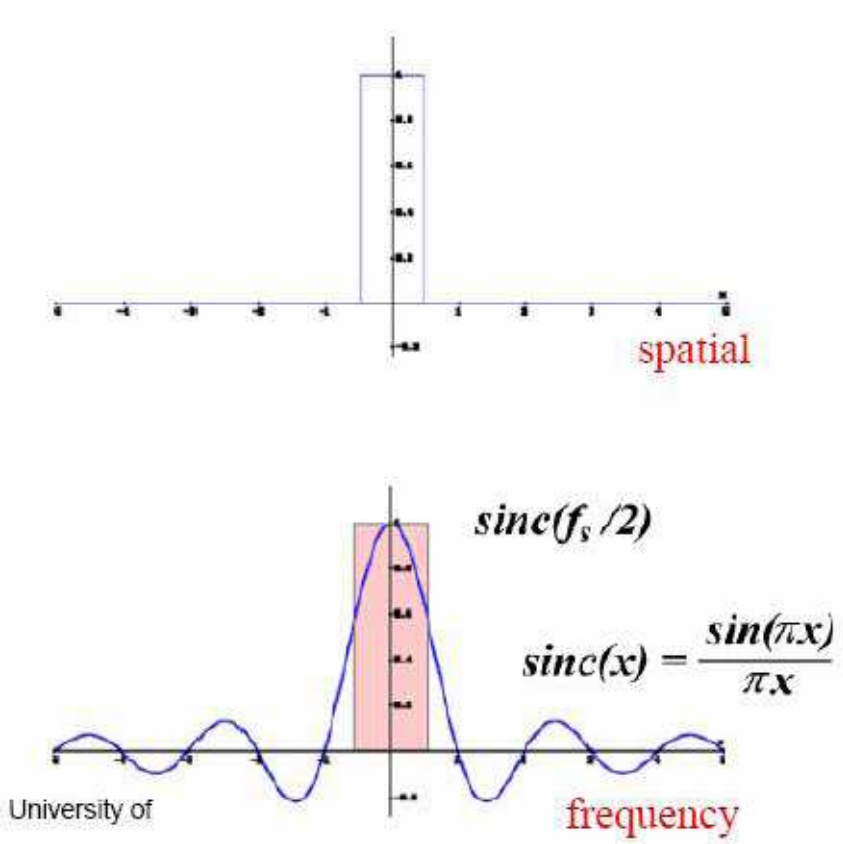
ฟิลเตอร์ (ต่อ)

- ฟิลเตอร์ที่คนนิยมใช้ ได้แก่
  - กล่อง
  - สามเหลี่ยม
  - เกาส์
  - มิทเชล
  - ซิงค์แบบถูกจำกัดขอบเขต (แลงค์ซอส

กล่อง



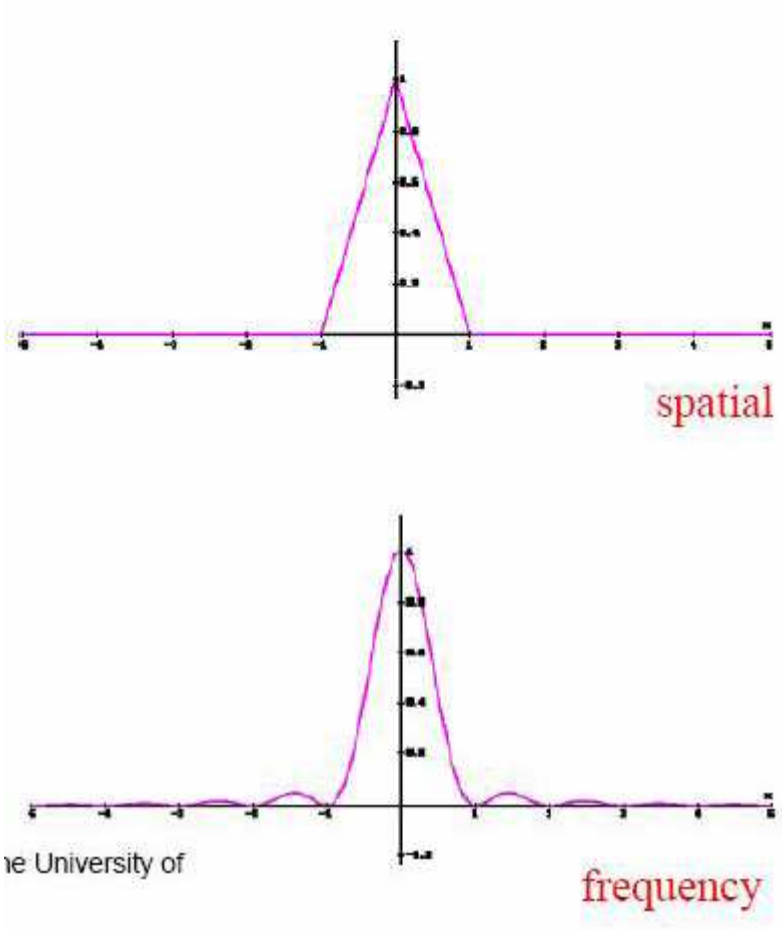
Courtesy of Leonard McMillan, Computer Science at the Unive
North Carolina in Chapel Hill. Used with permission.



เอเลียสซิงเต็มไปหมด

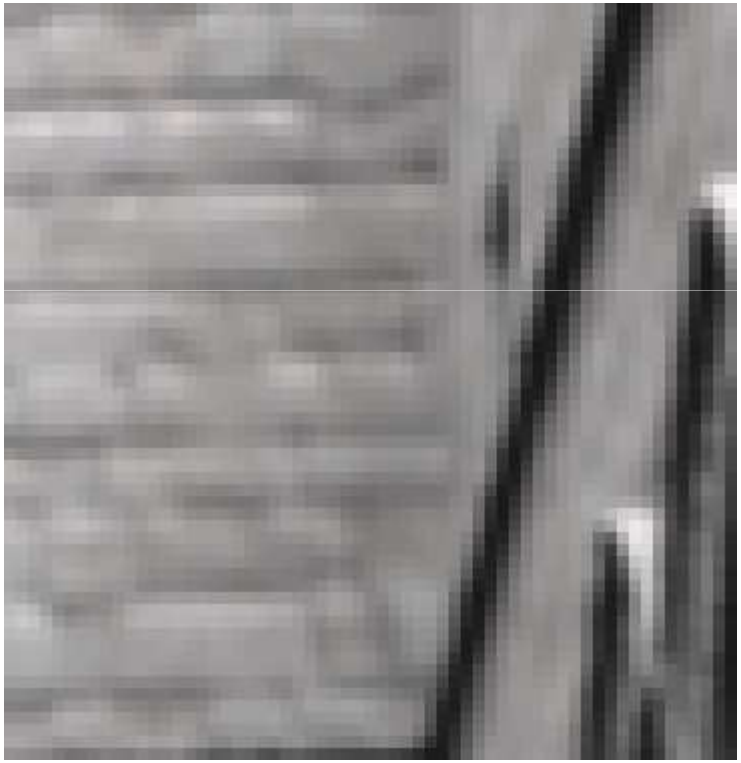
สามเหลี่ยม

Courtesy of Leonard McMillan, Computer Science at the
North Carolina in Chapel Hill. Used with permission.

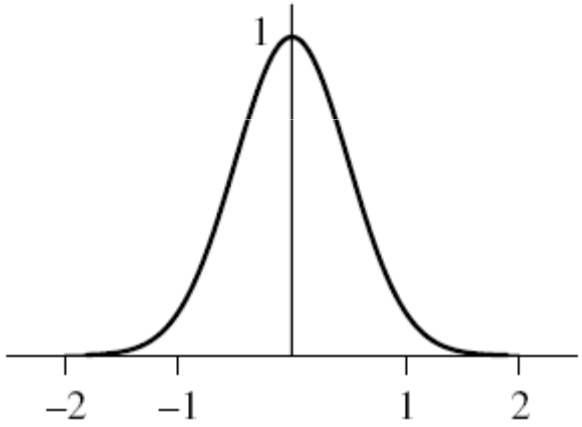


เรียบและเบลอ

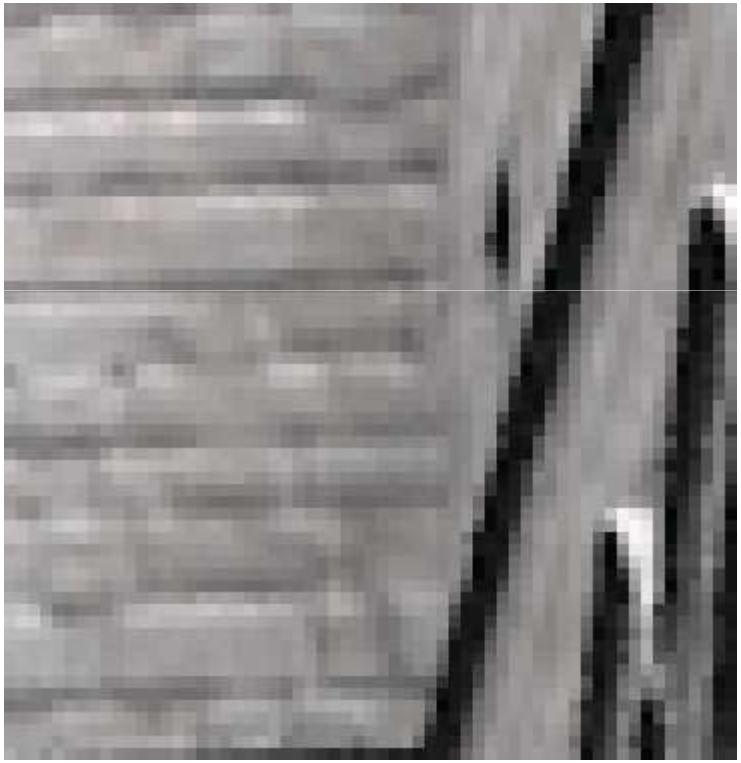
เกาส์



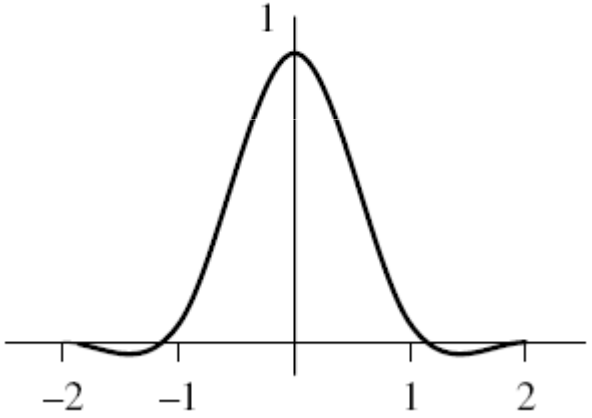
เบลอมากๆ



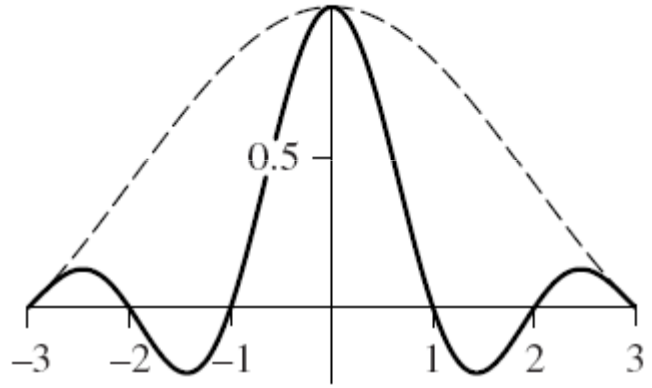
มิทเชล



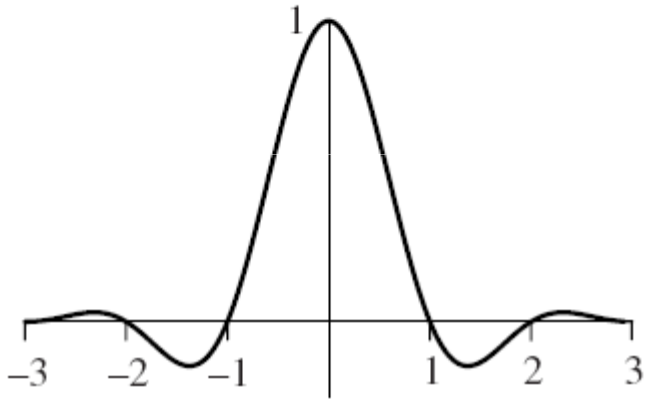
คมกว่าเกาส์



ซิงค์แบบถูกจำกัดขอบเขต



(a)



(b)

ANTI-ALISING ใน OPENGL

Antialiasing ใน OpenGL

- การทำ antialiasing ของจุดและเส้น
- การทำ multisampling

การทำ antialiasing ของจุดและเส้น

- ถ้าต้องการทำ antialiasing ของจุดให้สั่ง
  glEnable(GL_POINT_SMOOTH);
- ถ้าต้องการทำ antialiasing ของเส้นให้สั่ง
  glEnable(GL_LINE_SMOOTH);
- เมื่อถ้าจะเลิกใช้ให้สั่ง glDisable(...) ตามสมควร

การทำ antialiasing ของจุดและเส้น (ต่อ)

- การทำ antialiasing ของจุดและเส้นใน OpenGL เป็นการกำหนดค่า alpha ของ fragment ที่ประกอบขึ้นเป็นจุดหรือเส้นนั้น
- ค่า alpha ถูกคำนวณด้วย unweighted area sampling
- ดังนั้นถ้าจะทำ antialiasing จะต้อง
  - สั่ง glEnable(GL_BLENDING);
  - และสั่ง glBlendFunc(...) ให้เหมาะสม

การทำ antialiasing ของจุดและเส้น (ต่อ)

- แล้วจะสั่ง glBlendFunc อย่างไร?
- ปกติแล้วใช้

  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
- แต่ก็สามารถใช้

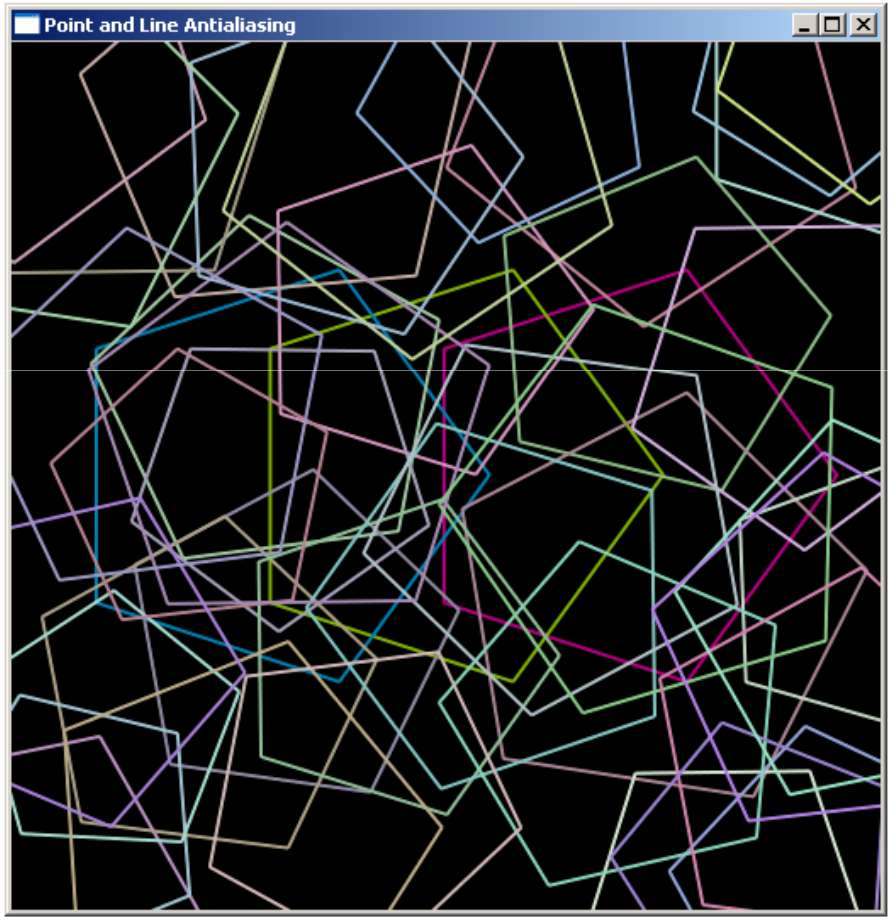
  glBlendFunc(GL_SRC_ALPHA, GL_ONE)

  เพื่อทำให้จุดตัดสว่างขึ้น

การทำ antialiasing ของจุดและเส้น (ต่อ)

- ข้อควรระวัง
  - เนื่องจากเราใช้ blending ลำดับการวาดเส้นและจุดจึงมีผลต่อรูป
  - ถ้ามีวัตถุทึบแสงอยู่ในฉากด้วย
    - ต้องวาดวัตถุทึบแสงให้หมดก่อน แล้วค่อยวาดเส้นและจุด
    - และจะต้องมีการทำให้ depth buffer เป็นแบบ readonly ด้วย
    - ดูรายละเอียดในการบรรยายครั้งที่ 16

ดู demo



การทำ multisampling

- สามารถใช้ได้กับรูปทรงใดๆ ก็ได้
- แต่เสียเวลาการทำนานมาก
- หลักการคือสร้าง fragment เพิ่มขึ้น 4 เท่า
- แล้วทำสร้างภาพที่จะนำไปแสดงโดยให้สีของ 1 fragment ของ primitive ที่จะเอาไปแสดง เท่ากับค่าเฉลี่ยของสีของ 4 fragment (สี่เหลี่ยมกว้าง 2 pixel ยาว 2 pixel) ของที่สร้างขึ้นมา

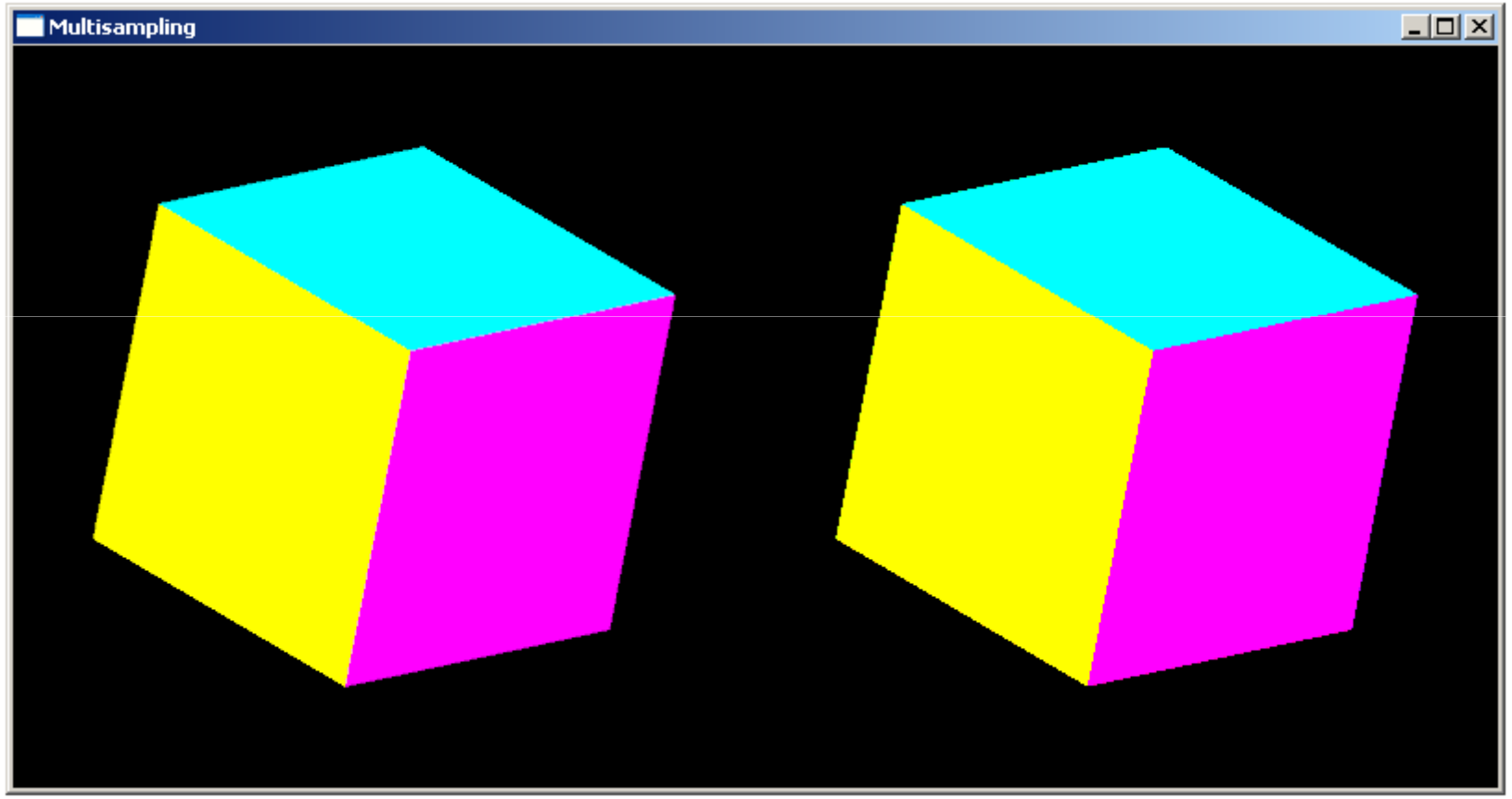
ขั้นตอน

- เพิ่ม GLUT_MULTISAMPLE ลงใน glutInitDisplayMode ดังนี้:

```
glutInitDisplayMode(GLUT_DOUBLE |
    GLUT_RGB | GLUT_DEPTH |
    GLUT_MULTISAMPLE);
```

- เวลาใช้ให้สั่ง glEnable(GL_MULTISAMPLING);

ดู demo



ปัญหาการ compile บน Windows

- Multisampling เป็นความสามารถใหม่ที่เพิ่มเข้ามาใน OpenGL เวอร์ชันหลังๆ
- แต่ OpenGL ที่มากับ Windows อยู่ที่เวอร์ชัน 1.1 ซึ่งมีอายุประมาณ 10 ปีแล้ว
- ดังนั้นถ้า compile โค้ดตัวอย่างตามธรรมดาแล้วจะพบว่ามันไม่รู้จักค่า GL_MULTISAMPLE

GLEW

- OpenGL Extension Wrangler (GLEW) เป็นไลบรารีที่ใช้ในการเข้าถึงความสามารถของ OpenGL อันใหม่ๆ ที่ไม่มีในเวอร์ชันเก่า หรือที่มีเฉพาะในฮาร์ดแวร์บางตัว
- สามารถ download มันได้ที่ http://glew.sourceforge.net
- ให้ download Win32 binary ของมันมา แล้วแตก zip ไฟล์
- นำ glew32.dll ไปใส่ไว้ที่ c:\windows\system32
- เสร็จแล้วนำ glew.h และ glew32.lib ไปใส่ไว้ใน solution ของโปรแกรมของเรา เช่นเดียวกับ GLUT

GLEW (ต่อ)

- เวลาใช้งาน GLEW ให้ include ไฟล์ glew.h ก่อน glut.h เช่น

```
#include <windows.h>
#include <GL/glew.h>
#include <GL/glut.h>
```