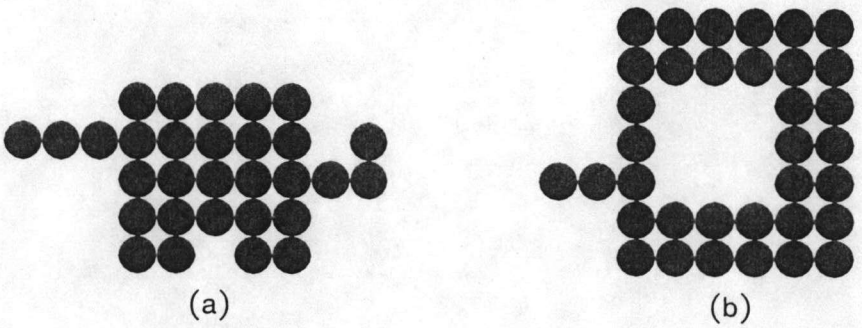
บทที่ 3

ทฤษฎีคอมพิวเตอร์กราฟฟิกที่เกี่ยวข้อง

คอมพิวเตอร์กราฟฟิกเป็นทฤษฎีพื้นฐานของระบบคอมพิวเตอร์ช่วยการออกแบบ (computer-aided design) หรือ CAD การใช้ภาพประกอบและกราฟฟิกทำให้ผู้ใช้มีความเข้าใจในข้อมูลดีขึ้น ระบบติดต่อผู้ใช้แบบกราฟฟิกช่วยให้การใช้งานโปรแกรมง่ายขึ้นมาก ความเข้าใจในพื้นฐานทฤษฎีทางคอมพิวเตอร์กราฟฟิกเป็นส่วนสำคัญอย่างยิ่งในการจำลองแบบทางกายภาพต่าง ๆ ทฤษฎีคอมพิวเตอร์กราฟฟิกที่เกี่ยวข้องกับการสร้างโปรแกรมนำเข้าข้อมูลพิกัดจากภาพที่ผู้วิจัยใช้ในงานวิจัยคือ การระบาย (flood-fill) เส้นโค้งบี-สไปลน์ และพื้นผิวคูนส์ ซึ่งจะกล่าวถึงต่อไป

การระบาย (flood-fill)*

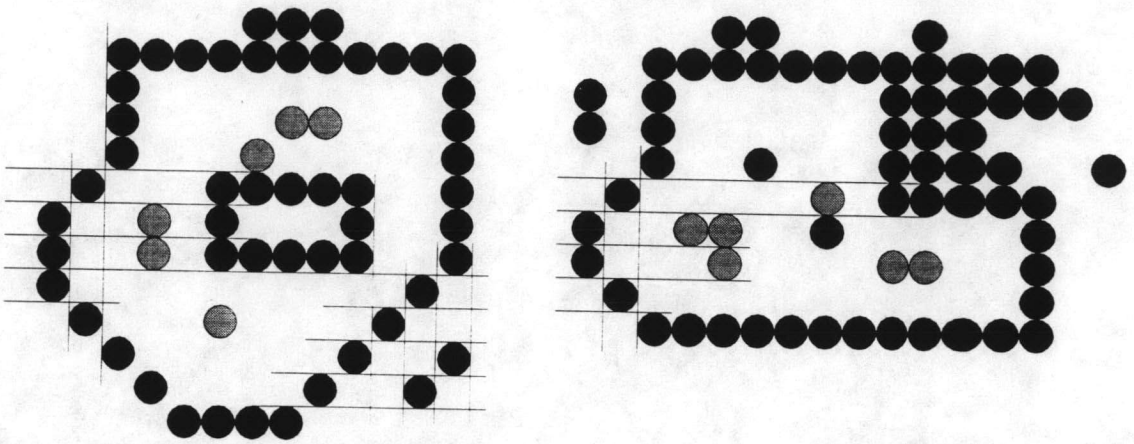
เขตภาพ (region) คือกลุ่มของจุดสีที่อยู่ติดกันต่อเนื่องกันไป การกำหนดขอบเขตของเขตภาพสามารถทำได้โดยกำหนดค่าค่าหนึ่งให้แก่จุดสีที่อยู่ในเขตภาพนั้น หรือกำหนดค่าให้แก่จุดสีที่เป็นขอบเขตของเขตภาพ รูป 3.1 และ 3.2 แสดงเขตภาพที่ประกอบด้วยจุดสีดำทั้งหมด



รูปที่ 3.1 เขตภาพ 4 ด้านกำหนดด้วยจุดภายใน (interior-defined 4-connected region) ซึ่งเป็นจุดสีดำ

---

* เนื้อความและภาพประกอบจาก Foley and Van Dam (1982)



(a) (b)

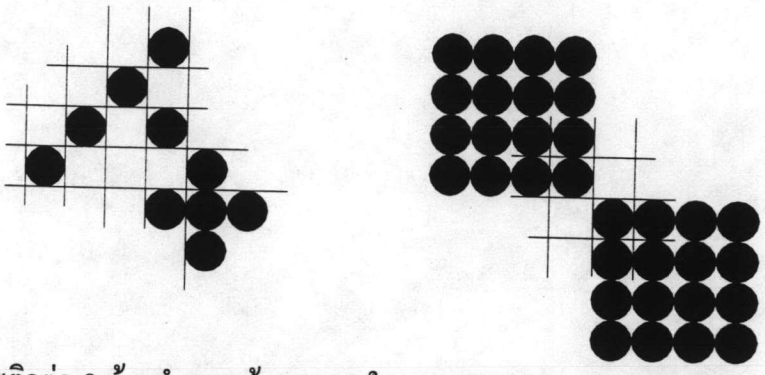
รูปที่ 3.2 เขตภาพ 4 ด้านกำหนดด้วยขอบเขตซึ่งเป็นจุดสีดำ สังเกตว่าจุดภายนอกขอบเขตสามารถเป็นสีเดียวกับขอบเขตได้

เขตภาพที่กำหนดโดยค่าของจุดที่อยู่ภายในเรียกว่า<u>กำหนดจากภายใน</u> (interior-defined) จุดสีทั้งหมดในเขตภาพจะมีค่าเป็น old_value และบนขอบเขตจะไม่มีจุดสีที่มีค่าเป็น old_value อัลกอริทึมที่เปลี่ยนค่าของจุดสีในขอบเขตนี้ทั้งหมดจาก old_value ให้เป็น new_value เรียกว่า flood-fill algorithm

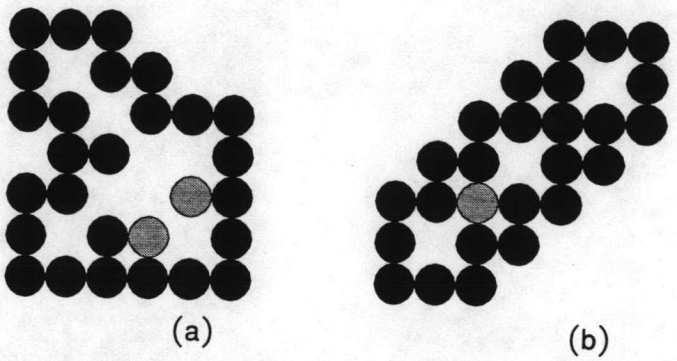
เขตภาพที่กำหนดโดยค่าของขอบเขตเรียกว่า<u>กำหนดด้วยขอบเขต</u> (boundary-defined) จุดสีที่อยู่บนขอบเขตจะมีค่าเป็น boundary_value จุดสีที่อยู่ในเขตภาพมีค่าเป็นอะไรก็ได้นอกจาก new_value อัลกอริทึมที่เปลี่ยนค่าของเขตภาพนี้ทั้งหมดเป็น new_value เรียกว่า boundary-fill algolrithm

ในการอธิบายอัลกอริทึมจะกำหนดคำเพื่อกล่าวถึงชนิดของเขตภาพที่อยู่ติดกับจุดสี เขตภาพที่อยู่ติดกับจุดสีมี 2 ชนิดคือ <u>ติดต่อ 4 ด้าน</u> (4-connected) และ<u>ติดต่อ 8 ด้าน</u> (8-connected) อัลกอริทึมที่ทำงานกับเขตภาพติดต่อ 8 ด้านจะใช้กับเขตภาพติดต่อ 4 ด้านได้ด้วย แต่อัลกอริทึมสำหรับเขตภาพติดต่อ 4 ด้านใช้กับเขตภาพติดต่อ 8 ด้านไม่ได้ ในอัลกอริทึมแบบติดต่อ 4 ด้านจากจุดสีหนึ่งจะสามารถไปถึงจุดสีอื่นๆได้ในทิศทาง ขึ้น ลง ซ้าย ขวา รูปที่ 3.1 แสดงเขตภาพกำหนดจากภายในที่เป็นชนิดติดต่อ 4 ด้าน ในรูป (b) จะมีช่องว่างในเขตภาพด้วย รูปที่ 3.2 แสดงเขตภาพกำหนดโดยขอบเขตชนิดติดต่อ 4 ด้าน 2 ผืนซึ่งมีขอบเขตเป็นสีดำและภายในเป็นสีขาวและสีเทา ในกรณีนี้จุดภายในเขตภาพสามารถเป็นสีอะไรก็ได้นอกจากสีดำ ในรูป 3.2 (a) พื้นที่สี่เหลี่ยมที่อยู่ภายในนับเป็นหนึ่งเขตภาพ พื้นที่นอกรูปสี่เหลี่ยมและอยู่ภายในขอบเขตด้าน

นอกนับเป็นอีกหนึ่งเขตภาพ พื้นที่สี่เหลี่ยมที่อยู่ภายในทำให้เกิดช่องว่างในเขตภาพด้านนอก ใน รูป 3.2 (b) จุดสีดำที่อยู่ภายในทำให้เกิดช่องว่างในเขตภาพ



รูป 3.3 เขตภาพติดต่อ 8 ด้านกำหนดด้วยจุดภายใน (interior-defined 4-connected region) ซึ่งเป็นจุดสีดำ



รูป 3.4 เขตภาพติดต่อ 4 ด้านกำหนดด้วยขอบเขตซึ่งเป็นจุดสีดำ

ในเขตภาพติดต่อ 8 ด้าน การเข้าถึงจุดสีหนึ่งจากจุดสีอื่น ๆ สามารถทำได้ในทิศทางแนวนอน แนวตั้ง และแนวทะแยงมุม รูป 3.3 แสดงเขตภาพติดต่อ 8 ด้านกำหนดจากภายในเป็นจุดสีดำ รูป 3.4 แสดงเขตภาพติดต่อ 8 ด้านกำหนดจากภายในถูกล้อมรอบด้วยขอบเขตสีดำ การสังเกตพบสิ่งที่น่าสนใจคือขอบเขตของเขตภาพติดต่อ 8 ด้านจะเป็นเขตภาพติดต่อ 4 ด้าน ในขณะที่ขอบเขตของเขตภาพติดต่อ 4 ด้านเป็นเขตภาพติดต่อ 8 ด้าน

อัลกอริทึมที่ใช้ระบาย (fill) เขตภาพติดต่อ 8 ด้านจะมีการทำงานกระโดดข้ามจุดสีที่ติดต่อกันในแนวเส้นทะแยงมุมซึ่งอาจให้ผลการระบายที่ไม่คาดคิดได้ ตัวอย่างเช่นถ้าทำการระบายรูปสี่เหลี่ยมบนซ้ายในรูป 3.3 ด้วยอัลกอริทึมแบบติดต่อ 8 ด้านจะทำให้รูปสี่เหลี่ยมด้านขวาล่าง

ถูกระบายไปด้วยซึ่งอาจเป็นสิ่งที่เกิดขึ้นโดยไม่ได้ตั้งใจ แต่การใช้อัลกอริทึมแบบติดต่อ 4 ด้าน ระบายสี่เหลี่ยมด้านซ้ายบนจะไม่ทำให้การระบายแพร่ไปถึงรูปสี่เหลี่ยมด้านซ้ายล่าง

อัลกอริทึมการระบายแบบติดต่อ 4 ด้าน (4-connected fill)

อัลกอริทึมอย่างง่ายที่ใช้ในการระบายเขตภาพกำหนดจากภาพในเป็นดังนี้

```
procedure FLOOD_FILL_4(
    x, y,                          {starting point in 4-connected
                                    interior-defined region}
    old_value,                     {value in interior-defined region}
    new_value : integer);          {replacement value for old_value}
                                   {old_value must not equal new_value}
begin
    if READ_PIXEL(x, y) = old_value
        then begin
            WRITE_PIXEL(x, y, new_value);  {change value}
            {attemp to propagate in each of four directions}
            FLOOD_FILL_4(x, y - 1, old_value, new_value);
            FLOOD_FILL_4(x, y + 1, old_value, new_value);
            FLOOD_FILL_4(x - 1, y, old_value, new_value);
            FLOOD_FILL_4(x + 1, y, old_value, new_value);
    end
end      {FLOOD_FILL_4}
```

วิธีการคือ ในขั้นแรกจะพิจารณาว่าจุดสี (x, y) เป็นส่วนของบริเวณซึ่งยังไม่เคยไปซึ่งก็คือมีค่าเป็น old_value หรือไม่ ถ้าเป็นเช่นนั้นก็จะเปลี่ยนค่าเป็น new_value และจะพิจารณาจุดสีที่อยู่ติดกัน(4 จุด, ซ้าย ขวา บน ล่าง)ต่อไป อัลกอริทึมนี้สามารถดัดแปลงให้ใช้ระบายเขตภาพติดต่อ 8 ด้านโดยดำเนินการกับจุดสี 8 จุดที่อยู่ล้อมรอบ

ในทำนองเดียวกัน อัลกอริทึมอย่างง่ายที่ใช้ในการระบายเขตภาพติดต่อ 4 ด้านกำหนดจากภายในเป็นดังนี้

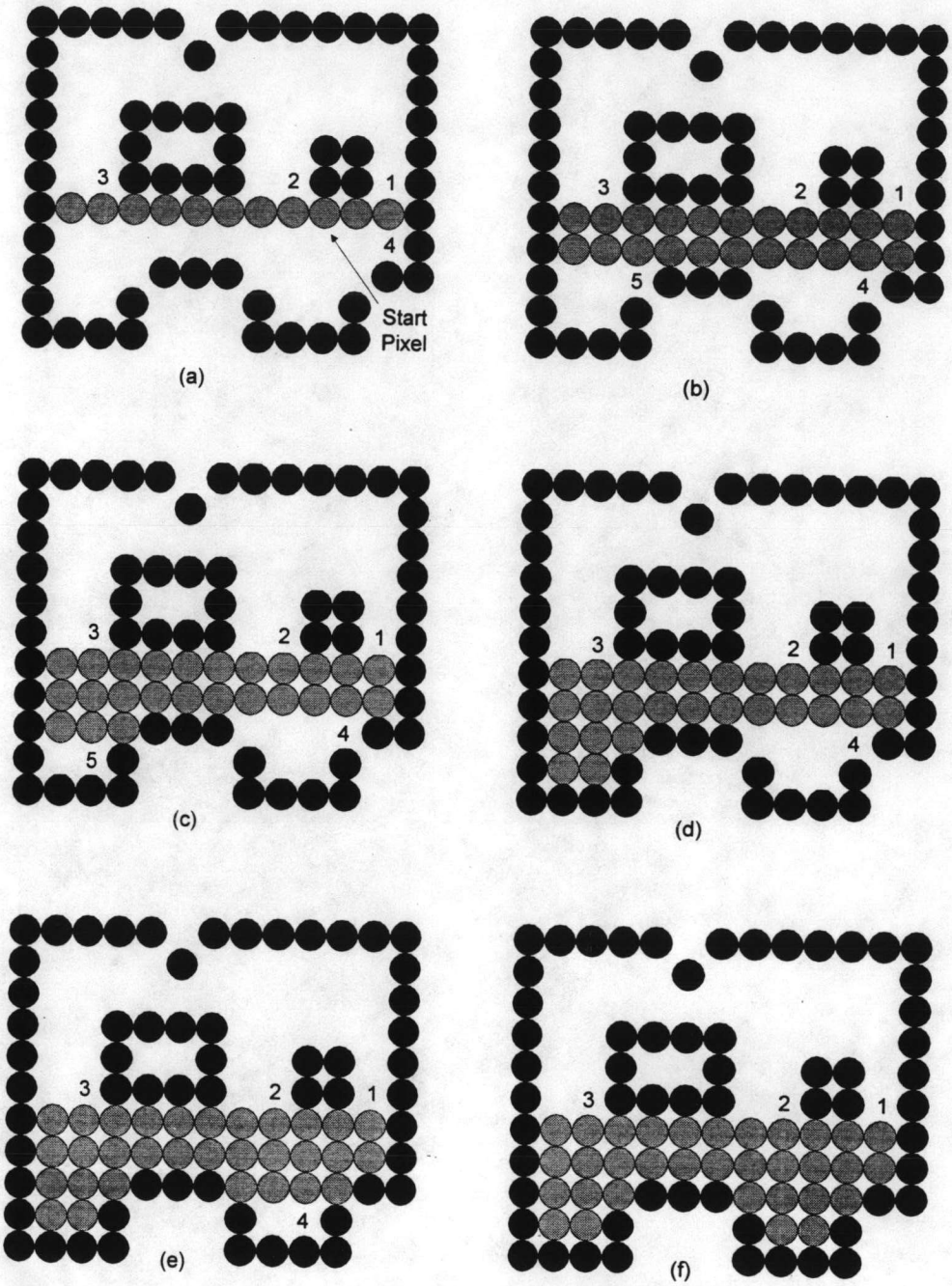
```
procedure BOUNDARY_FILL_4(
    x, y,                          {starting point in boundary-defined
                                    4-connected region}
    boundary_value,                {value on boundary}
    new_value : integer);          {replacement value}

    {it is permissible to have new_value = boundary_value;
     otherwise no pixels in region may be initially set to
     new_value}

begin
    if READ_PIXEL(x, y) <> boundary_value  {boundary not reached and}
    and READ_PIXEL(x, y) <> new_value      {previously filled pixel
                                            not reached}
        then begin
            WRITE_PIXEL(x, y, new_value);     {change value}
            {attempt to propagate in 4 directions}
            four calls to BOUNDARY_FILL_4
        end
end      {BOUNDARY_FILL_4}
```

จะเห็นได้ว่าวิธีการเหมือนกับ FLOOD_FILL_4 นอกจากการทดสอบที่จุดสี (x, y) ถ้าไม่ได้เป็นจุดที่เคยถูกระบายไปแล้วด้วย new_value และไม่ได้เป็นจุดที่มีค่าเป็น boundary_value ก็จะระบายจุดนั้นด้วย new_value การทดสอบนี้จะรวมเอากรณีที่เราไม่ยอมให้จุดสีภายในเขตภาพมีค่าเป็น new_value มาก่อนการระบาย(ยกเว้นกรณีให้ new_value = boundary_value) เพราะอัลกอริทึมจะไม่นับเอาจุดสีนั้นเป็นส่วนหนึ่งของเขตภาพ อัลกอริทึมนี้สามารถดัดแปลงให้เป็น BOUNDARY_FILL_8 เช่นเดียวกับอัลกอริทึม FLOO_FILL_4 ที่กล่าวมาแล้ว

การลดฟังก์ชั่นเรียกตัวเอง (decreasing the recursive depth)

วิธีการที่กล่าวมาแล้วแม้จะเป็นวิธีที่ง่ายแต่จำนวนครั้งที่ฟังก์ชั่นเรียกตัวเอง (recursive) มาทำงานมีจำนวนมาก ซึ่งการเรียกตัวเองของฟังก์ชั่นจำนวนหลาย ๆ ชั้นนั้นทำให้เสียเวลามากและยังก่อให้เกิด stack overflow ได้ในกรณีที่หน่วยความจำมีจำกัด วิธีการระบายเขตภาพที่มีประสิทธิภาพสูงได้ถูกพัฒนาขึ้น วิธีเหล่านี้ใช้การทดสอบทางตรรกะมากขึ้นแต่จะไม่ทำให้เกิดปัญหาในเรื่องปริมาณการใช้ stack อัลกอริทึมจะทำงานกับช่วงแนวนอน (run) ซึ่งคือกลุ่มของจุดสีทางแนวนอน (horizontal pixel) ช่วงแนวนอนจะถูกกำหนดความยาวโดยจุดสีขอบเขตซึ่งมีค่าเป็น boundary_value พิกัดของจุดทางขวาสุดใช้เป็นตัวบ่งชี้ช่วงแนวนอน อัลกอริทึมและจะไม่ดำเนินการกับจุดสีที่มีค่าเป็น new_value อยู่ก่อน

การทำงานของอัลกอริทึมจะเป็นดังนี้ เริ่มจากการระบายช่วงแนวนอนที่ภายในมีจุดสีที่เป็นจุดเริ่มต้น หลังจากนั้นจะตรวจสอบแถวที่อยู่ด้านบนและติดกับช่วงแนวนอนที่กำลังถูกระบายทีละช่วงแนวนอนจากขวาไปซ้ายเพื่อจะหาพิกัดจุดสีทางขวาสุด พิกัดจุดสีที่ได้เหล่านี้จะถูกเก็บ (push) บน stack และจะดำเนินการเช่นเดียวกันนี้กับแถวที่อยู่ติดกันด้านล่าง เมื่อดำเนินเช่นนี้และใช้พิกัดจุดสีที่อยู่บนสุดของ stack เป็นจุดเริ่มต้นในการทำงาน(รอบ)ใหม่ อัลกอริทึมจะสิ้นสุดการทำงานเมื่อ stack ว่าง รูปที่ 3.5 แสดงการทำงานของอัลกอริทึมซึ่งดัดแปลงจาก Smith, 1979 ในรูป 3.5 (a) ช่วงแนวนอนที่มีจุดเริ่มต้นอยู่จะถูกระบายและพิกัดจุดสีของช่วงแนวนอนตามหมายเลขจะถูกเก็บบน stack ตัวเลขจะแสดงลำดับการเก็บช่วงแนวนอนบน stack หมายเลข 1 ถูกเก็บไว้ล่างสุดของ stack และจะถูกดำเนินการเป็นลำดับสุดท้าย การระบายจะดำเนินการไปจนระบายเขตภาพครบทั้งหมด



รูปที่ 3.5 การระบายบริเวณที่กำหนดด้วยขอบเขต จุดที่มีเลขกำกับจะถูกเก็บไว้ใน stack

เส้นโค้งบี-สไปลน์ (B-spline curve)*

P(t) คือเวคเตอร์ตำแหน่ง (position vector) ที่อยู่บนเส้นโค้งเป็นฟังก์ชั่นของพารามิเตอร์ t สมการของเส้นโค้งบี-สไปลน์เขียนอยู่ในรูปของ

$$\mathbf{P}(t) = \sum_{i=1}^{n+1} \mathbf{B}_i N_{i,k}(t), \quad t_{\min} \le t \le t_{\max}, \quad 2 \le k \le n+1 \tag{3.1}$$

เมื่อ $\mathbf{B}_i$ เป็นเวคเตอร์ตำแหน่งของจุดยอดรูปหลายเหลี่ยมจำนวน n+1 จุด และ $N_{i,k}(t)$ คือฟังก์ชั่นพื้นฐานบี-สไปลน์ปรับค่า (normalized B-spline basis function) k เป็นอันดับ (order) ของฟังก์ชั่นพื้นฐานบี-สไปลน์

Cox-deBoor ได้นิยามฟังก์ชั่นพื้นฐานบี-สไปลน์ $N_{i,k}(t)$ ไว้ดังนี้

$$N_{i,1}(t) = \begin{cases} 1 & if\ x_i \le t < x_{i+1} \\ 0 & otherwise \end{cases} \tag{3.2}$$

และ

$$N_{i,k}(t) = \frac{(t - x_i) N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - t) N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}} \tag{3.3}$$

x_i คือเวกเตอร์ปม (knot vector) ซึ่งเป็นไปตามสมการ $x_i \le x_{i+1}$ พารามิเตอร์ t มีค่าตั้งแต่ t_{min} จนถึง t_{max} ตลอดความยาวเส้นโค้ง P(t) ถ้าผลหารของพจน์ใดในสมการเป็น 0/0 ให้พจน์นั้นมีค่าเป็นศูนย์

เส้นโค้งบี-สไปลน์ที่ได้จะเป็นโพลิโนเมียลอันดับ k มีกำลัง (degree) สูงสุดเป็น k-1 **P**(t) และอนุพันธ์อันดับ 1, 2, ... , k-2 มีความต่อเนื่องตลอดความยาวของเส้นโค้ง

คุณสมบัติของเส้นโค้งบี-สไปลน์

- ผลรวมของฟังก์ชั่นพื้นฐานบี-สไปลน์สำหรับแต่ละค่าพารามิเตอร์ t ใดๆมีค่าเท่ากับ 1

$$\sum_{i=1}^{n+1} N_{i,k}(t) \equiv 1 \tag{3.4}$$

- $N_{i,k}(t)$ แต่ละพจน์มีค่าเป็นบวกหรือศูนย์สำหรับทุกค่า t : $N_{i,k}(t) \ge 0$
- เส้นโค้งมีอันดับสูงสุดกำหนดเท่ากับจำนวนของจุดยอดรูปหลายเหลี่ยม (defining polygon vertices)

---

* เนื้อความและภาพประกอบจาก Roger and Adams (1989)

- เส้นโค้งมีรูปร่างตามจุดยอดรูปหลายเหลี่ยม
- เส้นโค้งจะวางตัวอยู่ใน convex hull ของจุดยอดรูปหลายเหลี่ยม
- การกระทำการแปลง (transformation) ใด ๆ กับเส้นโค้งทำโดยกระทำกับจุดยอดรูปหลายเหลี่ยม เส้นโค้งก็จะเปลี่ยนแปลงไปตามการแปลงนั้น

ชนิดของเส้นโค้งบี-สไปลน์

เส้นโค้งบี-สไปลน์แบ่งเป็น 2 ชนิดใหญ่ ๆ คือ

1. เส้นโค้งเปิด (open curve or nonperiodic)
2. เส้นโค้งปิด (close curve or periodic)

เส้นโค้ง 2 ชนิดนี้ต่างกันที่ค่าของ x_i

เส้นโค้งเปิดมีค่า x_i ดังนี้

$$x_i = 0 \qquad 1 \le i \le k$$

$$x_i = i-k \qquad k+1 \le i \le n+1$$

$$x_i = n-k+2 \qquad n+2 \le i \le n+k+1$$

จำนวนปม (knot) ที่ใช้เท่ากับ (n+k+1) และเนื่องจาก $0 \le x_i \le n-k+2$ ทำให้ $0 \le n-k+2$ นั่นคือ จำนวนจุดยอดรูปหลายเหลี่ยมต้องมากกว่า k-2

เส้นโค้งปิดมีค่า x_i ดังนี้

$$x_i=i-1 \qquad 1 \le i \le n+2$$

$$0 \le x_i \le n+1$$

จากสมการ 3.2 - 3.3 เห็นได้ว่าฟังก์ชั่นพื้นฐานที่มีอันดับ k จะมีค่าขึ้นกับฟังก์ชั่นพื้นฐานอันดับ k-1 จนกระทั่งถึงอันดับ 1 แผนภาพแสดงการขึ้นแก่กันของฟังก์ชั่นพื้นฐาน $N_{i,k}$ เป็นดังนี้

$N_{i,k}$

$N_{i,k-1}$ $N_{i+1,k-1}$

$N_{i,k-2}$ $N_{i+1,k-2}$ $N_{i+2,k-2}$

. . .

$N_{i,1}$ $N_{i+1,1}$ $N_{i+2,1}$ $N_{i+3,1}$. $N_{i+k-1,1}$

และในทางกลับกันฟังก์ชั่นพื้นฐานอันดับ 1, $N_{i,1}$ จะมีผลต่อฟังก์ชั่นพื้นฐานอันดับที่สูงกว่าดังนี้

$N_{i-k+1,k}$. $N_{i+k-1,1}$ $N_{i,k}$ $N_{i+1,k}$. $N_{i+k-1,k}$

. . . . . .

. . . . .

$N_{i-1,2}$ $N_{i,2}$ $N_{i+1,2}$

$N_{i,1}$

คุณสมบัติของเส้นโค้งบี-สไปลน์ที่มีประโยชน์ต่อการออกแบบคือ

1. การควบคุมเส้นโค้งเฉพาะแห่งสามารถทำได้โดย

เปลี่ยนตำแหน่งของจุดควบคุม (control point)

การใช้จุดควบคุมซ้ำหลายครั้ง (multiple control point) ที่จุดเดียวกัน

การใช้กำลัง (k-1) ที่ต่างๆกันออกไป

การใช้ค่าของปมซ้ำหลายครั้ง (multiple knot value)

การเปลี่ยนตำแหน่งของจุดควบคุมจุดหนึ่งจะมีผลต่อเส้นโค้งในบริเวณส่วนย่อยรูปหลายเหลี่ยม (polygon segment) จำนวน k ด้านเท่านั้น

2. เส้นโค้งบี-สไปลน์ปิดจะผ่านจุดควบคุมจุดแรก P_1 และจุดสุดท้าย P_{n+1} ที่จุดนั้นจะมีทิศทางของเส้นสัมผัสขนานกับ (P_2-P_1) และ $(P_{n+1}-P_n)$ ตามลำดับ

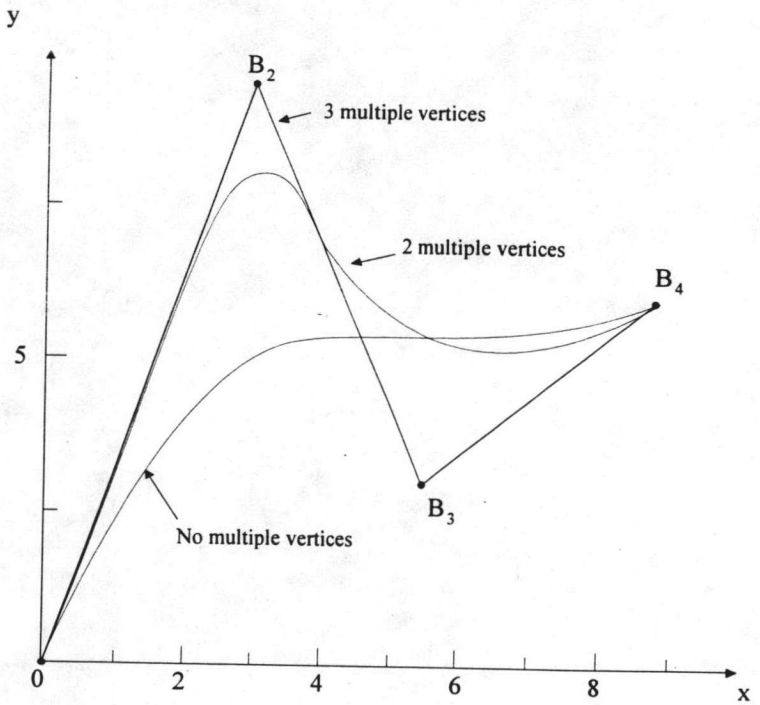
3. การใช้เส้นโค้งที่มีค่ากำลัง (k-1) ของโพลิโนเมียลมากขึ้นจะทำให้เส้นโค้งมีความตึงเพิ่มขึ้น โดยทั่วไปแล้วเส้นโค้งที่มีกำลังต่ำกว่าจะเข้าใกล้จุดควบคุมมากกว่า เมื่อให้ k=1 นั่นคือค่ากำลังเท่ากับศูนย์ เส้นโค้งจะกลายเป็นจุดควบคุม และเมื่อ k=2 เส้นโค้งก็จะกลายเป็นส่วนย่อยรูปหลายเหลี่ยม (polygon segment)

4. ในเส้นโค้งที่เป็นกำลังสอง เส้นโค้งจะสัมผัสกับจุดกึ่งกลางส่วนย่อยรูปหลายเหลี่ยมเสมอ ในกรณีที่กำลังเป็นอย่างอื่นก็จะไม่เป็นเช่นนี้

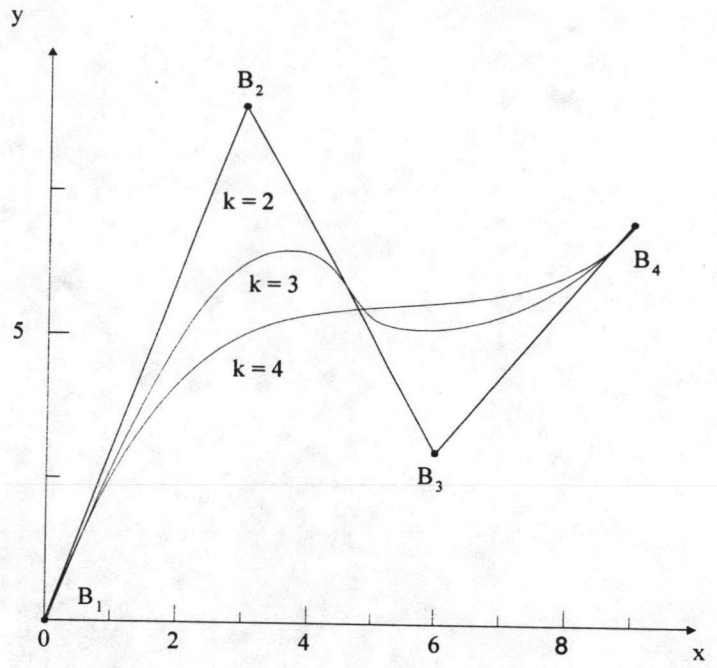
5. ถ้า k เท่ากับจำนวนของจุดควบคุม เส้นโค้งที่ได้จะกลายเป็นเส้นโค้งบีซิเออร์ (Bezier curve) และช่วงของ x_i จะอยู่ในช่วง 0 - 1

6. การใช้จุดควบคุมซ้ำหลายครั้งทำให้เกิดความโค้งมากในบริเวณนั้น สามารถทำให้เกิดมุมแหลมในเส้นโค้ง หรืออาจพูดได้ว่าการใช้จุดควบคุมซ้ำหลายครั้งทำให้เส้นโค้งถูกดึงเข้าไปใกล้จุดนั้นมากขึ้น

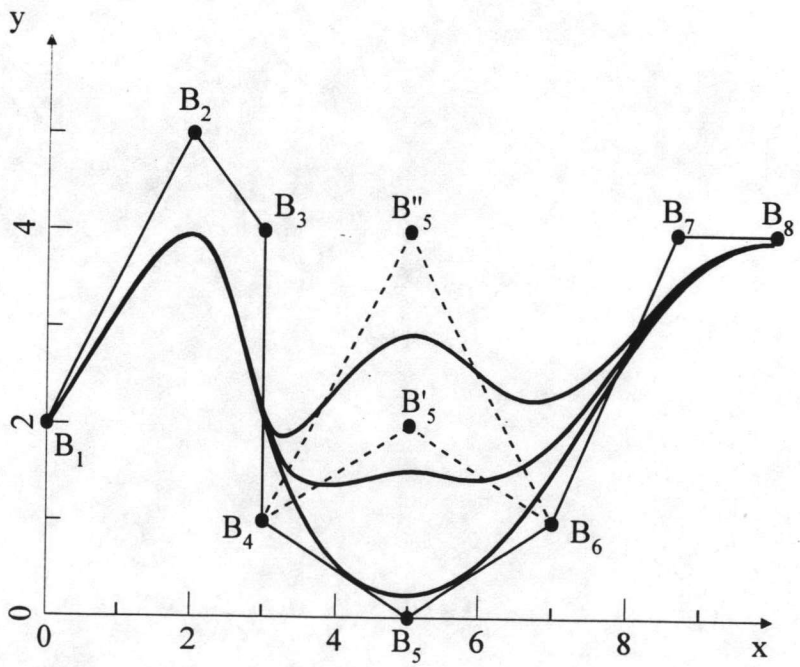
7. การใช้เส้นโค้งที่มีกำลังสูงจะยากต่อการควบคุมและการคำนวณให้เที่ยงตรง เส้นโค้งบี-สไปลน์กำลังสามก็เพียงพอแล้วต่อการใช้งานทั่วไป



รูปที่ 3.6 a ผลของการใช้จุดควบคุมหลายจุดซ้ำหลายครั้ง ในรูปเป็นเส้นโค้ง k = 4

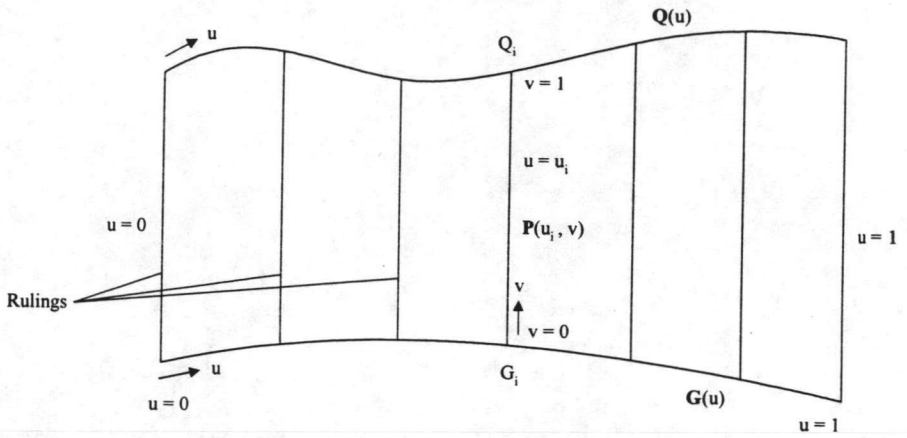


รูปที่ 3.6 b ผลของการใช้อันดับ k ต่าง ๆ



รูปที่ 3.6 c การเปลี่ยนตำแหน่งของจุดควบคุม

พื้นผิวเส้น (ruled surface)*



รูปที่ 3.7 การใช้พารามิเตอร์ของพื้นผิวเส้น

พื้นผิวเส้นสามารถสร้างได้โดยลากเส้นตรงเชื่อมจุดที่ตรงกันบนเส้นโค้งสองเส้น **G**(u) และ **Q**(u) เข้าด้วยกัน เส้นโค้ง **G**(u) และ **Q**(u) เรียกว่าราง (rail) เส้นตรงที่เชื่อมระหว่างเส้นโค้งสองเส้นเรียกว่าเส้นเชื่อม (ruling) ลักษณะสำคัญของพื้นผิวเส้นก็คือจะต้องมีเส้นตรงอย่างน้อยหนึ่งเส้นที่ผ่านจุด P(u, v) และเส้นตรงเส้นนั้นทั้งเส้นจะต้องอยู่ในพื้นผิวนี้ กรวย (cone) และทรงกระบอกเป็นตัวอย่างที่ง่ายที่สุดของพื้นผิวเส้น

การสร้างสมการพาราเมตริกของพื้นผิวเส้นจะพิจารณาเส้น $u = u_i$ ที่เชื่อมระหว่าง **G**(u) และ **Q**(u) สมการของเส้นเชื่อมนี้คือ

$$\mathbf{P}(u_i,\ v) = \mathbf{G}_i + v(\mathbf{Q}_i - \mathbf{G}_i) \qquad (3.5)$$

v เป็นพารามิเตอร์ตามความยาวของเส้นเชื่อม ขยายสมการ 3.5 สำหรับเส้นเชื่อมใด ๆ จะได้สมการพาราเมตริกของพื้นผิวเส้นที่กำหนดด้วยราง 2 เส้นคือ

$$\mathbf{P}(u,\ v) = \mathbf{G}(u) + v[\mathbf{Q}(u) - \mathbf{G}(u)] = (1-v)\mathbf{G}(u) + v\mathbf{Q}(u), \qquad (3.6)$$

$$0 \le u \le 1,\ 0 \le v \le$$

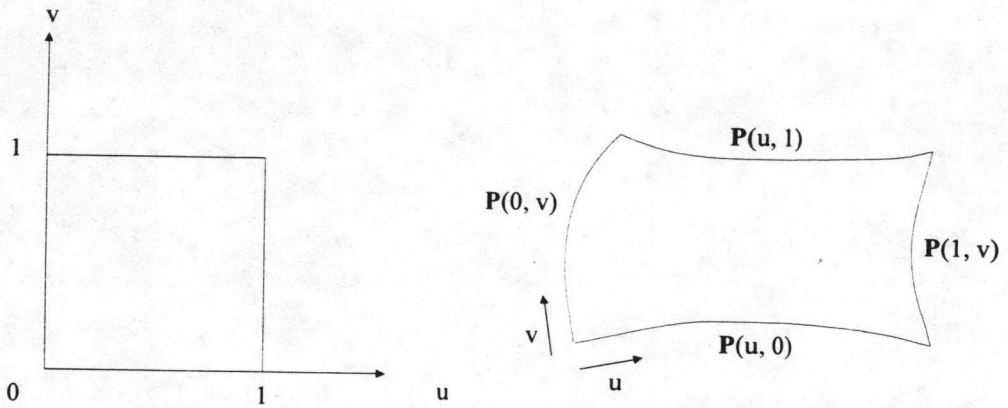
ถ้าให้ u ในสมการ 3.6 คงที่ จะได้เส้นเชื่อมในสมการ 3.5 แต่ถ้าให้ v คงที่ก็จะได้เส้นโค้งในทิศของ u ซึ่งเกิดจากการผสมเชิงเส้นของรางทั้งสอง สำหรับเส้นโค้ง v = ค่าคงที่ เมื่อพารามิเตอร์ v มีค่าเข้าใกล้ 0 **G**(u) จะมีอิทธิพลมากขึ้นในขณะที่ **Q**(u) มีอิทธิพลน้อยต่อเส้นโค้งนี้ ในทำนองเดียวกันเมื่อ v มีค่าเข้าใกล้ 1 เส้นโค้ง **Q**(u) ก็จะมีอิทธิพลมากขึ้น

---

* เนื้อความและภาพประกอบจาก Zeid (1991)

พื้นผิวคูนส์ (Coons surface)*

พื้นผิวคูนส์เป็นรูปแบบหนึ่งของการประมาณค่าภายในของข้อมูลไม่จำกัดจำนวน (transfinite interpolation) พื้นผิวจะได้จากจากประมาณค่าภายในจากจุดบนเส้นโค้งซึ่งมีจำนวนไม่จำกัด พื้นผิวคูนส์มีประโยชน์ในการสร้างพื้นที่ปิดที่เกิดจากการตัดกันของเส้นโค้ง 4 เส้นดังเช่นรูป 3.8 ในรูปแสดงเส้นโค้งที่เป็นขอบเขต ได้แก่ P(u, 0), P(1, v), P(u, 1) และ P(0, v) ในที่นี้จะสมมติให้ u และ v มีค่าอยู่ในช่วงระหว่าง 0 - 1 ตามความยาวของเส้นโค้งขอบเขต เส้นโค้งขอบเขตที่อยู่ด้านตรงข้ามกันจะต้องถูกกำหนดพารามิเตอร์ด้วยวิธีเดียวกัน



รูปที่ 3.8 ขอบเขตของพื้นผิวคูนส์

พิจารณาพื้นผิวคูนส์ผสมเชิงเส้นคู่ (bilinearly blended Coon patch) ซึ่งจะประมาณค่าภายในเส้นโค้งขอบเขตในรูป 3.8 พื้นผิวเส้นจะประมาณค่าภายในเส้นโค้งขอบเขตด้วยวิธีเชิงเส้นในทิศทางหนึ่งซึ่งแสดงด้วยสมการ 3.6 ดังนั้นการซ้อนทับกันของคู่พื้นผิวเส้นที่เชื่อมระหว่างเส้นโค้งขอบเขตสองคู่ที่เป็นด้านประชิดกันอาจทำให้ได้พื้นผิวคูนส์ที่มีขอบเขตเป็นเส้นโค้ง โดยการใช้สมการ 3.6 ในทิศทางของ u และ v เพื่อตรวจสอบข้อสันนิษฐานนี้

$$\mathbf{P}_1(u,v) = (1-u)\mathbf{P}(0,v) + u\mathbf{P}(1,v) \tag{3.7}$$

$$\mathbf{P}_2(u,v) = (1-v)\mathbf{P}(u,0) + v\mathbf{P}(u,1) \tag{3.8}$$

---

* เนื้อความและภาพประกอบจาก Zeid (1991)

บวกสมการ 3.7 และ 3.8 เข้าด้วยกันจะได้

$$\mathbf{P}(u,v) = \mathbf{P}_1(u,v) + \mathbf{P}_2(u,v) \tag{3.9}$$

พื้นผิวที่ได้จากสมการนี้จะมีขอบเขตที่ไม่ถูกต้อง คือยังไม่เป็นเส้นโค้งขอบเขต

$$\mathbf{P}(u,0) = \mathbf{P}(u,0) + [(1-u)\mathbf{P}(0,0) + u\mathbf{P}(1,0)] \tag{3.10}$$

$$\mathbf{P}(u,1) = \mathbf{P}(u,1) + [(1-u)\mathbf{P}(0,1) + u\mathbf{P}(1,1)] \tag{3.11}$$

พจน์ที่อยู่ในวงเล็บ [ ] ในสองสมการนี้เป็นส่วนที่เกินมาจากเส้นโค้งขอบเขตและควรกำจัดทิ้ง ให้ $P_3(u, v)$ แทนพื้นที่ส่วนเกินที่รวมอยู่ในสมการ 3.9 สามารถเขียนพจน์เหล่านี้เป็นการประมาณค่าภายในในทิศทางของ v ได้ดังนี้

$$\mathbf{P}_3(u,v) = (1-v)[(1-u)\mathbf{P}(0,0) + u\mathbf{P}(1,0)] + v[(1-u)\mathbf{P}(0,1) + u\mathbf{P}(1,1)] \tag{3.12}$$

ลบ $P_3(u, v)$ ซึ่งเรียกว่าผิวแก้ไข (correction surface) ออกจากสมการ 3.9 จะได้

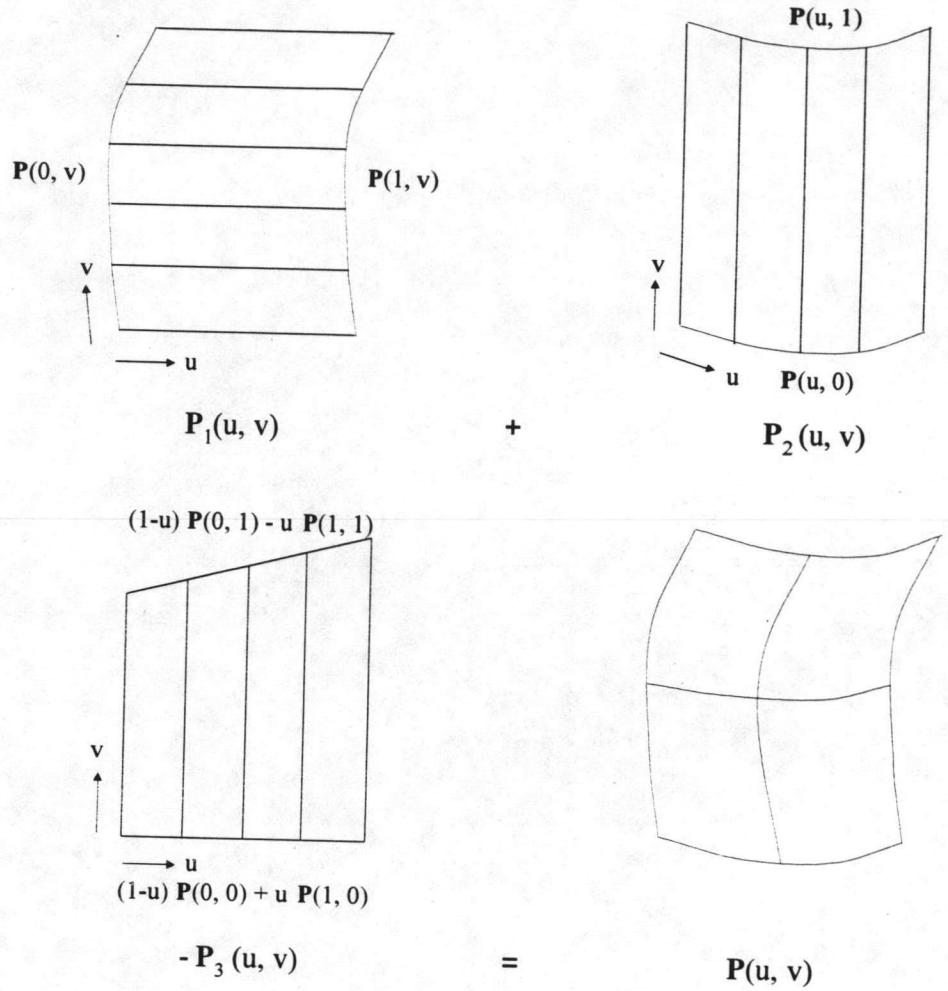
$$\mathbf{P}(u,v) = \mathbf{P}_1(u,v) + \mathbf{P}_2(u,v) - \mathbf{P}_3(u,v) \tag{3.13}$$

$$\mathbf{P}(u,v) = \mathbf{P}_1(u,v) \oplus \mathbf{P}_2(u,v) \tag{3.13}$$

เครื่องหมาย $\oplus$ คือ การบวกทางตรรกะ (boolean sum) ซึ่งเท่ากับ $P_1+P_2-P_3$ พื้นผิว P(u, v) ในสมการ 3.13 คือพื้นผิวคูนส์เชิงเส้นคู่ (bilinearly Coon patch) ที่มีเส้นโค้งขอบเขตสี่ด้านดังแสดงในรูปที่ 3.8 รูปที่ 3.9 เป็นการแสดงสมการ 3.13 ด้วยภาพ สมการ 3.13 ในรูปเมตริกซ์คือ

$$\mathbf{P}(u,v) = -\begin{bmatrix} -1 & (1-u) & u \end{bmatrix} \begin{bmatrix} 0 & \mathbf{P}(u,0) & \mathbf{P}(u,1) \\ \mathbf{P}(0,v) & \mathbf{P}(0,0) & \mathbf{P}(0,1) \\ \mathbf{P}(1,v) & \mathbf{P}(1,0) & \mathbf{P}(1,1) \end{bmatrix} \begin{bmatrix} -1 \\ 1-v \\ v \end{bmatrix} \tag{3.14}$$

แถวตั้งทางซ้ายและแถวบนของเมตริกซ์แทน $P_1(u, v)$ และ $P_2(u, v)$ ตามลำดับ กลุ่มทางขวาล่างจะเป็นกลุ่มของผิวแก้ไข $P_3(u, v)$ ฟังก์ชั่น -1, 1-u, u, 1-v และ v เรียกว่าฟังก์ชั่นผสม (blending function) เนื่องจากเป็นฟังก์ชั่นที่ผสมเส้นโค้งขอบเขตแต่ละอันเพื่อให้ได้พื้นที่ที่ต้องการ



รูปที่ 3.9 พื้นผิวคูนส์ที่ผสมโดยเชิงเส้นคู่

สมการพื้นผิวคูนส์กำลังสามคู่จะเป็นดังนี้

$$\mathbf{P}(u,v) = -\begin{bmatrix} -1 & F_1(u) & F_2(u) \end{bmatrix} \left[\begin{array}{c|cc} 0 & \mathbf{P}(u,0) & \mathbf{P}(u,1) \\ \hline \mathbf{P}(0,v) & \mathbf{P}(0,0) & \mathbf{P}(0,1) \\ \mathbf{P}(1,v) & \mathbf{P}(1,0) & \mathbf{P}(1,1) \end{array}\right] \begin{bmatrix} -1 \\ F_1(v) \\ F_2(v) \end{bmatrix} \tag{3.15}$$

ซึ่ง $F_1(x)$ และ $F_2(x)$ คือเฮอร์ไมต์โพลิโนเมียลกำลังสาม (cubic Hermite polynomial) ดังแสดงในสมการต่อไปนี้

$$F_1(x) = 2x^3 - 3x^2 + 1 \tag{3.16}$$

$$F_2(x) = -2x^3 + 3x^2 \tag{3.17}$$